A quantum circuit to find discrete logarithms on ordinary binary elliptic curves in depth $\mathrm{O}(\log^2 n)$

Martin Rötteler
Microsoft Research
One Microsoft Way
Redmond, WA 98052
martinro@microsoft.com

Rainer Steinwandt
Florida Atlantic University
Department of Mathematical Sciences
Boca Raton, FL 33431
rsteinwa@fau.edu

November 15, 2013

Abstract

Improving over an earlier construction by Kaye and Zalka [1], in [2] Maslov et al. describe an implementation of Shor's algorithm, which can solve the discrete logarithm problem on ordinary binary elliptic curves in quadratic depth $\mathrm{O}(n^2)$. In this paper we show that discrete logarithms on such curves can be found with a quantum circuit of depth $\mathrm{O}(\log^2 n)$. As technical tools we introduce quantum circuits for $\mathbb{F}_{2^n}$-multiplication in depth $\mathrm{O}(\log n)$ and for $\mathbb{F}_{2^n}$-inversion in depth $\mathrm{O}(\log^2 n)$.

1 Introduction

The practical significance of the discrete logarithm problem on ordinary binary elliptic curves (see, e. g., [3]) renders these groups a natural target for implementing Shor's algorithm [4]. To implement an actual discrete logarithm computation, efficient quantum circuits to implement the pertinent curve arithmetic are needed, and a number of authors have explored circuits to implement the relevant elliptic curve operations [1, 2, 5]. When considering a complete implementation of Shor's algorithm in such a group, Maslov et al.'s proposal in [2] shows that a quadratic depth circuit is sufficient. The reason for the quadratic depth is essentially two-fold: a double-and-add computation to compute the relevant scalar multiplications in Shor's algorithm and a finite field inversion are the dominating operations. As shown in [5,6], inversion in $\mathbb{F}_{2^n}$ can be implemented in depth $\mathrm{O}(n \log n)$, and so one may hope that the quadratic depth bound can indeed be overcome.

Our contribution. Below we show that an appropriate organization of the scalar multiplication(s) in Shor's algorithm in combination with an improved $\mathbb{F}_{2^n}$-arithmetic enables a solution to the discrete logarithm problem on ordinary binary elliptic curves in depth $\mathrm{O}(\log^2 n)$. To implement the necessary group operations we use complete binary Edwards curves as described in [5].

Structure of the paper. In the next section we briefly review some background on elliptic curves and Shor's algorithm. In particular we recall the the definition of binary Edwards curves as needed for the main part of the paper. Section 3 details how with this curve representation the addition of any two curve points can be implemented in logarithmic depth. Thereafter we discuss different options to organize the scalar

multiplications in Shor's algorithm, including a tree-based approach with polylogarithmic depth. After addressing the technical point of deriving a unique representation of group elements, in Section 4 we establish our main result.

2 Technical tools

This section reviews some known results on ordinary binary elliptic curves and on computing discrete logarithms with Shor's algorithm.

2.1 Quantum circuits for binary elliptic curve arithmetic

For $n \in \mathbb{N}$ a positive integer, we denote by $\mathbb{F}_{2^n}$ a finite field of size 2^n—for a cryptographic application, e. g., a digital signature scheme, a typical choice would be $n = 163$ or $n = 233$ [3]. To represent elements in $\mathbb{F}_{2^n}$ we use a polynomial basis representation. In other words we fix an irreducible polynomial $p \in \mathbb{F}_2[x]$ with coefficients in the integers modulo 2 and identify $\mathbb{F}_{2^n}$ with the quotient $\mathbb{F}_2[x]/(p)$, so that each $\alpha \in \mathbb{F}_{2^n}$ has a unique expression of the form $\alpha = \sum_{i=0}^{n-1} \alpha_i \cdot (x^i + (p))$ with $(\alpha_0, \ldots, \alpha_{n-1}) \in \mathbb{F}_2^n$.

Using a *short Weierstrass form*, each ordinary binary elliptic curve can be represented by a polynomial equation

$$y^2 + xy = x^3 + a_2 x^2 + a_6, \tag{1}$$

where $a_2, a_6 \in \mathbb{F}_{2^n}$ with $a_6 \neq 0$ [7, Chapters 13.1.4 and 13.1.5]. More precisely, the elliptic curve represented by Equation (1) consists of the points $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n}) := \{(u, v) \in \mathbb{F}_{2^n} : v^2 + uv = u^3 + a_2 u^2 + a_6\} \cup \{\mathcal{O}\}$, where $\mathcal{O} \in \mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n})$ is the unique projective point that is obtained when homogenizing Equation (1). Hasse's bound implies that the size of $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n})$ differs from 2^n by no more than $2^{1+(n/2)} + 1$, and the subgroups of $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n})$ considered in cryptographic applications typically have a very small cofactor. Hence n is a natural parameter to measure the complexity of a quantum circuit to solve the discrete logarithm problem in a $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n})$.

The set $\mathrm{E}_{a_2,a_6}(\mathbb{F}_{2^n})$ has a group structure, but implementing this group law directly comes at a certain inconvenience: case distinctions have to be made, which require the implementation of a (nested) if-then-else statement (cf. the discussion in [5, 8]). To avoid this issue, subsequently we use *complete binary Edwards curves* as introduced by Bernstein et al. [9]. For $n \geq 3$ each ordinary elliptic curve is birationally equivalent to a complete binary Edwards curve, and we can represent such a curve by an equation

$$d_1(x + y) + d_2(x^2 + y^2) = xy + xy(x + y) + x^2 y^2 \tag{2}$$

with $d_1 \in \mathbb{F}_{2^n}^*$ being non-zero, $d_2 \in \mathbb{F}_{2^n}$ and $\mathrm{Tr}(d_2) = 1$.[1] By $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ we denote the points in $\mathbb{F}_{2^n}^2$ satisfying Equation (2). The group law on $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ is given by the formula

$$\begin{aligned} x_3 &= \frac{d_1(x_1 + x_2) + d_2(x_1 + y_1)(x_2 + y_2) + (x_1 + x_1^2)(x_2(y_1 + y_2 + 1) + y_1 y_2)}{d_1 + (x_1 + x_1^2)(x_2 + y_2)} \text{ and} \\ y_3 &= \frac{d_1(y_1 + y_2) + d_2(x_1 + y_1)(x_2 + y_2) + (y_1 + y_1^2)(y_2(x_1 + x_2 + 1) + x_1 x_2)}{d_1 + (y_1 + y_1^2)(x_2 + y_2)}, \end{aligned}$$

where $(x_1, y_1), (x_2, y_2) \in \mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ can be arbitrary curve points—including the identity element $(0, 0)$. To derive efficient addition formula one can (similarly as for a Weierstrass form) pass to projective

[1] The condition $\mathrm{Tr}(d_2) = 1$ can equivalently be expressed as $\sum_{i=0}^{n-1} d_2^{2^i} = 1$.

coordinates. Bernstein et al. [9] show that from projective representations (X_1, Y_1, Z_1), (X_2, Y_2, Z_2) of two points one can derive a projective representation (X_3, Y_3, Z_3) of their sum by means of 21 multiplications in $\mathbb{F}_{2^n}$, four multiplications by one of the constant d_1, d_2, one squaring and 15 additions in $\mathbb{F}_{2^n}$.

$$\begin{array}{lllllllllllll} W_1 & = & X_1 + Y_1, & W_2 & = & X_2 + Y_2, & A & = & X_1 \cdot (X_1 + Z_1), & B & = & Y_1 \cdot (Y_1 + Z_1), \\ C & = & Z_1 \cdot Z_2, & D & = & W_2 \cdot Z_2, & E & = & d_1 C^2, & H & = & (d_1 Z_2 + d_2 W_2) \cdot W_1 \cdot C, \\ I & = & d_1 Z_1 \cdot C, & U & = & E + A \cdot D, & V & = & E + B \cdot D, & S & = & U \cdot V, \end{array}$$

$$\begin{array}{lll} X_3 & = & S \cdot Y_1 + (H + X_2 \cdot (I + A \cdot (Y_2 + Z_2))) \cdot V \cdot Z_1, \\ Y_3 & = & S \cdot X_1 + (H + Y_2 \cdot (I + B \cdot (X_2 + Z_2))) \cdot U \cdot Z_1, \\ Z_3 & = & S \cdot Z_1. \end{array}$$

From an asymptotic point of view, it suffices to observe that the number of field operations is constant. It is not necessary, however, to perform these field operations sequentially, and [5] suggest some parallelization, establishing the following upper bound for the depth of a point addition circuit, where $D_M(n)$ stands for the depth of an $\mathbb{F}_{2^n}$-multiplier $|\alpha\rangle\ |\beta\rangle\ |\gamma\rangle \mapsto |\alpha\rangle\ |\beta\rangle\ |\gamma + \alpha\beta\rangle$.

Proposition 2.1 ([5, Proposition 3.3]). *Let (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) be projective representations of two (not necessarily different) points $P_1, P_2 \in \mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$. Then the addition map*

$$|X_1\rangle\ |Y_1\rangle\ |Z_1\rangle\ |X_2\rangle\ |Y_2\rangle\ |Z_2\rangle\ |0\rangle\ |0\rangle\ |0\rangle \longrightarrow |X_1\rangle\ |Y_1\rangle\ |Z_1\rangle\ |X_2\rangle\ |Y_2\rangle\ |Z_2\rangle\ |X_3\rangle\ |Y_3\rangle\ |Z_3\rangle,$$

where (X_3, Y_3, Z_3) is a projective representation of $P_1 + P_2$, can be implemented in depth $5 \cdot D_M(n) + 4 \cdot \max(D_M(n), 2n) + \mathrm{O}(1)$.

From [2] it follows that we can choose $D_M(n) \in \mathrm{O}(n)$, and in Section 3.1 we will show that through a suitable use of trees $D_M(n)$ can be chosen to be of logarithmic depth. As the number of field operations to add to curve points is constant, this establishes immediately the existence of a logarithmic depth circuit for point addition. To optimize the circuit depth, we can exploit the bound from Proposition 2.1: looking into the proof of [5, Proposition 3.3], one recognizes that the term $2n$ occurring as argument of the maximum in Proposition 2.1 describes the multiplication of a binary $n \times n$-matrix with a binary vector. In the next section we will see that such a multiplication can be realized in logarithmic depth as well.

A technical issue that we address in Section 3.2.3 is the derivation of the unique (affine) representation from a projective representation of a curve point: The natural way to realize this is by means of an inversion in $\mathbb{F}_{2^n}$, but for none of the division circuits described in [1, 2, 5] a polylogarithmic depth bound is available. We modify the construction in [5] to achieve polylogarithmic depth.

2.2 Shor's algorithm

For our discussion we assume that a generator $P \in \mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ of a cyclic subgroup of $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ is fixed and the order $\mathrm{ord}(P)$ of this group generator is known. Moreover, we assume that a group element Q in the subgroup generated by P is fixed; our goal is to find the unique integer $r \in \{1, \ldots, \mathrm{ord}(P)\}$ such that $Q = r \cdot P$. The algorithm proceeds as follows. First, two registers of length $n + 1$ qubits[2] are created and each qubit is initialized in the $|0\rangle$ state. Then a Hadamard transform H is applied to each qubit, resulting in the state $\frac{1}{2^{n+1}} \sum_{k,\ell=0}^{2^{n+1}-1} |k, \ell\rangle$. Next, conditioned on the content of the register holding the label k or ℓ, we add the corresponding multiple of P and Q, respectively, i. e., we implement the map

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

[2] Hasse's bound guarantees that $\mathrm{ord}(P)$ can be represented with $n + 1$ bits.

Hereafter, the third register is discarded and a quantum Fourier transform $\mathrm{QFT}_{2^{2\cdot(n+1)}}$ on $2\cdot(n+1)$ qubits is computed. Finally, the state of the first two registers—which hold a total of $2\cdot(n+1)$ qubits—is measured. As shown in [4, 10], the factor r can be computed from this measurement data via classical post-processing. The corresponding quantum circuit is shown in Figure 1. In the following sections, we will be concerned with parallelizing the parts of the circuit in this figure. In Section 3.2.3 we will address the problem of having a non-unique representation of curve points, as the above description of Shor's algorithm implicitly assumes group elements to have a unique representation.

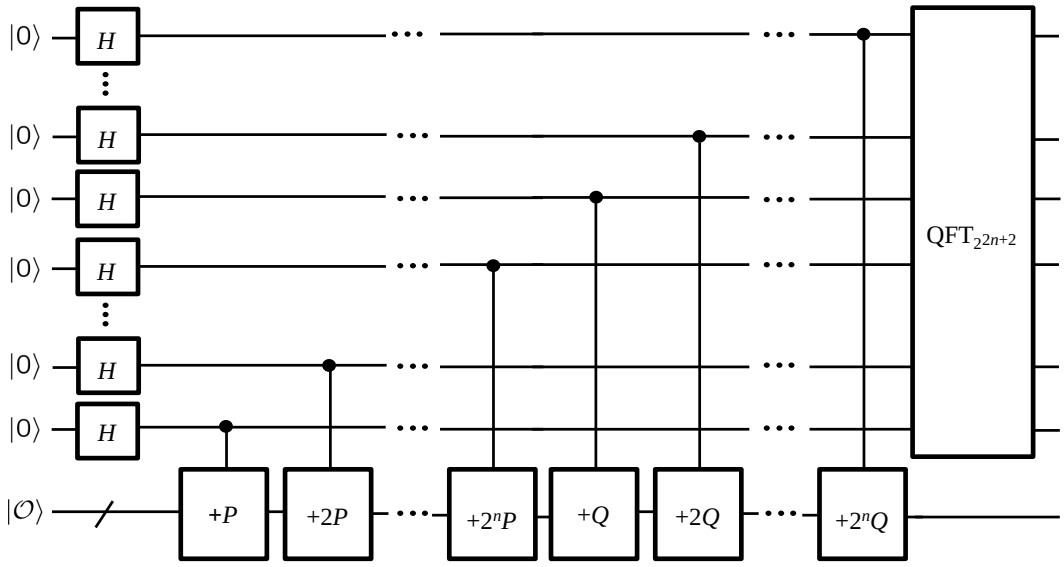

Figure 1: Shor's algorithm to compute the discrete logarithm in the subgroup of an elliptic curve generated by a point P. The input to the problem is a point Q, and the task is to find $r \in \{1, \ldots, \mathrm{ord}(P)\}$ such that $Q = r \cdot P$. The circuit naturally decomposes into three parts, namely (i) the Hadamard layer on the left, (ii) a double scalar multiplication (in this figure implemented as a cascade of point additions), and (iii) the quantum Fourier transform QFT at the end. We show that each of these parts can be implemented in a circuit depth of $\mathrm{O}(\log^2 n)$ to obtain the main result of this paper.

3 Parallelizing Shor's algorithm

To reduce the circuit depth, we parallelize Shor's algorithm on two different levels: (i) the computation of $\mathbb{F}_{2^n}$-multiplications is parallelized and (ii) the computation of the scalar products $k \cdot P + \ell \cdot Q$ is parallelized.

3.1 Multiplying $\mathbb{F}_{2^n}$-elements in depth $\mathrm{O}(\log n)$

A simple observation that will be useful is that we can implement the map

$$|x\rangle \underbrace{|0\rangle \dots |0\rangle}_{m} \longmapsto |x\rangle \underbrace{|x\rangle \dots |x\rangle}_{m} \quad (x \in \{0,1\})$$

in depth $\mathrm{O}(\log m)$ by arranging m CNOT gates as a tree. Figure 2, which derives from [11, Figure 2], shows such a 'multi-fan-out CNOT with $|0\rangle$-input' for the case $m = 7$. We note that in general such a tree is not functionally equivalent to a CNOT with fan-out greater than 1, but for the case of a $|0\rangle$-input this equivalence holds, and for our purposes this is the only case needed.

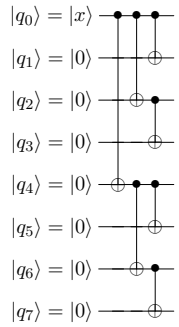

Figure 2: obtaining m 'copies' of a qubit in depth $\mathrm{O}(\log m)$

As starting point to implement multiplication in $\mathbb{F}_{2^n}$ we use the circuit proposed by Maslov et al. in [2]—which builds on a classical Mastrovito multiplier [12–14]. This construction reduces the task of multiplying two elements in $\mathbb{F}_2[x]/(p)$ to implementing a quantum circuit that evaluates two matrix-vector multiplications with a Toeplitz matrix, one matrix-vector multiplication with a matrix that depends only on the polynomial p, and an addition in $\mathbb{F}_2^n$. More specifically, the coefficients $(\gamma_0, \dots, \gamma_{n-1})$ of the product $(\sum_{i=0}^{n-1} \alpha_i \cdot (x^i + (p))) \cdot (\sum_{i=0}^{n-1} \beta_i \cdot (x^i + (p)))$ are obtained as follows, where $M \in \mathbb{F}_2^{n \times (n-1)}$ is independent of the specific field elements to be multiplied; the matrix M depends only on the irreducible polynomial p defining the underlying finite field $\mathbb{F}_2[x]/(p)$:

$$\vec{\gamma} = \underbrace{\begin{pmatrix} \alpha_0 & 0 & \dots & 0 & 0 \\ \alpha_1 & \alpha_0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2} & \alpha_{n-3} & \dots & \alpha_0 & 0 \\ \alpha_{n-1} & \alpha_{n-2} & \dots & \alpha_1 & \alpha_0 \end{pmatrix}}_{=L} \cdot \vec{\beta} + M \cdot \underbrace{\begin{pmatrix} 0 & \alpha_{n-1} & \alpha_{n-2} & \dots & \alpha_2 & \alpha_1 \\ 0 & 0 & \alpha_{n-1} & \dots & \alpha_3 & \alpha_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \alpha_{n-1} & \alpha_{n-2} \\ 0 & 0 & 0 & \dots & 0 & \alpha_{n-1} \end{pmatrix}}_{=U} \cdot \vec{\beta} \qquad (3)$$

3.1.1 Computing the products $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$

To implement the multiplications of L and U with $\vec{\beta}$ respectively, we first observe that—considering both the computation of $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$ combined—each coefficient α_i ($i = 0, \dots, n-1$) occurs in exactly n products of the form $\alpha_i \beta_j$. Similarly, each coefficient β_j ($j = 0, \dots, n-1$) occurs in a total of exactly n products. We want to compute *all* of these $\mathbb{F}_2$-products in parallel. So we ensure that n 'copies' of each of

$\beta_0, \ldots, \beta_{n-1}$ are available, using a 'multi-fan-out CNOT with $|0\rangle$-input' for each β_i. As the CNOT trees for β_i and β_j with $i \neq j$ operate on disjoint wires, they can be executed in parallel and implemented in depth $\mathrm{O}(\log n)$. Analogously, using a 'multi-fan-out CNOT with $|0\rangle$-input' for each of $\alpha_0, \ldots, \alpha_{n-1}$ we can—in depth $\mathrm{O}(\log n)$ and in parallel to the trees for copying the β_i-values—provide n 'copies' of each of $\alpha_0, \ldots, \alpha_{n-1}$.

Having, at the cost of $\mathrm{O}(n^2)$ qubits, all these copies at our disposal, we can now, in depth 1, compute all products $\alpha_i \cdot \beta_j$ that are necessary to find $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$ in parallel, using n^2 Toffoli gates. Having evaluated all these products we can simply compute each entry of $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$ by using a a depth $\mathrm{O}(\log n)$ addition tree for each entry of these two vectors. Only CNOT gates (and no further ancillae) are needed for this. Figure 3 shows an example for the case $n = 2$, i. e., we have

$$L = \begin{pmatrix} \alpha_0 & 0 \\ \alpha_1 & \alpha_0 \end{pmatrix}, U = \begin{pmatrix} 0 & \alpha_1 \end{pmatrix}.$$

In this case all occurring multiplications can be evaluated in depth $1 = \log_2(2)$, and the final addition trees reduce to a single CNOT gate to compute the 'last' entry $\alpha_1\beta_0 + \alpha_0\beta_1$ of $L \cdot \vec{\beta}$.

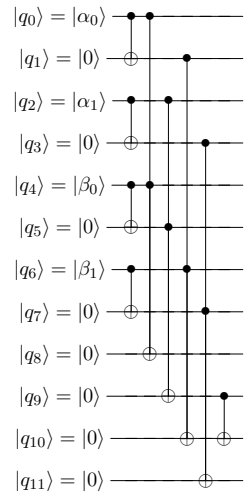

Figure 3: computing $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$ for $n = 2$: all four multiplications occur in parallel

3.1.2 Multiplication by the constant matrix M

From Equation (3), we see that the vector $\vec{\eta} = U \cdot \vec{\beta} \in \mathbb{F}_2^{n-1}$ needs to be multiplied from the left with the fixed $n \times (n-1)$-matrix M. Writing $\mathrm{hw}(M_i)$ for the Hamming weight of the i^{th} column of M and denoting by η_i the i^{th} entry of $\vec{\eta}$, we first create $\mathrm{hw}(M_i) \leq n$ copies of η_i ($i = 1, \ldots, n-1$), requiring $\mathrm{O}(n^2)$ qubits. For this we use again 'multi-fan-out CNOT gates with $|0\rangle$-input' that operate in parallel and can be realized in depth $\mathrm{O}(\log n)$. This allows us to compute all n entries of $M \cdot \vec{\eta}$ in parallel: for each entry of the result we can use an $\mathrm{O}(\log n)$-depth addition tree that computes the scalar product of the corresponding row of M with $\vec{\eta}$. As the matrix M is fixed, this can be done by means of CNOT gates. Figure 4 shows a 'worst case tree' of depth $3 = \log_2(8)$ for the case $n - 1 = 8$: multiplying a matrix row consisting entirely of 1s with $(\eta_0, \ldots, \eta_7)$.

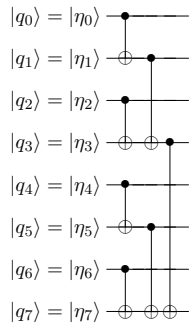

Figure 4: computing $(1,1,1,1,1,1,1,1) \cdot (\eta_0, \eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6, \eta_7)^{\mathrm{t}}$ in depth 3

Finally, to complete the evaluation of Equation (3), we add the binary vector $L \cdot \vec{\beta}$ to $M \cdot (U \cdot \vec{B})$ by means of n CNOT gates that operate in parallel. With all involved steps—computing $L \cdot \vec{\beta}$ and $U \cdot \vec{\beta}$, finding $M \cdot U\vec{\beta}$, and determining $L\vec{\beta} + MU\vec{\beta}$—being realizable in depth $\mathrm{O}(\log n)$ we obtain the following result.

Theorem 3.1 ($\mathbb{F}_{2^n}$-multiplication in logarithmic depth)**.** *There is a polynomial-size quantum circuit of depth* $\mathrm{O}(\log n)$ *which on input polynomial basis representations of* $\alpha, \beta \in \mathbb{F}_{2^n}$ *computes a polynomial basis representation of the product* $\alpha \cdot \beta \in \mathbb{F}_{2^n}$.

The above-described technique to multiply a vector with the fixed matrix M can also be used to implement other matrix-vector multiplications: Given a binary $n \times n$ matrix $A \in \mathbb{F}_2^{n \times n}$ and a vector $\vec{\gamma} \in \mathbb{F}_2^n$, we first use n 'multi-fan-out CNOT with $|0\rangle$-input' to create n copies of $\vec{\gamma}$, investing $\mathrm{O}(n^2)$ qubits. Handling all entries of $\vec{\gamma}$ in parallel, this can be done in depth $\mathrm{O}(\log n)$. Hereafter we can use an addition tree to compute the necessary n scalar products in parallel, just as in the discussion of the matrix M. We can apply this observation to the matrix multiplications occurring in the proof of Proposition 2.1 given in [5], which replaces the argument $2n$ of $\max$ with a function in $\mathrm{O}(\log n)$. In combination with Theorem 3.1, we obtain the following.

Corollary 3.1. *Using projective coordinates, a projective representation of the sum of two points on a complete binary Edwards curve* $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ *can be computed in depth* $\mathrm{O}(\log n)$.

3.2 Organizing the computation of $k \cdot P + \ell \cdot Q$

An essential part of an implementation of Shor's algorithm is a circuit which on input (binary representations of) $k = \sum_{i=0}^{n} k_i 2^i$ and $\ell = \sum_{i=0}^{n} \ell_i 2^i$ computes a (unique) representation of $k \cdot P + \ell \cdot Q$—because of Hasse's bound, we can assume that k and ℓ are represented with (at most) $n+1$ qubits each.

3.2.1 Sequential double-and-add

The approach taken in [2, 15] can be seen as implementation of a right-to-left version of the double-and add-algorithm[3]:

[3]More specifically, Maslov et al. first perform all necessary additions of the points $2^i \cdot P$ and then continue with the necessary additions of the points $2^i \cdot Q$; this change of addition order does not affect the circuit depth.

$R \leftarrow \mathcal{O}$ # initialize result to identity
for $i = 0$ to n step 1
  if $k_i = 1$ then $R \leftarrow R + 2^i \cdot P$
  if $\ell_i = 1$ then $R \leftarrow R + 2^i \cdot Q$
return R # $R = kP + \ell Q$

Applying Corollary 3.1, each point addition requires circuit depth $\mathrm{O}(\log n)$, and so we immediately obtain an $\mathrm{O}(n \log n)$-depth circuit to compute $kP + \ell Q$. A feature of this particular strategy is that all points $2^i P$ and $2^i Q$ can be precomputed classically, and in all adders involved one argument is constant. This can be exploited to simplify the addition circuits, but for a realistic value of n several hundred different addition circuits are involved. A left-to-right formulation of the double-and-add algorithm offers an alternative that involves only three types of circuits, but this uniformity comes at the cost of a general doubling circuit:

$R \leftarrow \mathcal{O}$ # initialize result to identity
if $k_n = 1$ then $R \leftarrow R + P$ # adjust starting value based on most significant bit
if $\ell_n = 1$ then $R \leftarrow R + Q$
for $i = n - 1$ to 0 step -1
  $R \leftarrow 2 \cdot R$
  if $k_i = 1$ then $R \leftarrow R + P$
  if $\ell_i = 1$ then $R \leftarrow R + Q$
return R # $R = kP + \ell Q$

This algorithm processes k and ℓ simultaneously, so that the total number of doublings is n (rather than $2n$)—the latter technique goes back to Straus [16] and is also known as *Shamir's trick* (cf. [17]). Realizing point doubling and addition of P respectively Q in depth $\mathrm{O}(\log n)$, this yields again an $\mathrm{O}(n \log n)$-depth circuit to find $kP + \ell Q$.

3.2.2 Parallelized double-and-add

To find $kP + \ell Q$ with a quantum circuit of smaller depth, we parallelize the computation of $kP + \ell Q$. To simplify the description we assume that the length $n+1$ of the binary representation of k and of ℓ is a power of 2—if this is not the case, the binary expansions can be padded with 0s accordingly.

$(R_0^{(\log_2(n+1))}, \ldots, R_n^{(\log_2(n+1))}) \leftarrow (k_0 \cdot 2^0 P, \ldots, k_n \cdot 2^n P)$
$(S_0^{(\log_2(n+1))}, \ldots, S_n^{(\log_2(n+1))})) \leftarrow (\ell_0 \cdot 2^0 S, \ldots, \ell_n \cdot 2^n S)$
for $i = \log_2(n+1) - 1$ to 0 step -1
  $(R_0^{(i)}, \ldots, R_j^{(i)}, \ldots, R_{2^i-1}^{(i)}) = (R_0^{(i+1)} + R_1^{(i+1)}, \ldots, R_{2j}^{(i+1)} + R_{2j+1}^{(i+1)}, \ldots, R_{2^{i+1}-2}^{(i+1)} + R_{2^{i+1}-1}^{(i+1)})$
  $(S_0^{(i)}, \ldots, S_j^{(i)}, \ldots, S_{2^i-1}^{(i)}) = (S_0^{(i+1)} + S_1^{(i+1)}, \ldots, S_{2j}^{(i+1)} + S_{2j+1}^{(i+1)}, \ldots, S_{2^{i+1}-2}^{(i+1)} + S_{2^{i+1}-1}^{(i+1)})$
return $R_0^{(0)} + S_0^{(0)}$

The summation computed here is essentially the same as in the first sequential version of the double-and-add algorithm we discussed, therewith following the approach in [2, 15]. Like Maslov et al. we parallelize the execution of gates that operate on disjoint wires, but we process more than one point $2^i P$ respectively $2^j Q$ at a time. The (affine) points $2^0 P, \ldots, 2^n P$ and $2^0 Q, \ldots, 2^n Q$ can be precomputed classically, and the leaves can be initialized with a sequence of CNOT gates, conditioned on the individual bits of the binary representation of k and ℓ. Using 'multi-fan-out CNOTs with $|0\rangle$-input', we can create $2n$ copies of the binary

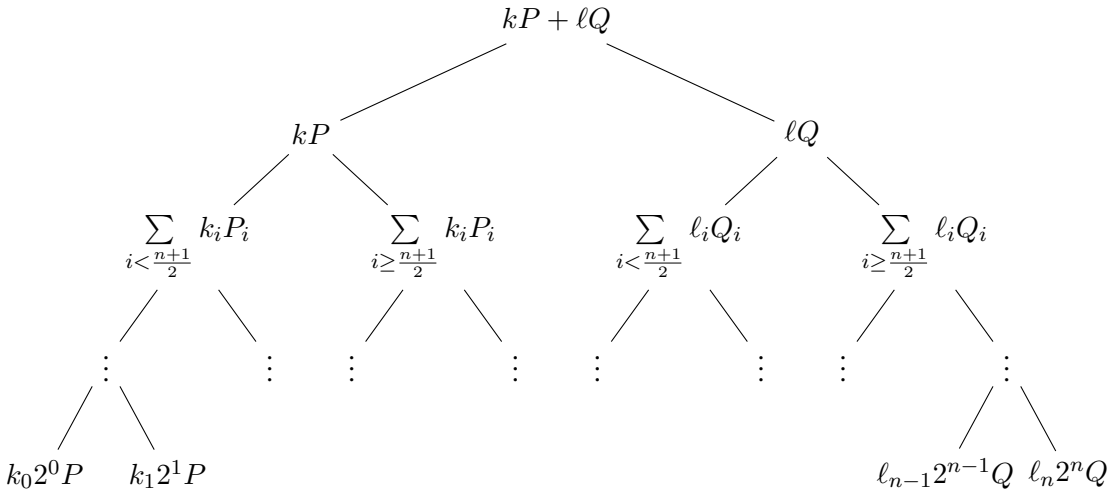

Figure 5: computing $kP + \ell Q$: parallelized double-and-add

representation of k and $2n$ copies of the binary representation of ℓ in depth $\mathrm{O}(\log n)$. Each leaf corresponds to a separate quantum register of linear length, initialized with $|\mathcal{O}\rangle = |(0,0,1)\rangle$, and hence the CNOT gates for different points operate on disjoint wires. Having $2n$ copies of each coefficient k_i and ℓ_j available, we can copy all $2n$ qubits representing a point $2^i P$ or $2^j Q$ on $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ in constant depth.[4] In particular, the complete initialization can be realized in depth $\mathrm{O}(\log n)$.

To proceed from one tree level to the next, a layer of addition circuits is used that operate in parallel, each circuit adding two curve points. Realizing each such addition circuit in logarithmic depth as in Corollary 3.1 results in a quantum circuit of depth $\mathrm{O}(\log^2 n)$ to compute $kP + \ell Q$. Figure 5 shows the resulting tree structure to compute $kP + \ell Q$. We obtain the following lemma.

Lemma 3.1. *Using projective coordinates, for a complete binary Edwards curve the map*

$$|k\rangle\ |\ell\rangle\ |0\rangle \mapsto |k\rangle\ |\ell\rangle\ |kP + \ell Q\rangle,$$

where $0 \leq k, \ell < 2^{n+1}$*, can be realized in depth* $\mathrm{O}(\log^2 n)$.

3.2.3 Deriving a unique point representation

In our discussion of the elliptic curve arithmetic we focused on the use of projective coordinates, as this avoids the use of a (costly) $\mathbb{F}_{2^n}$-inversion. We pay for this by a non-unique point representation, however. To pass from projective coordinates $(X_1, Y_1, Z_1) \in \mathbb{F}_{2^n}^3$ for a point on $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$ to the unique affine coordinates $(X_1/Z_1, Y_1/Z_1)$, it is sufficient to compute the inverse of $Z_1 \in \mathbb{F}_2^*$ followed by two multiplications in $\mathbb{F}_{2^n}$. From Theorem 3.1 we know that these two multiplications can be implemented in depth $\mathrm{O}(\log n)$, and we are left with the task of computing Z_1^{-1}. In [5] a depth $\mathrm{O}(n \log n)$ circuit is proposed for this task which builds on a classical algorithm by Itoh ans Tsuji [18]. This algorithm consists of two main parts followed by a final squaring in $\mathbb{F}_{2^n}$.

The two main parts require a total of $\mathrm{O}(\log n)$ multiplications in $\mathbb{F}_{2^n}$ and $\mathrm{O}(\log n)$ exponentiations with fixed powers of 2. The latter maps are bijective and $\mathbb{F}_2$-linear and can be implemented as matrix-vector

[4] The 'projective coordinate' of $2^i P$ and $2^j Q$ is always 1.

multiplications with a fixed matrix. Consequently all necessary computations in these two parts can be implemented in depth $\mathrm{O}(\log^2 n)$. The final squaring operation can again be realized as a matrix-vector multiplication with a fixed matrix, and so the complete inversion can be implemented in depth $\mathrm{O}(\log^2 n)$.

Theorem 3.2. *There is a polynomial-size quantum circuit of depth* $\mathrm{O}(\log^2 n)$ *which on input a polynomial basis representation of* $\alpha \in \mathbb{F}_{2^n}^*$ *computes a polynomial basis representation of the inverse* $\alpha^{-1} \in \mathbb{F}_{2^n}^*$.

Implementing an $\mathbb{F}_{2^n}$-inverter in this way and combining it with two multiplication circuits as in Theorem 3.1, we can derive the unique affine representation of a curve point in polylogarithmic depth. More specifically, we have the following.

Corollary 3.2. *There is a polynomial-size quantum circuit of depth* $\mathrm{O}(\log^2 n)$ *which on input a projective representation* (X_1, Y_1, Z_1) *of a point on* $\mathrm{E}_{\mathrm{B},d_1,d_2}(\mathbb{F}_{2^n})$*, returns the affine representation* $(X_1/Z_1, Y_1/Z_1)$ *of this point.*

4 Computation of a discrete logarithm in depth $\mathrm{O}(\log^2 n)$

We now have all the pieces in place to establish our main result.

Corollary 4.1. *Shor's algorithm to compute the discrete logarithm on a complete binary Edwards curve can be implemented using a quantum circuit of polynomial size and depth* $\mathrm{O}(\log^2 n)$.

Proof: Referring to the three stages as in Figure 1, we first note that the Hadamard gates on the left can be implemented in depth 1. Next, as shown in Subsection 3.2.2, we can implement the operation that computes $kP+\ell Q$ in depth $\mathrm{O}(\log^2 n)$ with a circuit of size polynomial in n. In order to complete the proof, we have to bound the depth of the Fourier transform $\mathrm{QFT}_{2^{2n+2}}$. For this, we use a result shown in [19] that the Fourier transform on m qubits with target error ε can be implemented with a circuit of depth $\mathrm{O}(\log m + \log\log 1/\varepsilon)$ and polynomial size. Choosing $m = n + 1$ and $\varepsilon = 1/2^{2m}$ is sufficient for Shor's algorithm to find the discrete logarithm with constant probability of success [4], i. e., we can upper bound the depth of the QFT by $\mathrm{O}(\log n)$ and hence the overall depth of the circuit by $\mathrm{O}(\log^2 n)$. □

5 Conclusion and outlook

The above discussion shows that the depth $\mathrm{O}(n^2)$ bound for discrete logarithm computations on ordinary binary elliptic curves can be improved exponentially: using parallelization at multiple levels, we can implement Shor's algorithm on complete binary Edwards curves in depth $\mathrm{O}(\log^2 n)$. To show our result we introduced the first sublinear-depth circuits for $\mathbb{F}_{2^n}$-multiplication and $\mathbb{F}_{2^n}$-inversion, which may be of independent interest. The price for the exponential reduction in depth is the introduction of ancillae—a parallelized $\mathbb{F}_{2^n}$-multiplier as discussed in Section 3.1 adds $\mathrm{O}(n^2)$ additional qubits. Depending on the number of qubits available, for cryptographically significant parameters, say $n \geq 163$, trade-offs that put less emphasis on depth-optimization could be interesting. For instance, one could combine the parallelized double-and-add computation with a linear-depth field arithmetic. Even though not being depth-optimal, thousands of qubits could be saved in this way. One could also try to avoid the general point addition circuits in the parallelized double-and-add procedure (which involve many field multiplications) and instead rely on a sequential depth $\mathrm{O}(n \log n)$ solution with specialized point addition circuits as discussed in Section 3.2.1. It is not clear at this point how the best trade-off for a large-scale discrete logarithm computation looks like.

It is interesting to compare our findings for bounding the depth of Shor's algorithm over an additive group on an ordinary binary elliptic curve to the case of factoring. In both cases the high-level structure—namely, phase estimation over an abelian group—of the algorithm is the same, however, the details on how to implement the arithmetic differ significantly. In the case of the elliptic curve arithmetic, the bottle neck are the addition formulae for points on the curve which involve finite field divisions. Our polylogarithmic depth implementation of Shor's algorithm may be compared with the findings of [20] in which an $\mathrm{O}(\log^2 n)$ upper bound on the depth for Shor's algorithm for integer factorization was given for a 2D nearest neighbor architecture and to [21], where a circuit of depth $\mathrm{O}(n^2)$ was derived, albeit on a 1D nearest neighbor architecture. The circuits presented in this paper assume an arbitrary coupling model and we leave it as an open problem whether they can be adapted to a 2D nearest neighbor architecture while maintaining the same upper bound on the circuit depth.

To enable parallelization, we make intensive use of 'multi-fan-out CNOTs with $|0\rangle$-input', and one may hope for further depth improvements if multi-fan-out gates are provided. At the moment we do not know how to derive an asymptotic benefit of such gates that goes beyond a constant factor improvement. The remaining bottleneck for an asymptotic improvement in the $\mathbb{F}_{2^n}$-arithmetic seems the logarithmic depth for parity computations. Finally, Amy et al.'s [22] observation that $\mathbb{F}_{2^n}$-multiplication can be realized in T-depth 2 suggests another natural direction for follow-up research: minimizing T-depth and the number of T-gates necessary to compute discrete logarithms on binary elliptic curves.

Acknowledgments. We thank the reviewers for their constructive feedback and we thank Schloss Dagstuhl, Germany, for providing an excellent environment for part of this research through a *Quantum Cryptanalysis* seminar. RS was supported by the Spanish *Ministerio de Economía y Competitividad* through the project grant MTM-2012-15167 and by NATO's Public Diplomacy Division in the framework of "Science for Peace", Project MD.SFPP 984520. This work was carried out while MR was with NEC Laboratories America, Princeton, NJ 08540, USA.

References

[1] Phillip Kaye and Christof Zalka. Optimized quantum implementation of elliptic curve arithmetic over binary fields. arXiv:quant-ph/0407095v1, July 2004. Available at `http://arxiv.org/abs/quant-ph/0407095v1`.

[2] Dmitri Maslov, Jimson Mathew, Donny Cheung, and Dhiraj K. Pradhan. An $O(m^2)$-depth quantum algorithm for the elliptic curve discrete logarithm problem over GF(2^m). *Quantum Information & Computation*, 9(7):610–621, 2009. For a preprint version see [15].

[3] National Institute of Standards and Technology, Gaithersburg, MD 20899-8900. *FIPS PUB 186-4. Federal Information Processing Standard Publication. Digital Signature Standard (DSS)*, July 2013. Available at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.

[4] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[5] Brittanney Amento, Martin Rötteler, and Rainer Steinwandt. Efficient quantum circuits for binary elliptic curve arithmetic: reducing T-gate complexity. *Quantum Information & Computation*, 13:631–644, July 2013.

[6] Brittanney Amento, Martin Rötteler, and Rainer Steinwandt. Quantum binary field inversion: improved circuit depth via choice of basis representation. *Quantum Information & Computation*, 13:116–134, January 2013.

[7] Henri Cohen and Gerhard Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete mathematics and its applications. Chapman & Hall/CRC, 2006.

[8] Jerome A. Solinas. An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.

[9] Daniel J. Bernstein, Tanja Lange, and Reza Rezaeian Farashahi. Binary Edwards Curves. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 244–265. International Association for Cryptologic Research, Springer, 2008.

[10] Aleksei Yu. Kitaev. Quantum computations: algorithms and error correction. *Russian Math. Surveys*, 52(6):1191–1249, 1997.

[11] Frederic Green, Steven Homer, Cristopher Moore, and Christopher Pollett. Counting, Fanout, and the Complexity of Quantum ACC. *Quantum Information & Computation*, 2(1):35–65, 2002.

[12] Edoardo D. Mastrovito. VLSI designs for multiplication over finite fields $GF(2^m)$. In Teo Mora, editor, *Proceedings of the Sixth Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 297–309. Springer, 1988.

[13] Edoardo D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linköping University, Linköping, Sweden, 1991.

[14] Arash Reyhani-Masoleh and M. Anwar Hasan. Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$. *IEEE Transactions on Computers*, 53(8):945–959, 2004.

[15] Dmitri Maslov, Jimson Mathew, Donny Cheung, and Dhiraj K. Pradhan. On the Design and Optimization of a Quantum Polynomial-Time Attack on Elliptic Curve Cryptography. arXiv:0710.1093v2, February 2009. Available at `http://arxiv.org/abs/0710.1093v2`.

[16] Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70:806–808, 1964.

[17] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1985.

[18] Toshiya Itoh and Shigeo Tsujii. Structure of parallel multipliers for a class of fields $GF(2^m)$. *Information and Computation*, 83:21–40, 1989.

[19] Richard Cleve and John Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 526–536, 2000.

[20] Paul Pham and Krysta Svore. A 2D nearest-neighbor quantum architecture for factoring in polylogarithmic depth. *Quantum Information & Computation*, 13(11&12):937–962, 2013.

[21] Samuel Kutin. Shor's algorithm on a nearest-neighbor machine. arXiv:quant-ph/0609001, September 2006. Available at `http://arxiv.org/abs/quant-ph/0609001`.

[22] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth Optimization of Clifford+T circuits via Matroid Partitioning. arXiv:quant-ph/1303.2042, March 2013. Available at `http://arxiv.org/abs/1303.2042`.