Improved quantum circuits for elliptic curve discrete logarithms

Thomas Häner[1], Samuel Jaques[2] *†, Michael Naehrig[3], Martin Roetteler[1], and Mathias Soeken[1]

[1] Microsoft Quantum, Redmond, WA, USA
{thhaner,martinro,a-masoek}@microsoft.com
[2] Department of Materials, University of Oxford, UK
samuel.jaques@materials.ox.ac.uk
[3] Microsoft Research, Redmond, WA, USA
mnaehrig@microsoft.com

Abstract. We present improved quantum circuits for elliptic curve scalar multiplication, the most costly component in Shor's algorithm to compute discrete logarithms in elliptic curve groups. We optimize low-level components such as reversible integer and modular arithmetic through windowing techniques and more adaptive placement of uncomputing steps, and improve over previous quantum circuits for modular inversion by reformulating the binary Euclidean algorithm. Overall, we obtain an affine Weierstrass point addition circuit that has lower depth and uses fewer T gates than previous circuits. While previous work mostly focuses on minimizing the total number of qubits, we present various trade-offs between different cost metrics including the number of qubits, circuit depth and T-gate count. Finally, we provide a full implementation of point addition in the Q# quantum programming language that allows unit tests and automatic quantum resource estimation for all components.

Keywords: Quantum cryptanalysis, elliptic curve cryptography, discrete logarithm problem, Shor's algorithm, resource estimates.

1 Introduction

Shor's algorithm [29,30] solves the discrete logarithm problem for finite abelian groups with only polynomial cost. When run on a large-scale, fault-tolerant quantum computer, its variant for elliptic-curve groups could efficiently break elliptic curve cryptography with parameters that are widely used and far out of reach of current classical adversaries.

Barring the efficient classical post-processing of the measured data, Shor's quantum algorithm consists of three steps: first, a superposition of exponents is created, then those exponents control the evaluation of a group exponentiation, and finally a quantum Fourier transform is applied to the exponent register, which is then measured. The group operations in the exponentiation must be computed in superposition and this is by far the most expensive step of the algorithm. Thus, the precise cost of Shor's algorithm depends on a detailed resource estimation for implementing the group operation on a quantum computer. For solving the elliptic curve discrete logarithm problem (ECDLP), the relevant operation is the repeated controlled addition of classical elliptic curve points to an accumulator point in a quantum register.

The first detailed discussion of the elliptic curve case was given by Proos and Zalka in [25]. Based on this work, Roetteler et al. [27] (hereinafter referred to as RNSL) presented explicit quantum circuits for point addition and all its components and automatically derive their resource estimates from a concrete implementation. Both papers focus on minimizing the number of qubits required to run the algorithm, since its polynomial runtime implies that it will run fast once an adversary has enough qubits to do so. They count the required number of *logical* qubits. For example, RNSL estimate that Shor's algorithm needs 2330 logical qubits to attack a 256-bit elliptic

* Partially supported by the University of Oxford Clarendon fund.
† Most of this work was done by Samuel Jaques, while he was an intern at Microsoft Research.

curve. Under plausible assumptions about physical error rates, this could translate into $6.77 \cdot 10^7$ physical qubits [11]. But the number of logical qubits is not the only important cost metric, and one might prioritize others such as circuit depth, the total number of gates, or the total number of likely expensive gates such as the Toffoli gate or the T gate.

Our goal in this work is not only to improve the circuits proposed by RNSL [27], but also to explore different trade-offs favoring different cost metrics. To this end, we provide resource estimates for point addition circuits optimized for depth, T gate count, and width, respectively. We also report on initial experiments with automatic optimization for T-depth and T gate count. By using the automatic compilation techniques presented in [19], we find low T-depth and low T-count circuits for a modular multiplication component and show significant improvements compared to their manually designed counterparts, however, at a very high cost to the number of qubits.

Beyond alternative choices for low-level arithmetic components, we also improve the higher-level structure of RNSL's circuit. While many components stay the same, the most dramatic improvements come from windowing techniques similar to those proposed by Gidney and Ekerå in [14] and a better memory management via *pebbling*. For example, instead of copying out the result in an out-of-place circuit that uses Bennett's method for embedding an irreversible function in a reversible computation, the result can be used for the next operation before it is uncomputed. This technique does not treat modular operations merely as black boxes, but can adaptively reduce the cost of the higher-level circuit they are used in. Along with a reformulation of the binary extended Euclidean algorithm, it significantly reduces costs for the modular inversion circuit.

One of our main contributions is a modular, testable library[4] of functions for elliptic curve arithmetic in the Q# programming language for quantum computing [31]. These incorporate different possible choices for subroutines like addition and modular multiplication. Besides enabling unit testing for all components, the Q# development environment allows automated resource estimation.

We strictly improve RNSL's estimates under all metrics. For example, for solving the ECDLP on a 256-bit elliptic curve, we reduce the number of qubits from 2338 to 2124, improve the T-count by a factor of 119 and the T-depth by a factor of 54. Asymptotically, we estimate that the number of T gates in our circuit for Shor's algorithm scales as $436n^3 + o(n^3)$.

Under a different trade-off optimizing for depth, our circuit costs 2^{33} T gates with T-depth of 2^{25} and 2871 qubits. Compared to RNSL, this is a factor 6000 reduction in T-depth with only a 22% increase in width. Asymptotically, our circuits can achieve $1115n^3/\lg n + o(n^3/\lg n)$ T gates, or $285n^2 + o(n^2)$ T-depth.

Extrapolating analogous values, breaking RSA-3072 would cost 2^{34} T gates and 9287 logical qubits [14]. This suggests that, at similar classical security levels, elliptic curve cryptography is less secure than RSA against a quantum attack.

2 Preliminaries

This section only gives a very brief discussion of the basic concepts used in this work. For a more detailed introduction to quantum computing, we refer to [24], for Shor's algorithm see [29] and [30] and for its ECDLP variant [25] and [27].

Quantum computing. A quantum computer acts on *quantum states* by applying quantum gates to its qubits. A quantum state is denoted by $|x\rangle$ for some label x. We work entirely with computational basis states, so x is always a bit string. As the fundamental gate set we use the Clifford+T gate set and assume that the T gate is by far the most expensive, including measurements. This is a plausible assumption because, in a quantum computer using a surface code for error correction, T gates consume special states which require many qubits and many surface code cycles to produce, and surface codes require frequent measurements for all gates [10]. A quantum algorithm is described by a sequence of gates in the form of a quantum circuit.

[4] Our code will be released under an open source license.

We use standard quantum circuit diagrams, with black triangles representing output registers. For circuit design and testing, we use circuits built from NOT, CNOT and Toffoli gates as those can be simulated efficiently at scale on classical inputs [27]. However, for cost estimation, we use decompositions over the Clifford+T gate set, such as the ones introduced in [1,12,28].

Shor's algorithm for the ECDLP. Let an instance of the ECDLP be given by two $\mathbb{F}_p$-rational points $P, Q \in E(\mathbb{F}_p)$ on an elliptic curve E over a finite field of large characteristic p such that $\mathrm{ord}(P) = r$, $Q \in \langle P \rangle$. The problem is to find the unique integer $m \in \{1, \ldots, r\}$ such that $Q = mP$. Shor's algorithm applies a Hadamard transform to two registers with $n+1$ qubits initialized to $|0\rangle$ to create the state $\frac{1}{2^{n+1}} \sum_{k,\ell=0}^{2^{n+1}-1} |k\rangle\,|\ell\rangle$. Next, the state $\frac{1}{2^{n+1}} \sum_{k,\ell=0}^{2^{n+1}-1} |k\rangle\,|\ell\rangle\,|kP + \ell Q\rangle$ is computed using the elliptic curve group law. A quantum Fourier transform $\mathrm{QFT}_{2^{n+1}}$ on $n+1$ qubits is applied to both $|k\rangle$ and $|\ell\rangle$ and the $2(n+1)$ qubits in $|k\rangle\,|\ell\rangle$ are measured. Classical post-processing then yields the discrete logarithm m. We assume that the algorithm is modified using the semiclassical Fourier transform method [15], which means that a small number of qubits can be re-used to act as the $2n+2$ qubits for $|k\rangle\,|\ell\rangle$. RNSL use only one qubit for this [27], but we use more than one to allow windowed arithmetic (see Section 5.1). The most cost-intensive part is the double-scalar multiplication to compute $|kP + \ell Q\rangle$.

Functions as quantum circuits. The elliptic curve group law is built from various classical functions that operate on bit strings of varying lengths n. For any function f, we use U_f to denote a quantum circuit that computes f.

We often want U_f to compute f *in-place*, meaning it has the action $U_f : |x\rangle \mapsto |f(x)\rangle$ on inputs $x \in \{0,1\}^n$. If U_f is built out of Clifford+T gates, each gate is easy to invert; thus, an in-place circuit U_f automatically yields an in-place circuit $U_f^\dagger$ that computes f^{-1}. As quantum circuits need to be reversible, an in-place circuit is not always possible, e.g. if f is not injective.

As an example consider modular multiplication. Let p be an integer modulus. The function $f : (x, y) \mapsto (x, xy \mod p)$ is injective for inputs $x, y \in \mathbb{Z}/p\mathbb{Z}$ that are co-prime to p. Its inverse is modular division. At present, the best circuits we know to compute modular division use some variant of the Euclidean algorithm. Thus, an in-place modular multiplication circuit would either be more expensive than the Euclidean algorithm or would provide a state-of-the-art circuit for modular division.

When an in-place circuit is not possible, U_f needs to implement f *out-of-place* as $|x\rangle\,|0\rangle^n \mapsto |x\rangle\,|f(x)\rangle$. Some circuits might require a number m of *auxiliary qubits*, such that

$$U_f : |x\rangle\,|0\rangle^n\,|0\rangle^m \xmapsto{U_f} |x\rangle\,|f(x)\rangle\,|g(x)\rangle\,, \tag{1}$$

where g is some function of the input. The auxiliary qubits are entangled with the other registers, and g must be *uncomputed* before the end of the computation to restore them to their original state. In our circuit diagrams, white triangles indicate such outputs.

If it is too costly to compute $g(x)$ from x and $f(x)$, one can use a method due to Bennett [4] to clean any auxiliary qubits. By adding another n-qubit register, the output $f(x)$ is "copied" to that register using CNOT gates. Then, the inverse $U_f^\dagger$ is applied. This trick roughly doubles the cost to reversibly compute f.

In a sequence of out-of-place circuits, uncomputing early steps prevents us from uncomputing later steps. To make the full algorithm work, we either need to keep intermediate steps at the cost of an increasing number of qubits or recompute them repeatedly at the cost of additional gates. This is an instance of a general problem known as a *pebbling game*.

Controlled circuits. As larger circuits are composed from smaller ones, the smaller circuit often needs to be controlled with a single qubit. We can "promote" each Clifford+T gate in the smaller circuit into a controlled variant, which is expensive, e.g., for CNOT and Toffoli gates. Thus, we want to optimize which gates we control. For example, a circuit using Bennett's trick can be controlled

by changing only the middle CNOT gates into Toffoli gates. For other circuits, we design controlled versions as needed.

3 Components

Design strategies. A full cost estimate of Shor's algorithm requires estimates at all layers of the architecture, including error correction, layout, and possibly architecture design. We focus only on the logical layer, and provide circuits that operate on abstracted logical qubits. From this level we cannot decide which design choices will be optimal. A shallower circuit with more logical qubits would have smaller error correction overhead and could have fewer physical qubits.

Thus, we provide different approaches and tradeoffs. We measure the T-depth, T-count, depth including all gates, and total number of qubits used ("width"). We focus on three strategies, favouring depth, T-count, or width. We could instead make different choices for each sub-circuit of Shor's algorithm, resulting in a large parameter space for potential optimization. Since we wrote Q# functions for all circuits, future work could combine different choices for each step.

Toffoli and AND gates. As explained by RNSL in the introduction of [27], circuits that are expressed as Toffoli gate networks can be implemented exactly over the Clifford+T gate set and can be classically simulated and tested. Therefore, our circuits are designed using the same approach. The only source of T-gates in such a circuit is from Toffoli gates. In many instances, we know that the output qubit is in the $|0\rangle$ state and we can use a dedicated AND circuit with a lower T-count instead. We use an AND gate design which combines Jones' and Selinger's AND gates [17,28]. It uses 4 T gates, while a Toffoli gate uses 7, and the inverse AND† uses no T gates while Toffoli† uses 7. AND gates use 1 auxiliary qubit for T-depth 1; Toffoli gates can use 4 auxiliary qubits for T-depth 1 or none for T-depth 3, see [1].

Integer addition. The adder of lowest known T-count is Gidney's modification of the Cuccaro et al. adder [8] (hereinafter called the CDKMG adder), which uses only $4n$ T gates to add two n-bit numbers, but uses n auxiliary qubits [12]. The adder of lowest known T-depth is the carry lookahead adder of Draper et al. [9] (hereinafter the DKRS adder), with logarithmic depth but $2n - O(n)$ auxiliary qubits. The adder of lowest width is due to Takahashi et al. [32] (hereinafter the TTK adder), which computes in-place using no auxiliary qubits. RNSL used the TTK adder. The DKRS adder uses $10n$ Toffoli gates, but of these, $4n$ can be replaced with an AND or AND† gate.

Controlled addition. We provide new methods for controlling the addition circuits. The DKRS adder uses a circuit to propagate carries, which is uncomputed. These gates do not need to be controlled. The remaining gates which must be promoted to controlled versions are all CNOT gates, leading to only a slight increase in the T-count. To control the CDKMG adder, we promote two CNOT gates per bit to Toffoli gates, as Figure 1 shows, and we change the final CNOT for the carry qubit to a Toffoli gate. Unfortunately, for both addition circuits, the new Toffoli gates cannot be replaced with AND gates.

Constant addition. When tackling an ECDLP instance, the prime modulus is a classically known integer, and we need a circuit to add it to integers encoded in quantum registers. The simplest method allocates a new quantum register, inputs the integer into the quantum register, runs any quantum addition circuit, then uncomputes the integer from the quantum register to free the qubits. This simple method is easy to control. Only copying the integer into the quantum register is controlled, and then an uncontrolled addition is used, as shown in Figure 2a. Copying an integer uses only X gates, so a controlled copy operation only uses CNOT gates, giving the same T-cost as uncontrolled quantum addition. We use this strategy with the CDKMG and TTK adders.

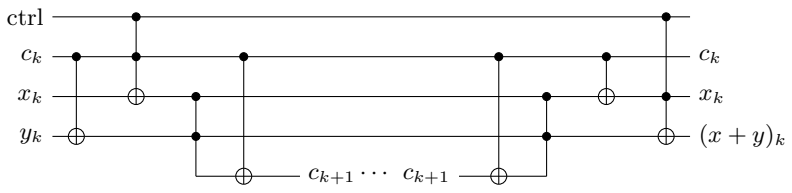
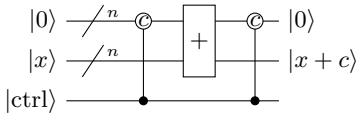
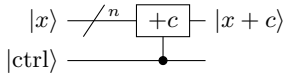


Fig. 1: Controlled addition block, adapted from [12].

An alternative is to "curry" the quantum addition circuit. In the DKRS adder, the qubits of one of the two inputs are only used as controls for CNOT and Toffoli gates. We can replace these with X and CNOT gates which are conditionally applied according to the bits of the classical integer. A controlled classical addition with this method needs to control the entire curried circuit (see Figure 2b), but we found that this is the most efficient approach with the DKRS adder.



(a) Copying the classical constant c is controlled, followed by uncontrolled addition.



(b) The addition circuit is curried for the constant c, then the full circuit is controlled.

Fig. 2: Two methods for controlling addition by a constant c.

Comparing integers. An addition circuit immediately gives a comparator by the one's complement trick. For integers x and y represented in binary, if we let x' denote the one's complement (all bits flipped), then $(x' + y)' = x - y$. If $y > x$, then $x - y < 0$ and the leading bit of $(x' + y)'$ will be 1. Thus, to construct a comparator, we simply flip all bits in x, compute the first half of an addition circuit, copy out the carry, then uncompute the addition circuit. With a control, we only need to control the final copy of the carry bit. We use this technique for comparators based on the previous adders. DKRS provide a comparator which we did not use for ease of implementation.

Fan-out and fan-in of control qubits. When a single qubit controls multiple parallel gates, they must be performed sequentially. To avoid this increase in depth, we opt to allocate extra qubits and fan-out the control [22] to these qubits using CNOTs. Then these qubits can control all the gates in parallel before we clear them again. For n simultaneous controlled gates, this requires n auxiliary qubits and at most $4n$ CNOT gates in depth $\lceil \lg n \rceil$, but no T gates. Our low-width optimization does not do this.

The fanned-out auxiliary control qubits could be retained to control many gates, depending on the application. Because the Q# language allocates qubits in a stack, this is often difficult. A function that allocates clean auxiliary qubits must restore them to the $|0\rangle$ state before returning. Because of this difficulty and the low gate cost of fan-out, we do not make such optimizations.

To control a single gate with the logical AND of n qubits requires a fan-in of control qubits. For low width, we use Barrenco et al.'s method [3], as RNSL did. This performs a multi-AND in-place but with $8n$ Toffoli gates and linear depth. If we instead allocate $n - 1$ auxiliary qubits and "compress" the controls with a tree of AND gates, it requires only $n - 1$ AND gates and AND-depth $\lceil \lg n \rceil$.

4 Modular arithmetic

Because modular reduction is irreversible, we cannot design an in-place circuit which maps x to $x \bmod N$ without creating some auxiliary qubits that represent the quotient $\lfloor x/N \rfloor$. Any algorithm for modular arithmetic that uses many modular reductions thus creates significant qubit overhead. Instead, we design bespoke circuits for each operation. Primarily, we follow RNSL [27]. We find that working in Montgomery representation [21] has the lowest costs. For an odd modulus p with $n = \lceil \lg p \rceil$, the Montgomery representation of an integer x is $x2^n \bmod p$.

We use the modular addition circuit from RNSL. Note that addition in Montgomery representation is the same as in standard representation. For controlled modular addition, we only need to control two operations, as Figure 3 shows. This automatically gives us modular addition by a constant, by replacing the quantum-quantum addition and comparison circuits by their quantum-classical counterparts.

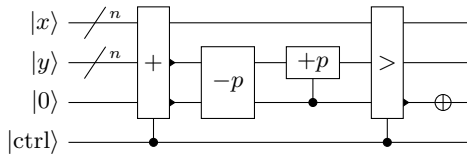


Fig. 3: Controlled modular addition.

4.1 Multiplication and squaring

RNSL provide two circuits each for multiplication and squaring. We use the one that operates on n-bit integers in Montgomery form with $2n$ additions and n halvings, and justify this choice in Appendix A.1. This circuit is a direct translation of classical Montgomery multiplication.

Windowed arithmetic. Windowed multiplication, such as [13], is not directly possible in our setting because we multiply two quantum integers, but we can adapt some of these techniques to our setting. When computing $x \cdot y$, the Montgomery multiplier uses the ith bit of $|x\rangle$ to control an addition of $|y\rangle$ to the output register. It then copies the lowest bit of the output to an auxiliary register $|m\rangle$ and uses this bit to control addition of the modulus p. This ensures that the sum is even, and so we rotate the register to divide by 2.

When using a window of size k, the integer x is split into k-bit words $x_{(1)}, \ldots, x_{(n/k)}$, analogous to classical interleaved radix-2^k Montgomery multiplication. The k-bit value $x_{(i)}$ is multiplied with y, adding the $(n+k)$-bit result to the output register. To add a suitable multiple of p to the output to set the k least significant bits to 0, these k bits are copied to an auxiliary register m_i. The multiple $t_{m_i}p$ such that $t_{m_i}p + m_i \equiv 0 \mod 2^k$ can be looked up from a classically pre-computed table T, where $T[m_i] = t_{m_i}p$ for $t_{m_i} \equiv p^{-1}m_i \mod 2^k$.

We use the bits in m_i as an address for a sequential quantum look-up [2][5], writing the resulting $(n+k)$-bit integer $t_{m_i}p$ into an auxiliary register. Then an uncontrolled addition of $t_{m_i}p$ into the output register clears the bottom k bits, so we can cyclically rotate by k bits. Figure 4 illustrates this process.

[5] This technique is referred to as *QROM* in [2]; some papers also call it *QRAM*; we use *quantum look-up* to distinguish it from quantum random oracles and to emphasize that it is constructed out of Clifford+T gates and does not require special QRAM technology.

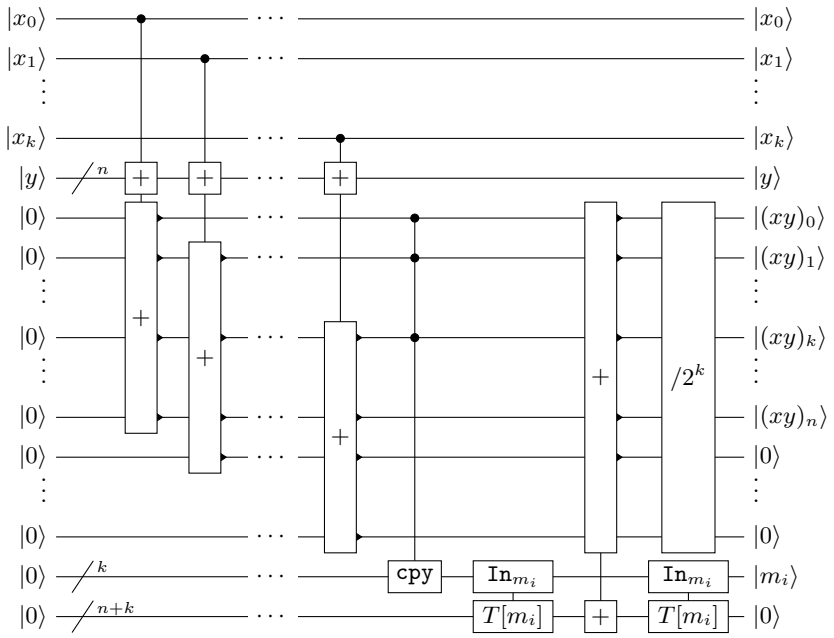
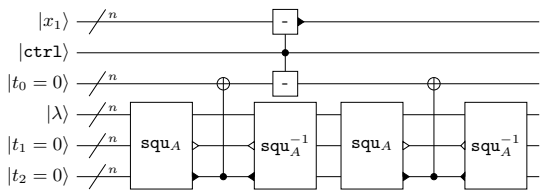


Fig. 4: Circuit for a single window of windowed add-and-halve multiplication. The `cpy` gate copies k bits with k CNOT gates. The gate $\mathtt{In}_{m_i}$ performs a quantum table look-up of $T[m_i]$, where T is the table described in the text. The circuit in this figure is repeated $\lceil n/k \rceil$ times, with a final modular correction step, to perform a single modular multiplication.

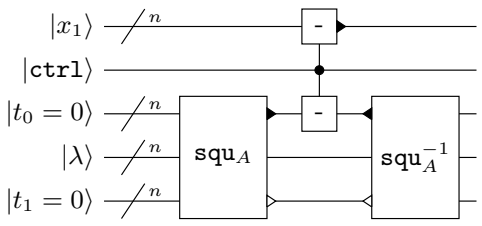
Pebbling. Given integers x, y, and z, one method to compute $xy + z$ in place is to first compute xy in an auxiliary register, then to add it to the register containing z, and finally to uncompute xy. This works for any generic multiplication circuit, but at the cost of two multiplications.

The circuit we use requires the Bennett method, and we replace its CNOT step with an addition. This is just a pebbling technique [5]; we are keeping the auxiliary qubits until we have finished the next computation (an addition) before uncomputing them. The cost of this multiply-then-add is just the sum of the costs of a multiplication and an addition, rather than twice the cost of a multiplication and an addition.

In particular, RNSL treat their squaring circuit as a black box in the full elliptic curve addition circuit. The computed square is only needed for one subtraction before it must be uncomputed; therefore we can use this multiply-then-add trick. Figure 5 shows how this saves almost half the gate cost and depth of the squaring, as well as saving an auxiliary register.



(a) Original circuit [27] with `squ` circuit expanded, $\mathtt{squ}_A$ produces dirty auxiliary qubits in $|t_1\rangle$.



(b) More efficient version.

Fig. 5: Improvement to the squaring circuit from [27] in the context of elliptic curve point addition.

Automatic optimization. The multiply-then-add operation is a very costly component in the point addition circuit. We explore what is possible when reducing its T-depth and T-gate count. Appendix E shows the results for a compilation using the method from [19] after automatic optimization [33] of a logic network generated from the operation. We achieve extremely low T-depth and T-count, but the number of logical qubits increases significantly. We leave further exploration of such techniques for future work.

4.2 Modular inversion

Modular inversion uses variants of the extended Euclidean algorithm (EEA). The EEA repeatedly divides integers. For two n-bit integers, we might expect each division requires $O(n^2)$ operations and, in the worst case, we need $O(n)$ divisions. In practice the complexity is smaller, because the complexity of the divisions becomes smaller in the course of the algorithm. Unfortunately, to exploit this fact, the circuit must be *dynamic*: The number of divisions and the circuit for each division depend on the inputs. We cannot build such a quantum circuit, since we cannot observe the input (it may be in superposition).

We considered several approaches, but found that RNSL's circuit is the best. Appendix A.2 details our reasoning. However, we found several improvements to it. Their circuit models an algorithm of Kaliski [18], which applies a round operation (Algorithm 7a) for up to $2n$ iterations. Conditional logic selects one of four different cases that can occur in each round. As a quantum circuit, this requires applying all four possible rounds, each with a different control. Algorithm 7b shows our alternative formulation based on swaps.

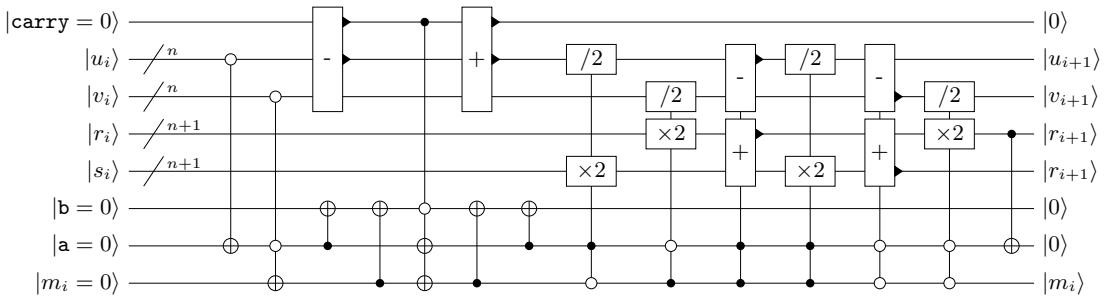
Figure 6a shows the round operation of RNSL's quantum circuit implementation of Kaliski's algorithm. This circuit repeats the controlled additions, doublings, and halvings, with different registers playing the role of input or output. Our method performs each of these operations once, using controlled swaps to arrange inputs and outputs for the respective case. The lower auxiliary bit and $|m_i\rangle$ uniquely specify the round and we use these bits to control a swap. This leads to Figure 6b. A controlled n-bit swap is approximately the same cost as a controlled n-bit cyclic shift.

Correcting pseudo-inverses in parallel. Classically, the Kaliski algorithm uses only k rounds, with $n \leq k \leq 2n$ for n-bit integers. RNSL's quantum circuit executes $2n$ rounds, controlled in such a way that only k actually modify the input. This produces an auxiliary qubit counter with a value of $2n - k$ and a pseudo-inverse output of $x^{-1}2^{-n+k} \bmod p$ for an input x. We want to output the Montgomery inverse $x^{-1}2^n \bmod p$, which requires correcting the pseudo-inverse.

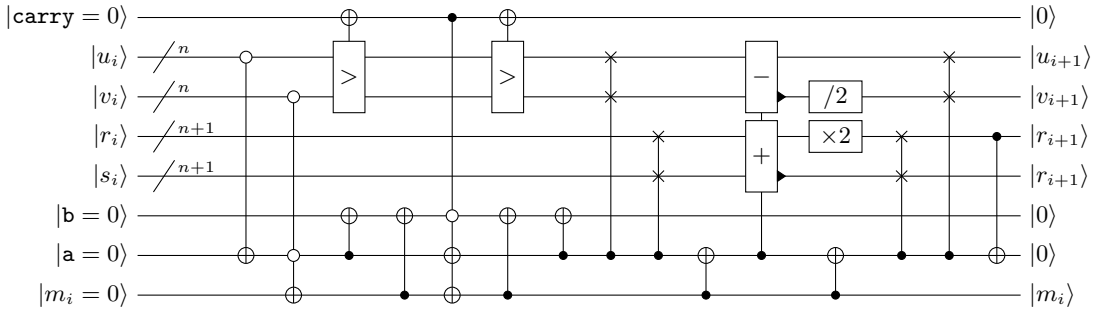
Instead of a separate doubling circuit like the one RNSL use, we add a doubling operation to each division round. After copying out the pseudo-inverse, during the subsequent uncomputation we use the same control that the round operation uses: in each of the $2n$ rounds, either we double the output register or perform a division round. These can be done in parallel, which improves the depth without increasing the total gate count. The result is that we compute $2n - k$ doublings, exactly what is needed to correct the pseudo-inverse.

Modular division. In elliptic curve point addition, the inversion is only necessary to compute a division. To divide an integer x by y modulo a prime p, RNSL first invert y in an auxiliary register, perform the multiplication by the result, then invert y again to uncompute the auxiliary qubits as shown in Figure 8a.

Because the inversion uses Bennett's method, we pebble in the same way as the multiply-then-add circuit. To save qubits, we notice that after the inversion, three registers contain known values of 0, 1, and the modulus p. We can clear these auxiliary qubits with at most $3n$ parallel X gates, then re-use them for modular multiplication. Denoting the inverse and multiplication operations with auxiliary qubits by $\texttt{inv}_A$ and $\texttt{mul}_A$, respectively, Figure 8b depicts a full modular division. The pseudo-inverse correction is compatible with modular division.



(a) Round operation from [27].



(b) Improved round operation.

Fig. 6: Improvement to the round operation in the binary extended Euclidean algorithm from [27] addressing the different cases by controlled swaps.

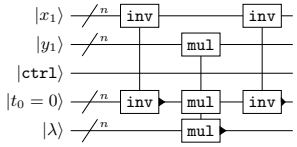
(a) Kaliski's algorithm

1: **if** u odd and v even **then**
2: $v \leftarrow v/2$
3: $r \leftarrow 2r$
4: **else if** u even and v odd **then**
5: $u \leftarrow u/2$
6: $s \leftarrow 2s$
7: **else if** u odd and v odd and $u > v$ **then**
8: $u \leftarrow (u - v)/2$
9: $r \leftarrow r + s$
10: $s \leftarrow 2s$
11: **else if** u odd and v odd and $v \geq u$ **then**
12: $v \leftarrow (v - u)/2$
13: $s \leftarrow r + s$
14: $r \leftarrow 2r$
15: **end if**

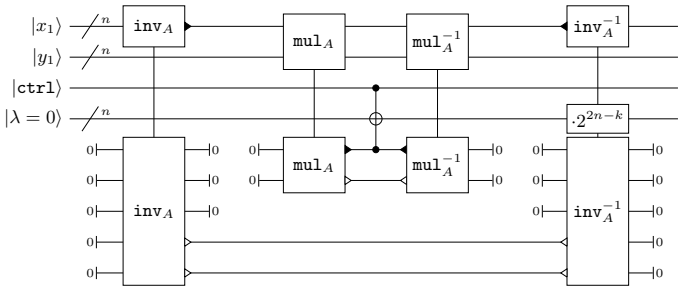
(b) Equivalent formulation

1: $b_{swap} \leftarrow$ `false`
2: **if** u even and v odd, or u and v both odd and $u > v$ **then**
3: **swap** u and v
4: **swap** r and s
5: $b_{swap} \leftarrow$ `true`
6: **end if**
7: **if** u odd and v odd **then**
8: $v \leftarrow v - u$
9: $s \leftarrow r + s$
10: **end if**
11: $v \leftarrow v/2$
12: $r \leftarrow 2r$
13: **if** b_{swap} **then**
14: **swap** u and v
15: **swap** r and s
16: **end if**

Fig. 7: Kaliski's algorithm, and an equivalent formulation based on swaps.



(a) Inversion steps in elliptic curve point addition from [27]. Auxiliary qubits are not shown. Each `inv` block represents $\texttt{inv}_A$, copying out, then $\texttt{inv}_A^{-1}$ (with the same pattern for `mul`). Thus, this circuit contains 4 $\texttt{inv}_A$ steps and 2 $\texttt{mul}_A$ steps.



(b) More efficient modular division. 0⊢ depicts initializing an auxiliary qubit to zero, and ⊣0 depicts removing an auxiliary qubit known to be zero. The doubling of the output λ in the last step corrects the pseudo-inverse output.

Fig. 8: Improvement to the modular division circuit from [27] by clearing and re-using auxiliary qubits before the full uncomputation.

5 Elliptic curves

Elliptic curve arithmetic has been heavily optimized for *classical* computers, but because Shor's algorithm requires unique representations of points and in-place point addition, few of these optimizations apply. We find affine Weierstrass coordinates to be the most efficient method; Appendix A.4 explains this choice in more detail.

Affine Weierstrass coordinates are one of the conceptually simplest methods, and RNSL use them for their circuit. We assume the elliptic curve equation has the form $E : y^2 = x^3 + ax + b$ with $a, b \in \mathbb{F}_p$ and we represent points by pairs (x, y) that satisfy this equation. Combining RNSL's circuit with the optimizations from this paper, we obtain the circuit in Figure 9. The total cost is 2 divisions, 2 multiplications, 1 squaring, and 9 additions.

The simple formulas with affine coordinates are naturally in-place. The general concept is the same as in [27]. The original x and y coordinates are replaced by multiplying and adding to them, and once we have, we can use the new coordinates to uncompute the slope. This also means that the circuit produces incorrect outputs if P or Q is the point at infinity or if $P = \pm Q$, because in these cases the slope does not exist. Proos and Zalka [25] and RNSL [27] both argue that these exceptional cases only slightly distort the desired quantum state in Shor's algorithm and their influence is negligible.

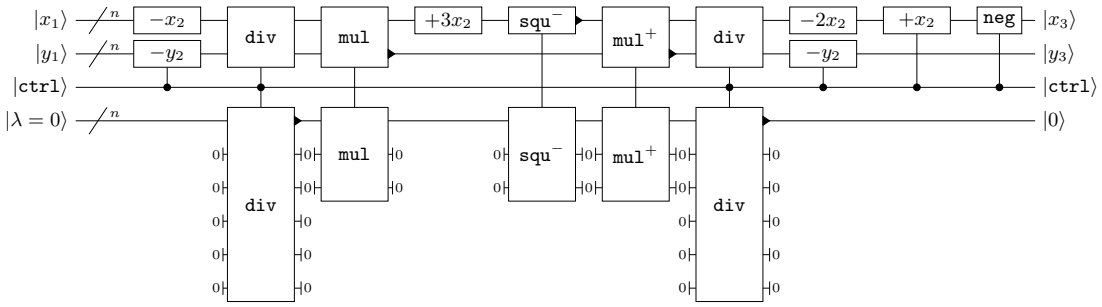


Fig. 9: A single elliptic curve addition of a classical point $P = (x_2, y_2)$ with a quantum point $Q = (x_1, y_1)$ in affine Weierstrass coordinates, with $P + Q = (x_3, y_3)$. `div` indicates the circuit from Figure 8b, $\texttt{squ}^-$ indicates squaring-and-subtracting (as in Figure 5b) and $\texttt{mul}^+$ indicates multiplying-and-adding.

5.1 Windowed arithmetic

Shor's algorithm uses $2n$ qubits as control qubits, half of which control circuits that add P, $2P$, $4P$, etc.; the other half control addition of Q, $2Q$, etc. The points are added to a single quantum register which we call the *accumulator*. This process requires $2n$ point additions. We instead use a windowed approach, analogous to ubiquitous classical pre-computation techniques and similar to the techniques used for RSA in [14]. Other classical pre-computation strategies are less effective, as Appendix A.5 discusses.

For windowed scalar multiplication, we use the index as an address for a sequential quantum look-up, which loads a superposition of points $\mathcal{O}, P, 2P, 3P, \ldots, (2^\ell - 1)P$ into an auxiliary register which we call the *cache*. For elliptic curves, this requires us to switch from a circuit adding a classical point to a circuit adding two quantum points, but this has little effect on the cost. The depth and T-count of the look-up are exponential in the window size ℓ, but we save ℓ point additions.

Signed addition. We can save a factor of 2 in windowing by using one qubit to control the sign of $P = (x, y)$ and only using $\ell - 1$ for the look-up. Since $-P = (x, -y)$, the extra qubit only needs to control a cheap modular negation. This changes the indexing slightly. Our original windowed circuit took an address register $|b\rangle$ and an input register $|R\rangle$, and produced $|b\rangle\,|[b]P + R\rangle$. Using the top bit $b_{\ell-1}$ of $b = b_{\ell-1}2^{\ell-1} + b'$ as a sign and looking up $b'P$ if $b_{\ell-1} = 1$ and $(2^{\ell-1} - b')P$ and negating it if $b_{\ell-1} = 0$, we can implement the operation

$$\sum_{b\in\{0,1\}^\ell} |b\rangle\,|R\rangle \mapsto \sum_{b\in\{0,1\}^\ell} |b\rangle\,\left|[b - 2^{\ell-1}]P + R\right\rangle. \tag{2}$$

Thus, we get an offset in each round, but since the offset is constant, it has no effect on the final phase estimation.

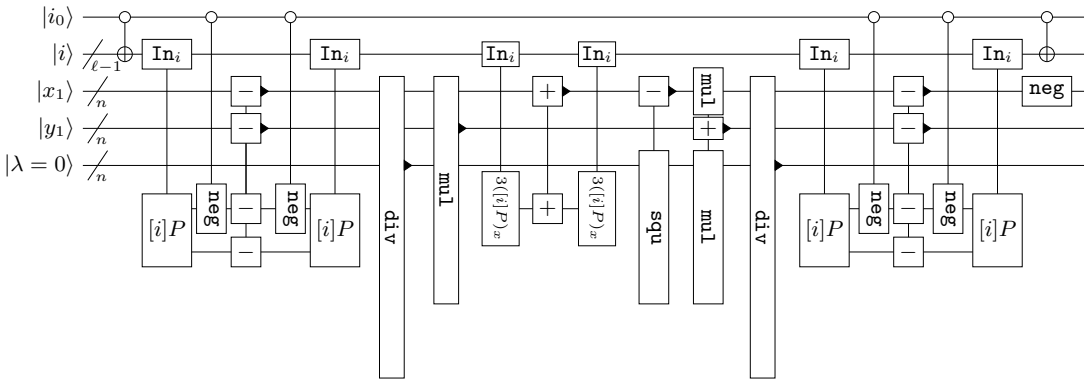


Fig. 10: Signed windowed elliptic curve addition, with ℓ address qubits. In_i is the indexing half of the look-up gate, which writes the classical data shown into the output registers. The width of the circuits is proportional to the number of auxiliary qubits used.

6 Results

We present quantum resource estimates for Shor's algorithm based on our cost estimates for windowed elliptic curve point addition. Results are obtained for optimizing three different cost metrics: either minimizing circuit width (i.e. the total number of logical qubits), the total number of T gates in the circuit, or total circuit depth. Window sizes above 18 were too large to simulate,

so we extrapolated from the costs for smaller lookup tables. With Q#, we calculated the cost of a point addition on the three NIST curves [34] P256, P384 and P521, using 8-bit look-ups, then subtracted the cost of six 8-bit look-ups and added the cost of six ℓ-bit look-ups to get the cost of point addition with an ℓ-bit window. We multiplied this cost by the number of windows dividing $2n$ to get the full cost of Shor's algorithm for ECDLP. From this we manually selected optimal window sizes. We can use n instead of $n+1$ because the order of each NIST curve is less than its modulus [34].

Table 1 shows our results together with those of RNSL [27] for comparison. Their circuits use fewer than 2 Toffoli gates per time step on average, so we assume that with 8 extra qubits (c.f. Section 3) they can use Toffoli gates of T-depth 1. Our optimization for the number of qubits is shown in the row labeled *Low W*. Since RNSL optimize for the same metric, those results allow a direct comparison. We were able to improve on RNSL's circuit in all metrics. For P256, we reduce the number of logical qubits from 2338 to 2124, while reducing the T-depth and T-count by factors of 54 and 119, respectively.

Additionally, we report more significant improvements over RNSL's work in depth and T-count when optimizing for those. For 521-bit moduli, the improvement is a factor of 13,792 in depth for an increase of 22% in qubits, or a 463 factor reduction in T-gates for a 12% increase in qubits.

Circuit	Window Size	Gates				Depth		Width
		Cliffords	Measure	T	Total	T	All gates	Qubits
256-bit modulus								
RNSL	–	–	–	$1.60 \cdot 2^{39}$	–	$1.69 \cdot 2^{36}$	–	2338
Low W	19	$1.32 \cdot 2^{34}$	$1.76 \cdot 2^{26}$	$1.72 \cdot 2^{32}$	$1.45 \cdot 2^{35}$	$1.98 \cdot 2^{30}$	$1.89 \cdot 2^{32}$	2124
Low T	19	$1.75 \cdot 2^{33}$	$1.95 \cdot 2^{27}$	$1.08 \cdot 2^{31}$	$1.80 \cdot 2^{34}$	$1.44 \cdot 2^{29}$	$1.85 \cdot 2^{31}$	2619
Low D	15	$1.04 \cdot 2^{34}$	$1.61 \cdot 2^{28}$	$1.34 \cdot 2^{32}$	$1.40 \cdot 2^{34}$	$1.12 \cdot 2^{24}$	$1.40 \cdot 2^{27}$	2871
384-bit modulus								
RNSL	–	–	–	$1.44 \cdot 2^{41}$	–	$1.51 \cdot 2^{38}$	–	3492
Low W	21	$1.46 \cdot 2^{36}$	$1.23 \cdot 2^{29}$	$1.51 \cdot 2^{34}$	$1.57 \cdot 2^{37}$	$1.68 \cdot 2^{32}$	$1.77 \cdot 2^{34}$	3151
Low T	19	$1.05 \cdot 2^{35}$	$1.28 \cdot 2^{29}$	$1.74 \cdot 2^{32}$	$1.10 \cdot 2^{36}$	$1.21 \cdot 2^{31}$	$1.31 \cdot 2^{33}$	3901
Low D	15	$1.73 \cdot 2^{35}$	$1.34 \cdot 2^{30}$	$1.13 \cdot 2^{34}$	$1.17 \cdot 2^{36}$	$1.23 \cdot 2^{25}$	$1.48 \cdot 2^{28}$	4278
521-bit modulus								
RNSL	–	–	–	$1.81 \cdot 2^{42}$	–	$1.91 \cdot 2^{39}$	–	4727
Low W	22	$1.85 \cdot 2^{37}$	$1.59 \cdot 2^{30}$	$1.82 \cdot 2^{35}$	$1.98 \cdot 2^{38}$	$1.99 \cdot 2^{33}$	$1.09 \cdot 2^{36}$	4258
Low T	20	$1.45 \cdot 2^{35}$	$1.49 \cdot 2^{30}$	$1.00 \cdot 2^{34}$	$1.57 \cdot 2^{36}$	$1.40 \cdot 2^{32}$	$1.54 \cdot 2^{34}$	5273
Low D	15	$1.10 \cdot 2^{37}$	$1.70 \cdot 2^{31}$	$1.43 \cdot 2^{35}$	$1.48 \cdot 2^{37}$	$1.13 \cdot 2^{26}$	$1.27 \cdot 2^{29}$	5789

Table 1: Resource estimates for Shor's full algorithm to compute the ECDLP. RNSL results are taken from [27]. Rows labeled Low W/Low T/ Low D show estimates for circuits minimizing width, T gate count and total depth, respectively.

Asymptotic formulas. We ran resource estimates for the various components of the elliptic curve point addition circuits on a range of input bit sizes. We then determined models for fitting the data with linear regression and deduced asymptotic formulas for the costs of these operations parameterized by the bit size n. Tables 2, 3 and 4 in Appendix D show the results of our experiments.

Table 4 also shows such formulas for a full signed windowed elliptic curve point addition and the full circuit of Shor's algorithm using such point addition. The results show that our circuit optimized for low width can solve the ECDLP on an n-bit elliptic curve with about $8n + 10.2 \lfloor \lg n \rfloor - 1$ logical qubits using roughly $436n^3 - 1.05 \cdot 2^{26}$ T-gates at a T-depth of $120n^3 - 1.67 \cdot 2^{22}$. The total number of gates is $2900n^3 - 1.08 \cdot 2^{31}$ with depth $509n^3 - 1.84 \cdot 2^{27}$.

Our circuits optimized for a low number of T gates require about $10n + 7.4 \lfloor \lg n \rfloor + 1.3$ qubits and need $1115n^3 / \lg n - 1.08 \cdot 2^{24}$ T gates with T depth $389n^3 / \lg n - 1.70 \cdot 2^{22}$. Optimizing for low

depth brings down the overall depth to $2523n^2 + 1.10 \cdot 2^{20}$ and the T-depth to $285n^2 - 1.54 \cdot 2^{17}$, but requires $11n + 3.9 \lfloor \lg n \rfloor + 16.5$ qubits.

Acknowledgements. We thank Dan Bernstein, Martin Ekerå, Iggy van Hoof, and Tanja Lange for helpful suggestions about elliptic curve arithmetic. We thank Martin Albrecht for lending computing power to run resource estimates.

References

1. Matthew Amy, Dmitri Maslov, Michele Mosca, 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.
2. Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity. *Physical Review X*, 8(4):041015, October 2018. arXiv: quant-ph/1805.03662.
3. Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995. arXiv: quant-ph/9503016.
4. Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, Nov 1973.
5. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, August 1989.
6. Daniel J. Bernstein and Tanja Lange, 2007. https://www.hyperelliptic.org/EFD.
7. Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, May 2019.
8. Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. 2004. arXiv:quant-ph/0410184.
9. Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. June 2004. arXiv: quant-ph/0406142.
10. Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.
11. Vlad Gheorghiu and Michele Mosca. Benchmarking the quantum cryptanalysis of symmetric, public-key and hash-based cryptographic schemes, 2019.
12. Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, June 2018.
13. Craig Gidney. Windowed quantum arithmetic. 2019. arXiv: quant-ph/1905.07682.
14. Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. May 2019. arXiv: quant-ph/1905.09749.
15. Robert Griffiths and Chi-Sheng Niu. Semiclassical Fourier transform for quantum computation. *Phys. Rev. Letters*, 76(17):3228–3231, 1996.
16. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. SIKE, 2017. Submission to [23]. http://sike.org.
17. Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, 2013.
18. Burton S. Kaliski. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, Aug 1995.
19. Giulia Meuli, Mathias Soeken, Earl Campbell, Martin Roetteler, and Giovanni De Micheli. The role of multiplicative complexity in compiling low T-count oracle circuits, 2019. arXiv: quant-ph/1908.01609.
20. Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjørner, and Giovanni De Micheli. Reversible pebbling game for quantum memory management. In *Design, Automation & Test in Europe Conference*, pages 288–291, 2019.
21. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
22. Cristopher Moore. Quantum circuits: Fanout, parity, and counting, 1999. arXiv: quant-ph/9903046.
23. National Institute of Standards and Technology. Post-quantum cryptography standardization, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

24. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK, 2000.
25. John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. January 2003. arXiv: quant-ph/0301141.
26. Rich Rines and Isaac Chuang. High Performance Quantum Modular Multipliers. January 2018. arXiv: quant-ph/1801.01081.
27. Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *ASIACRYPT 2017*, volume 10625 of *Lecture Notes in Computer Science*, pages 241–270. Springer, 2017.
28. Peter Selinger. Quantum circuits of T-depth one. *Physical Review A*, 87(4):042302, April 2013. arXiv: 1210.0974.
29. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS 1994*, pages 124–134, 1994.
30. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.
31. Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher E. Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *RWDSL@CGO 2018*, 2018.
32. Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *Quantum Information and Computation*, 10, 10 2009.
33. Eleonora Testa, Mathias Soeken, Luca G. Amarù, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Design Automation Conference*, page 74, 2019.
34. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-4, 2013. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

A Alternative approaches

A.1 Modular multiplication.

RNSL provide two circuits for modular multiplication. The first is the one proposed by Proos and Zalka [25], which uses a double-and-add approach, where doubling and addition are both modular operations modulo p. The other is reversible Montgomery multiplication, which uses an add-and-halve approach and works in Montgomery form. The primary motivation for considering Montgomery multiplication instead of the straightforward double-and-add method is that modular reduction is achieved by suitable additions to clear lower order bits and divisions by 2 (i.e. bit rotations) as part of the whole circuit, not delegated to the addition and halving circuits. This results in simpler operations per bit.

However, Montgomery multiplication has the downside that it entangles with a register of auxiliary qubits which must be cleared. In our case, at every point in an elliptic curve point addition, we have enough spare auxiliary qubits for this. Overall, it is cheaper, even with the Bennett method, and especially with the multiply-then-add technique of Section 4.1.

Rines and Chuang recently provided high-performance modular multipliers [26]. A direct comparison is difficult, since they give costs in Toffoli gates. Our low-T multiplier scales as $38n^2 + o(n^2)$ T gates, while their best exact multiplier uses $4n^2$ Toffoli gates, suggesting perhaps $28n^2$ T gates. However, some of these Toffoli gates might be replaceable with AND gates. Their adder seems to use $4.5n$ additions, slightly more than our $2n + O(n/\log n)$ additions. We leave a true comparison and Q# implementation for future work.

A.2 Modular Inversion.

Proos and Zalka [25] (PZ) gave an approach to modular inversion based on precise control of a bit-shift division operation, with asymptotic complexity of $O(n^2)$. There are $O(n)$ iterations of a *round*. Each round implements conditional logic by computing state qubits, then using those state qubits to control some operations on the integer registers.

RNSL use a similar round-based construction, which implements a reversible binary extended Euclidean algorithm. As with multiplication, the primary difference between the PZ division and the RNSL division is that PZ's is based on doubling and integer long division, while RNSL's is based on halving and binary operations. The PZ inversion leaves only $O(\lg n)$ auxiliary qubits, while RNSL creates $2n + O(\lg n)$ auxiliary qubits, but PZ has a higher depth and gate cost.

Naively, the PZ approach uses $5n$ qubits, though they show that, with fidelity loss on the order of $O(n^{-3})$ per round, they require only $2n + 8\sqrt{n} + O(\log n)$ qubits. The RNSL approach uses $6n$ qubits. We choose to use the RNSL algorithm. It is exactly correct, so it can be used for higher depth algorithms, and the total T-cost and depth are less than half of the PZ approach.

A.3 Recursive GCD Algorithms.

There are several sub-quadratic GCD algorithms (such as [7]). These work by defining a series of 2×2 matrices T_n such that $T_n T_{n-1} \cdots T_1 (u, v)^T$ will map integers u and v to the nth step of the Euclidean algorithm. These can be computed and multiplied together recursively.

Adapted to quantum circuits, these approaches require quantum matrix multiplication. We could find no efficient method to do this in-place, meaning that each recursive call would require a new set of auxiliary qubits to store the matrix output. This would quickly overwhelm our qubit budget. The base case of [7] is nearly identical to our approach for a single round.

One of the primary advantages of [7] is that the recursive process allows much of the arithmetic to be done with small integers which fit into the registers of classical CPUs. All the qubits in our model of a quantum computer are identical, so it has no caching or register issues. If quantum technologies arise with different kinds of qubits (perhaps a "memory" with higher coherence times but lower gate fidelity), then recursive GCD algorithms should be revisited. It is also possible that the specific structure of the matrices in this approach permit an easy, in-place multiplication circuit. We leave this to future work.

A.4 Alternate curve representations

Projective coordinates. Projective coordinates use equivalence classes $(X : Y : Z)$ of triples (X, Y, Z) to represent an elliptic curve point, where $(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2)$ iff there is some non-zero constant c such that $X_1 = cX_2$, $Y_1 = cY_2$, and $Z_2 = cZ_2$. These can be used with many different families of curves. Projective coordinates lend themselves to efficient, inversion-free arithmetic, which is appealing for classical computers.

Projective coordinates do not give a *unique* representation of each point, which Shor's algorithm requires to ensure history independence and thus proper interference of states in superposition. Dividing by the Z coordinate produces a unique representation but requires an expensive division. It is an open problem to provide a unique projective representation with division-free arithmetic.

Another issue is that the classical elliptic curve formulas, naively adapted to quantum circuits, operate out-of-place. An out-of-place addition circuit is easy to adapt into an in-place addition circuit. If we can construct a circuit U_{+Q} to add a point Q, we can construct a circuit U_{-Q}, and we can construct an in-place point addition as shown in Figure 11 up to a final swap of the two qubit registers. This doubles the cost of point addition.

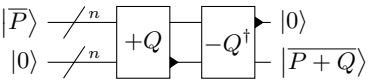


Fig. 11: An in-place point addition from out-of-place.

This technique requires a unique representation. If $(P + Q) - Q$ does not have the same representation as P, we cannot cancel them out. Thus, for any current algorithm to compute

addition with projective coordinates with cost C, we can transform it to a quantum-suitable in-place version with cost $2C + 2D$, where D is the cost of division. The division creates a unique representation.

Some projective coordinate systems such as projective x-only Montgomery coordinates can only handle differential addition. In this case, we start with P, Q, and $Q - P$, and use $Q - P$ as a helper to compute $P + Q$. We did not find a method to clear the register with P from the outputs Q, $Q - P$, and $P + Q$, so we are further restricted to curves that have direct addition formulas.

According to the Explicit Formulas Database [6], the lowest-cost addition, possibly assuming $Z = 1$ for the initial point, uses 6 squares and/or multiplications. With the required reductions, the total cost is 12 squares/multiplications and 2 divisions, much higher than affine Weierstrass coordinates. Thus, we choose not to use projective coordinates in this work.

Elliptic curve point addition in affine Montgomery coordinates. An alternative affine coordinate system is provided by elliptic curves in Montgomery form

$$by^2 = x^3 + ax^2 + x. \tag{3}$$

Again, we represent points as pairs (x, y) that satisfy the curve equation.

Following the reference implementation for affine coordinate addition from [16], we construct the circuit in Figure 12. This uses one more register than affine Weierstrass coordinates, one more multiplication, and one more squaring.

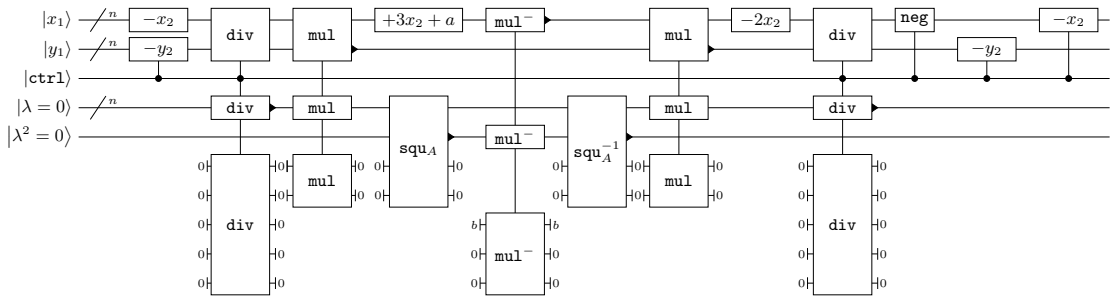


Fig. 12: Constant point addition in affine Montgomery coordinates.

However, we note that the main source of extra costs is that, for a slope λ, there is a $b\lambda^3$ term. This requires 4 sequential multiplications/squarings. If we are lucky enough to have a curve where $b = 1$, then we save space and operations by directly adding the λ^2 to the x_1 register. In fact, in this case, we can simply use the Weierstrass addition circuit, so long as we replace the $+3x_2$ operator with $+3x_2 + a$. Thus, we have no reason to prefer affine Montgomery coordinates, and we will continue to use short Weierstrass form.

A.5 Precomputation.

Precomputed tables of certain powers of the base element can speed up exponentiations. The "comb" method is a standard technique used for elliptic curve scalar multiplication. To multiply a point P by a scalar k, we divide k into $k_1 + 2k_2 + \cdots + 2^\ell k_\ell$ for some ℓ, with the property that k_j contains bits of k in positions congruent to j modulo ℓ (each k_j looks like a comb of bits). We then precompute a table of all multiples of P by scalars of the form $b_0 + b_1 2^\ell + b_2 2^{2\ell} \ldots$, with $b_i \in \{0, 1\}$. By the definition of k_j, each $k_j P$ is a precomputed point in this table for all j. Thus, we can compute kP by using k_j to look up elements of the table, adding them to a running total, and doubling the running total.

The advantage of the comb technique is that it saves precomputation. We only precompute one table and use it for the entire computation. Unfortunately for the quantum case, precomputation is essentially free because it is entirely classical, but look-ups are expensive. The comb technique does not reduce the number of table look-ups, since we must do a separate look-up for each index k_j.

Further, efficient in-place point doubling is unlikely, since it implies efficient in-place point halving. Thus, doubling points in the comb would require some pebbling technique which would likely add significant depth or width costs.

B Modular division and addition

For elliptic curve addition, we only need to divide integers and copy the result to a blank output, but other applications may wish to construct a circuit that, given registers containing x, y, and z, will compute $yx^{-1} + z$.

We might simply invert, multiply, and then add the output of the multiplication instead of copying. However, doubling the output to correct the pseudo-inverse while uncomputing will also multiply z by a factor of 2^{2n-k}. To correct for this, we can repeatedly halve z during the *forward* computation of the modular inverse. This means that while we compute the modular inverse, we control a modular halving of the register containing z by the counter, which will halve z exactly $2n - k$ times. Then we multiply the pseudo-inverse by y and add the result to the register with z, producing the state

$$\left|x^{-1}2^{2n-k} \bmod p\right\rangle \left|y2^n \bmod p\right\rangle \left|z2^{2n-k} + x^{-1}y2^{2n-k} \bmod p\right\rangle. \tag{4}$$

From here, if we perform controlled modular doublings of the register containing z as we uncompute the inversion circuit, this will correct both z and the pseudo-inverse of x, producing the desired output.

C Analysis of windowed arithmetic

A quantum look-up to N elements requires $4N$ T-gates [2]. To optimize window costs, we balance this cost against the operations we save.

Multiplication. Section 4.1 describes a single windowed multiplication round. For n-bit integers with window size k, repeating this round $\lceil n/k \rceil$ times performs the full multiplication. Since the quantum look-up will cost $4 \cdot 2^k$ T gates [2] and uncontrolled $n + k$-bit addition costs $O(n + k)$ T gates, we expect the optimal window size to be approximately $k = O(\lg n)$. The total multiplication cost is still $O(n^2)$ because we only window addition by p, not addition of the quantum register y. Compared to un-windowed add-and-halve multiplication, windowing should save a factor of roughly $\frac{1}{2} + O(\frac{1}{\lg n})$. Similar reasoning suggests savings of $\frac{1}{2} + O(\frac{1}{\lg \lg n})$ in depth.

Numerical estimates show a window size of $k = c_1 \lg n + c_2$ optimizes T-count, where $0.68 \leq c_1 \leq 0.75$ and $0.12 \leq c_2 \leq 0.78$, and $1.97 \lg \lg n - 1.11$ optimizes T-depth. At the scale we estimate, this is only noticeable in the leading coefficient of the cost. We found a 22% reduction in T-depth at 384 bits, for example.

Windowing adds a significant cost of roughly $n + k$ auxiliary qubits, but the full elliptic curve point addition circuit has enough unused auxiliary qubits during any multiplication that this does not make a difference.

Point addition. Windowing requires 2 extra registers as the cache to load the precomputed points. We use the components of the second point three times during point addition (see Figure 9). We could perform the look-up once and keep the values, increasing total circuit width by two registers. Alternatively, we can fit the look-ups within the existing space. As Figure 9 shows, at every point where x_2 or y_2 are added, the circuit has spare auxiliary qubits available. Thus, we can

perform the look-up, add the point to the quantum register, then uncompute the quantum look-up to free the qubits for the expensive modular division. This requires us six look-ups (including uncomputing) rather than just two, but uses no extra registers.

With a window size of ℓ, including sign bit, each look-up costs $4 \cdot 2^{\ell-1}$ T gates and T-depth. The windowing saves us $\ell - 1$ point additions. If point addition costs A T gates, we would expect $\ell \approx \lg(\mathsf{A}/24)$ to be the optimal value, leading to a factor ℓ reduction in T-gate cost.

D Numerical results and interpolations

We also estimated the costs for the basic operations that comprise the elliptic curve addition.

With Q#, we counted resources for arithmetic operations for all bit sizes from 4 up to 64, then for random bit sizes between 65 and 2048. Based on theoretical analysis and examining the best fit, we decided on the best model and then fit the data with linear regression. Tables 2 and 3 show the results.

For modular arithmetic, we counted resources for all bit sizes from 4 to 64, then the common elliptic curve bit sizes 110, 160, 224, 256, 384, and 521. These were the only larger bit sizes we estimated due to the high computational cost of performing these estimates. With linear regression we estimated only the leading term of the cost growth. Table 4 contains these costs.

We found that addition with the DKRS adder had very different depths for numbers with an odd bitlength compared to an even bitlength. This behaviour is likely due to the structure of the binary tree for carry bit propagation. Odd bitlengths were much cheaper, so we opted to simply allocate an extra qubit to add two numbers of even bitlengths.

We also noticed that addition and modular addition are substantially more expensive when controlled, while constant addition and modular doubling costs increase only negligibly. The T-depth of the uncontrolled DKRS adder increases monotonically with bitlength. The data for the controlled DKRS adder is much noisier. Hence, the apparent growth of the $\lg n$ term is slower than the controlled DKRS adder, which is nonsensical; this is likely just an artifact of the noise.

For the final set of formulas for the entire cost of Shor's algorithm, we extrapolated the costs for a single point addition, then performed the same optimization as Section 6 describes. Though the exact cost was not convex, it should be $O(n/\lg n)$ times the cost of a single addition, and indeed such a curve fit the data well ($r^2 > 0.98$ for the depth and gate costs).

E Automatic compilation for aggressive T-count and T-depth reduction

In this section, we motivate automatic compilation methods to drastically reduce the T-count and the T-depth if we allow a significant increase in circuit width.

The modular multiplication followed by an addition is one of the most costly operations in the overall algorithm. It is implemented as a unitary $U : |x\rangle|y\rangle|z\rangle|0\rangle \mapsto |x\rangle|y\rangle|(xy + z) \bmod p\rangle|0\rangle$ that adds the result of the multiplication of two numbers x and y onto a third number z, all in Montgomery form with bit-width n and modulus p. We apply the following procedure to automatically obtain a quantum circuit for this operation:

1. We generate logic networks over the gate basis $\{\text{AND}, \text{XOR}, \text{INV}\}$, called Xor-And-inverter Graphs (XAGs), for the functions $xy \bmod p$, $(x + y) \bmod p$, and $(x - y) \bmod p$, where x and y are integers in Montgomery form.
2. We apply the logic optimization method described in [33] to minimize the number of AND gates in the XAGs.
3. The optimized XAGs are then translated into out-of-place quantum circuits using the method in [19], which requires 4 T gates for each AND gate in the XAG. Optimizing these circuits for depth requires roughly 2 qubits for each AND gate in the XAG, by using the AND gate construction from Section 3.

Circuit	Metric	Depth	Gates	Qubits		
				In/out	Auxiliary	Total
Fanin						
Low W	T	$13.50n - 30.61$	$56.0n - 168$	n	2	$n + 2$
	All	$40.5n - 89.8$	$130n - 382$			
Low T/D	T	$0.99 \lg n + 0.48$	$4.00n - 4.00$	n	$n + 1$	$2n + 1$
	All	$6.98 \lg n + 8.38$	$22.2n - 16.2$			
Addition						
Low W	T	$9.00n - 4.00$	$14.0n - 7.0$	$2n$	1	$2n + 1$
	All	$31.0n - 15.0$	$47.0n - 22.0$			
Low T	T	$1.00n + 0.00$	$4.00n + 0.00$	$2n$	$n + 2$	$3n + 2$
	All	$12.0n + 0.2$	$30.2n + 0.5$			
Low D	T	$3.86 \lg n + 1.16$	$39.9n - 206$	$2n$	$3n - 6.8$	$5n - 6.8$
	All	$28.8 \lg n + 14.5$	$172n - 853$			
Addition (Controlled)						
Low W	T	$14.00n + 9.00$	$21.0n + 14.0$	$2n + 1$	2	$2n + 3$
	All	$45.0n + 13.8$	$65.0n + 36.0$			
Low T	T	$11.0n - 0.0$	$18.0n + 0.0$	$2n + 1$	$n + 2$	$3n + 3$
	All	$39.0n + 3.0$	$66.3n + 4.7$			
Low D	T	$2.38 \lg n + 24.79$	$53.9n - 191$	$2n + 1$	$3n - 2.8$	$5n - 1.8$
	All	$25.6 \lg n + 67.5$	$222n - 815$			
Comparator						
Low W	T	$9.00n - 4.00$	$14.0n - 7.0$	$2n$	1	$2n + 1$
	All	$30.0n - 12.0$	$48.0n - 22.0$			
Low T	T	$1.00n + 0.00$	$4.00n + 0.00$	$2n$	$n + 2$	$3n + 2$
	All	$13.0n - 1.4$	$32.2n + 1.0$			
Low D	T	$3.75 \lg n + 1.27$	$35.9n - 170$	$2n$	$2n - 1.4$	$4n - 1.4$
	All	$27.7 \lg n + 3.9$	$148n - 674$			
Comparator (Controlled)						
Low W	T	$9.00n + 10.00$	$14.0n + 14.0$	$2n + 1$	3	$2n + 4$
	All	$30.0n + 30.0$	$48.0n + 36.0$			
Low T	T	$1.00n + 5.00$	$4.00n + 7.00$	$2n + 1$	$n + 2$	$3n + 3$
	All	$13.0n + 12.7$	$32.2n + 13.3$			
Low D	T	$3.08 \lg n + 12.7$	$35.9n - 155$	$2n + 1$	$2n - 1.4$	$2n - 0.4$
	All	$25.5 \lg n + 27.5$	$148n - 605$			
Addition (No Carry)						
Low W	T	$9.00n - 8.00$	$14.0n - 14.0$	$2n$	0	$2n$
	All	$31.0n - 30.0$	$47.0n - 44.0$			
Low T	T	$1.00n + 0.00$	$4.00n + 0.00$	$2n$	$n + 1$	$3n + 1$
	All	$12.0n - 0.8$	$30.2n - 0.1$			
Low D	T	$4.03 \lg n - 0.49$	$39.9n - 251$	$2n$	$3n - 10.1$	$5n - 10.1$
	All	$29.4 \lg n + 9.2$	$172n - 1032$			
Addition (No Carry, Controlled)						
Low W	T	$14.00n - 9.00$	$21.0n - 14.0$	$2n + 1$	0	$2n + 1$
	All	$45.0n - 44.2$	$65.0n - 42.0$			
Low T	T	$11.0n - 5.0$	$18.0n - 7.0$	$2n + 1$	$n + 1$	$3n + 2$
	All	$39.0n - 12.0$	$66.3n - 22.9$			
Low D	T	$3.73 \lg n + 4.35$	$53.9n - 258$	$2n + 1$	$3n - 6.1$	$5n - 5.1$
	All	$29.2 \lg n + 24.8$	$222n - 1054$			

Table 2: Basic arithmetic estimates.

Circuit	Metric	Depth	Gates	Qubits		
				In/out	Auxiliary	Total
Constant Addition						
Low W	T	$53.9n - 307$	$44n \lg n - 7252$	n	1.6	$n + 1.6$
	All	$176n - 767$	$122n \lg n - 18983$			
Low T	T	$1.00n + 0.00$	$4.00n + 0.00$	n	$2n + 1$	$3n + 1$
	All	$12.0n + 1.7$	$28.3n + 80.7$			
Low D	T	$4.03 \lg n - 1.49$	$35.9n - 247$	n	$3n - 10.1$	$4n - 10.1$
	All	$29.2 \lg n + 1.7$	$147n - 940$			
Constant Addition (Controlled)						
Low W	T	$19n \lg n - 1870$	$45n \lg n - 6120$	$n + 1$	0	$n + 1$
	All	$62n \lg n - 5721$	$126n \lg n - 16076$			
Low T	T	$1.00n + 0.00$	$4.00n + 0.00$	$n + 1$	$2n + 1$	$3n + 2$
	All	$12.0n + 3.8$	$28.3n + 87.7$			
Low D	T	$4.03 \lg n - 0.49$	$42.9n - 254$	$n + 1$	$3n - 6.1$	$4n - 5.1$
	All	$29.2 \lg n + 9.6$	$173n - 950$			
Modular Addition						
Low W	T	$30.8n \lg n - 168$	$87.0n \lg n - 2433$	$2n$	1	$2n + 1$
	All	$101n \lg n - 779$	$261n \lg n - 8827$			
Low T	T	$4.00n - 2.00$	$16.0n + 4.0$	$2n$	$2n + 4$	$4n + 4$
	All	$49.0n - 23.1$	$111n + 147$			
Low D	T	$15.6 \lg n + 1.1$	$155n - 843$	$2n$	$3n - 3.7$	$5n - 3.7$
	All	$111 \lg n + 29$	$633n - 3268$			
Modular Addition (Controlled)						
Low W	T	$31.5n \lg n - 210$	$87.8n \lg n - 2280$	$2n + 1$	2	$2n + 3$
	All	$102n \lg n - 523$	$263n \lg n - 8434$			
Low T	T	$14.0n + 4.0$	$30.0n + 11.0$	$2n + 1$	$2n + 4$	$4n + 5$
	All	$76.0n + 0.5$	$147n + 164$			
Low D	T	$13.4 \lg n + 36.2$	$169n - 812$	$2n + 1$	$3n - 1.8$	$5n - 0.8$
	All	$106 \lg n + 102$	$683n - 3179$			
Modular Doubling						
Low W	T	$28.3n \lg n - 484$	$83.9n \lg n - 2526$	n	2.5	$n + 2.5$
	All	$92.3n \lg n - 1382$	$252n \lg n - 9314$			
Low T	T	$2.00n + 10.00$	$15.0n + 4.0$	n	$2n + 4$	$3n + 4$
	All	$24.0n + 32.8$	$79.8n + 165.4$			
Low D	T	$7.98 \lg n + 8.75$	$85.8n - 466$	n	$3n - 3.7$	$4n - 3.7$
	All	$58.0 \lg n + 50.9$	$343n - 1757$			
Modular Doubling (Controlled)						
Low W	T	$28.9n \lg n - 416$	$84.7n \lg n - 2451$	$n + 1$	1.5	$n + 2.5$
	All	$94.0n \lg n - 1185$	$254n \lg n - 9102$			
Low T	T	$2.00n + 14.0$	$15.0n + 11.0$	$n + 1$	$2n + 5$	$3n + 6$
	All	$24.0n + 51.9$	$79.9n + 185$			
Low D	T	$7.98 \lg n + 13.8$	$85.8n - 459$	$n + 1$	$3n - 3.7$	$4n - 2.7$
	All	$60.0 \lg n + 61.8$	$343n - 1723$			

Table 3: Alternate arithmetic estimates.

Circuit	Metric	Depth	Gates	Qubits		
				In/out	Auxiliary	Total
Modular Squaring						
Low W	T	$3.54n^2 \lg n + 1.32 \cdot 2^{15}$	$5.62n^2 \lg n + 1.24 \cdot 2^{16}$	$2n$	$3n + 15.1$	$5n + 15.1$
	All	$11.5n^2 \lg n + 1.09 \cdot 2^{17}$	$17.3n^2 \lg n + 1.94 \cdot 2^{17}$			
Low T	T	$21.4n^2 + 1.43 \cdot 2^{10}$	$37.8n^2 + 1.37 \cdot 2^{12}$	$2n$	$4n + 12.3$	$6n + 12.3$
	All	$75.7n^2 + 1.46 \cdot 2^{13}$	$143n^2 + 1.74 \cdot 2^{15}$			
Low D	T	$11.5n \lg n + 1.88 \cdot 2^9$	$124n^2 + 1.29 \cdot 2^{12}$	$2n$	$6n + 11.6$	$8n + 11.6$
	All	$86.4n \lg n + 1.40 \cdot 2^{11}$	$509n^2 + 1.56 \cdot 2^{15}$			
Modular Multiplication						
Low W	T	$3.54n^2 \lg n + 1.35 \cdot 2^{15}$	$5.62n^2 \lg n + 1.28 \cdot 2^{16}$	$3n$	$3n + 15.0$	$6n + 15.0$
	All	$11.5n^2 \lg n + 1.12 \cdot 2^{17}$	$17.3n^2 \lg n + 2.00 \cdot 2^{17}$			
Low T	T	$21.4n^2 + 1.35 \cdot 2^{10}$	$37.8n^2 + 1.40 \cdot 2^{12}$	$3n$	$4n + 12.2$	$7n + 12.2$
	All	$75.7n^2 + 1.45 \cdot 2^{13}$	$143n^2 + 1.73 \cdot 2^{15}$			
Low D	T	$11.4n \lg n + 1.84 \cdot 2^9$	$124n^2 + 1.29 \cdot 2^{12}$	$3n$	$6n + 11.6$	$9n + 11.6$
	All	$84.8n \lg n + 1.42 \cdot 2^{11}$	$509n^2 + 1.54 \cdot 2^{15}$			
Modular Inversion						
Low W	T	$72n^2 \lg n + 1.76 \cdot 2^{16}$	$236n^2 \lg n + 1.30 \cdot 2^{18}$	$2n$	$5n + 1 \lfloor \lg n \rfloor + 11$	$7n + 1 \lfloor \lg n \rfloor + 11$
	All	$233n^2 \lg n + 1.44 \cdot 2^{18}$	$703n^2 \lg n + 1.49 \cdot 2^{19}$			
Low T	T	$162n^2 + 1.09 \cdot 2^{12}$	$496n^2 + 1.70 \cdot 2^{14}$	$2n$	$7n + 1 \lfloor \lg n \rfloor + 11$	$9n + 1 \lfloor \lg n \rfloor + 11$
	All	$592n^2 + 1.42 \cdot 2^{14}$	$1789n^2 + 1.96 \cdot 2^{16}$			
Low D	T	$84.4n \lg n + 1.10 \cdot 2^{12}$	$1062n^2 - 1.28 \cdot 2^{16}$	$2n$	$8n + 12.8$	$10n + 12.8$
	All	$529n \lg n + 1.83 \cdot 2^{13}$	$4162n^2 - 1.28 \cdot 2^{18}$			
Modular Division						
Low W	T	$76n^2 \lg n + 1.26 \cdot 2^{17}$	$243n^2 \lg n + 1.66 \cdot 2^{18}$	$3n$	$5n + 3.1 \lfloor \lg n \rfloor + 9.7$	$8n + 3.1 \lfloor \lg n \rfloor + 9.7$
	All	$245n^2 \lg n + 1.02 \cdot 2^{19}$	$722n^2 \lg n + 1.03 \cdot 2^{20}$			
Low T	T	$184n^2 + 1.46 \cdot 2^{12}$	$534n^2 + 1.03 \cdot 2^{15}$	$2n$	$8n + 1 \lfloor \lg n \rfloor + 11$	$10n + 1 \lfloor \lg n \rfloor + 11$
	All	$667n^2 + 1.12 \cdot 2^{15}$	$1932n^2 + 1.43 \cdot 2^{17}$			
Low D	T	$95.8n \lg n + 1.35 \cdot 2^{12}$	$1186n^2 - 1.13 \cdot 2^{16}$	$2n$	$9n + 18.8$	$11n + 18.8$
	All	$614n \lg n + 1.12 \cdot 2^{14}$	$4672n^2 - 1.01 \cdot 2^{18}$			
Signed Windowed Elliptic Curve Point Addition (window size 8)						
Low W	T	$144n^2 \lg n + 1.19 \cdot 2^{19}$	$503n^2 \lg n + 1.26 \cdot 2^{20}$	$2n$	$6n + 3.8 \lfloor \lg n \rfloor + 17.1$	$8n + 3.8 \lfloor \lg n \rfloor + 17.1$
	All	$465n^2 \lg n + 1.98 \cdot 2^{20}$	$1423n^2 \lg n + 1.99 \cdot 2^{21}$			
Low T	T	$432n^2 + 1.07 \cdot 2^{14}$	$1182n^2 + 1.41 \cdot 2^{16}$	$2n$	$8n + 1.5 \lfloor \lg n \rfloor + 18.9$	$10n + 1.5 \lfloor \lg n \rfloor + 18.9$
	All	$1562n^2 + 1.85 \cdot 2^{16}$	$4306n^2 + 1.31 \cdot 2^{19}$			
Low D	T	$226n \lg n + 1.77 \cdot 2^{13}$	$2746n^2 - 1.31 \cdot 2^{16}$	$2n$	$9n + 28.6$	$11n + 28.6$
	All	$1485n \lg n + 1.60 \cdot 2^{15}$	$10891n^2 - 1.39 \cdot 2^{16}$			
Shor's algorithm (with signed windowed point addition)						
Low W	T	$120n^3 - 1.67 \cdot 2^{22}$	$436n^3 - 1.05 \cdot 2^{26}$	$2n$	$6n + 10.2 \lfloor \lg n \rfloor - 1.0$	$8n + 10.2 \lfloor \lg n \rfloor - 1.0$
	All	$509n^3 - 1.84 \cdot 2^{27}$	$2800n^3 - 1.08 \cdot 2^{31}$			
Low T	T	$389n^3 / \lg n - 1.70 \cdot 2^{22}$	$1115n^3 / \lg n - 1.08 \cdot 2^{24}$	$2n$	$8n + 7.4 \lfloor \lg n \rfloor + 1.3$	$10n + 7.4 \lfloor \lg n \rfloor + 1.3$
	All	$1701n^3 / \lg n - 1.23 \cdot 2^{24}$	$6262n^3 / \lg n - 1.72 \cdot 2^{24}$			
Low D	T	$285n^2 - 1.54 \cdot 2^{17}$	$3120n^3 / \lg n - 1.49 \cdot 2^{27}$	$2n$	$9n + 3.9 \lfloor \lg n \rfloor + 16.5$	$11n + 3.9 \lfloor \lg n \rfloor + 16.5$
	All	$2523n^2 + 1.10 \cdot 2^{20}$	$12478n^3 / \lg n - 1.25 \cdot 2^{29}$			

Table 4: Estimates for non-linear modular arithmetic, elliptic curve point addition and Shor's algorithm.

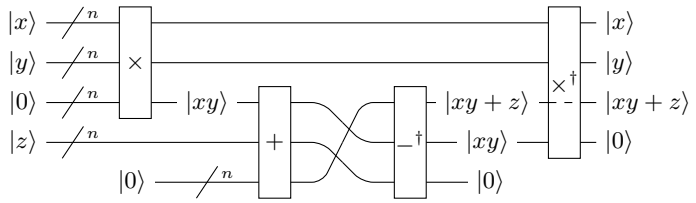


Fig. 13: Quantum circuit that implements $xy+z \bmod p$, using out-of-place constructions for modular multiplication, modular addition, and modular subtraction.

4. The automatically generated unitaries are composed as described in Figure 13, which uses the technique shown in Figure 11 to turn the out-of-place addition and subtraction into an in-place addition.

Table 5: Comparison of resource costs between a manual and automatic construction to implement $|xy + z \bmod p\rangle$.

	Manual construction			Automatic construction		
Bit-width	T-count	T-depth	Width	T-count	T-depth	Width
256	8,176,739	50,253	2,319	1,576,296	1,542	394,588
384	18,322,671	76,125	3,470	3,550,552	2,310	888,408
521	33,751,240	137,183	4,702	6,535,384	3,132	1,634,890

Table 5 lists the resource costs in terms of T-count, T-depth, and circuit width, for both the manual construction and the automatic construction. Several factors of reduction in T-count and T-depth are possible, while the increase in the number of qubits is significant. However, such a design point can be of high interest, in particular when combined with automatic quantum memory strategies, e.g., pebbling [20], that can find intermediate trade-off points that lie in between the manual and automatic construction.