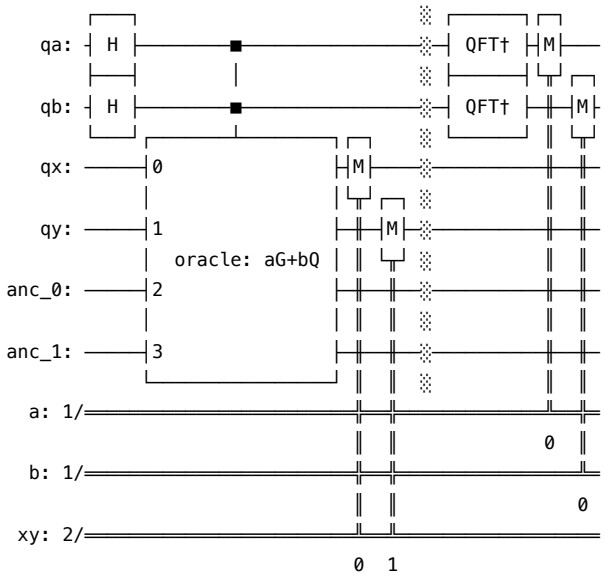
Details of the Quantum Circuit

The overall structure of the quantum circuit utilizing Shor's algorithm is shown below.



The central operation of this circuit is the calculation of the elliptic-curve point combination aG + bQ. Below we explain how this is constructed; as an example we describe the 8-bit case.

If the integers a and b are represented by bit strings

$a = |a_7 a_6 \dots a_0\rangle\,, \quad b = |b_7 b_6 \dots b_0\rangle,$

then

$$aG + bQ = |a_7\rangle\, G^{2^7} + |a_6\rangle\, G^{2^6} + \cdots + |a_0\rangle\, G + |b_7\rangle\, Q^{2^7} + \cdots + |b_0\rangle\, Q$$

That is, we construct the superposition of point additions by adding the contributions of each bit. In the naive decomposition this would require 15 point additions for the 8-bit example.

Because ECC point addition is a relatively heavy quantum subroutine, Version 3 of this work reduced the number of quantum point additions by precomputing small superpositions classically. Concretely, Version 3 computed

$$aG + bQ = |a_7 b_7\rangle\, f(a_7, b_7) + |a_6 b_6\rangle\, f_2(a_6, b_6) + \cdots + |a_0 b_0\rangle\, f(a_0, b_0)$$

where classically computed maps $f(a_i, b_i) = |a_i\rangle\, G^{2^i} + |b_i\rangle\, Q^{2^i}$ (a superposition of up to four coordinates) are loaded into quantum registers. This reduces the number of quantum point additions from 15 to 7.

In Version 4 (this submission) we further reduce the number of quantum point additions by grouping multiple bits and preparing larger precomputed superpositions classically. The decomposition used here is

$$\begin{aligned} aG + bQ = {} & |a_1 a_2 a_3\rangle\, g_3(a_1, a_2, a_3) + |a_4 a_5 a_6\rangle\, g_3(a_4, a_5, a_6) + |a_0 a_7\rangle\, g_2(a_0, a_7) \\ & + |b_1 b_2 b_3\rangle\, q_3(b_1, b_2, b_3) + |b_4 b_5 b_6\rangle\, q_3(b_4, b_5, b_6) + |b_0 b_7\rangle\, q_2(b_0, b_7) \end{aligned}$$

Here the classically prepared functions create the following superpositions which are then loaded into quantum registers:

$$\begin{aligned} g_3(a_i, a_j, a_k) &= |a_i\rangle\, G^{2^i} + |a_j\rangle\, G^{2^j} + |a_k\rangle\, G^{2^k} \\ g_2(a_i, a_j) &= |a_i\rangle\, G^{2^i} + |a_j\rangle\, G^{2^j} \\ q_3(b_i, b_j, b_k) &= |b_i\rangle\, Q^{2^i} + |b_j\rangle\, Q^{2^j} + |b_k\rangle\, Q^{2^k} \\ q_2(b_i, b_j) &= |b_i\rangle\, Q^{2^i} + |b_j\rangle\, Q^{2^j} \end{aligned}$$

Using this approach, the number of quantum point additions is reduced to five.

As an additional micro-optimization, when particular control branches that add complexity (such as doubling the same coordinate or adding the inverse leading to the point at infinity O) are provably unnecessary, we replace the point-addition circuit with a simpler variant. Concretely we use two circuit modes:

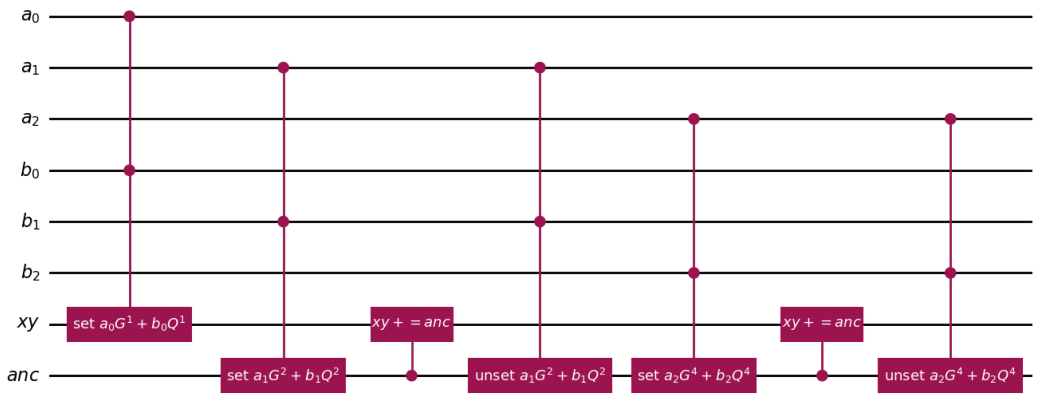
$$\begin{aligned} add_{normal}: &\ |a_1a_2a_3\rangle\, g_3(a_1,a_2,a_3) + |a_4a_5a_6\rangle\, g_3(a_4,a_5,a_6) \\ add_{nodbl}: &\ |a_1a_2\ldots a_6\rangle\, g_6(a_1,a_2,\ldots,a_6) + |a_0a_7\rangle\, g_2(a_0,a_7) \end{aligned}$$

When creating the superposition for aG, we select between add_{normal} (no doubling or inverse additions occur) and add_{nodbl} (inverse additions can occur but doubling does not). This choice reduces the number of quantum gates; the same logic is applied symmetrically when building bQ.

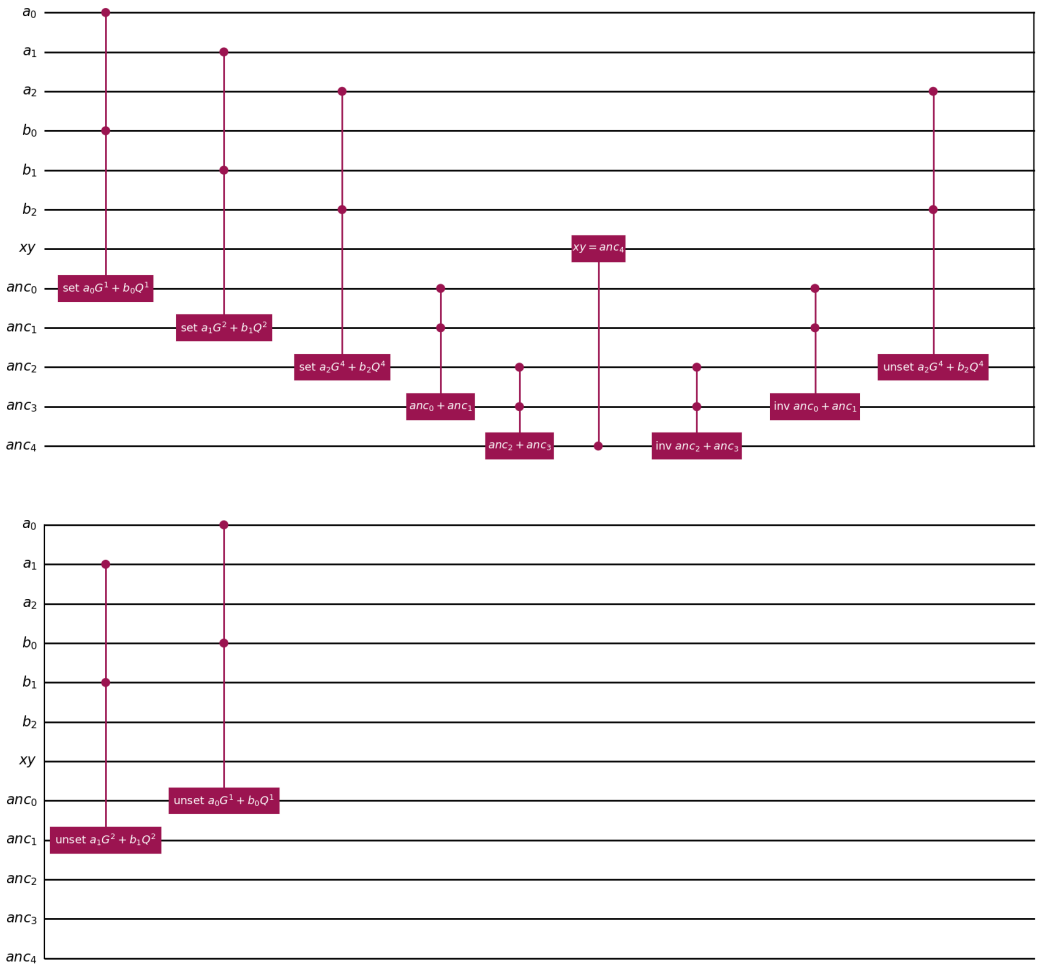
Two circuit variants

This implementation provides two variants of the quantum circuit:

- compact: minimizes the number of qubits at the cost of deeper circuits.
  - The circuit uncomputes after each point addition, allowing ancilla reuse.
  - Effective on simulators where memory scales with qubit count.
  - Illustration:



- wide: increases qubit count to reduce circuit depth.
  - All ECC additions are performed (in parallel where possible), then the state is copied and finally uncomputed.
  - Effective for real quantum hardware that benefits from parallelism.
  - Illustration:



With current hardware noise levels neither variant produced the expected measurement outcomes, but we expect that future improvements in quantum hardware will enable correct execution for one of these approaches.

Quantum Circuit: ECC Addition

ECC addition is a very complex quantum circuit.
The addition $|xx\rangle + |yy\rangle \mapsto |zz\rangle$ is realized with the following quantum register structure:

$$|xx\rangle\,|yy\rangle\,|zz\rangle\,|ancilla\rangle$$

Each quantum register is further decomposed as follows:

$$\begin{aligned}|xx\rangle &= |y_1, x_1\rangle \\ |yy\rangle &= |y_2, x_2\rangle \\ |zz\rangle &= |y_3, x_3\rangle \\ |ancilla\rangle &= |f_1, f_2, f_3, f_4, carry, \lambda, a_1, a_2\rangle\end{aligned}$$

Details of ancilla:

- f_1: Flag for $(x_1, y_1) = O$
- f_2: Flag for $(x_2, y_2) = O$
- f_3: Flag for $(x_1, y_1) + (x_2, y_2) = O$
- f_4: Flag for $(x_1, y_1) = (x_2, y_2)$ (addition of identical coordinates)
- $carry$: Carry bit used in various calculations
- λ: Quantum register for $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ or $\frac{3x_1^2 + a}{2y_1}$ during ECC addition
- a_1: Used for Y coordinate calculation in the judgment $(x_1, y_1) + (x_2, y_2) = O$
- a_2: Quantum register for modular inverse calculation in λ denominator

In Version 3 and later, by efficiently using the registers described above, we achieved a significant reduction in the number of quantum gates.
The detailed computation steps are shown below:

Target Register	Control Bits	Quantum Operation	Value After	Remarks
f_1	y_1, x_1	$\|0\rangle \mapsto \|1\rangle, \text{if}(x_1 = 0, y_1 = 0)$	$\begin{cases} \|1\rangle, & \text{if}(x_1 = 0, y_1 = 0) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set flag for $(x_1, y_1) = O$
f_2	y_2, x_2	$\|0\rangle \mapsto \|1\rangle, \text{if}(x_2 = 0, y_2 = 0)$	$\begin{cases} \|1\rangle, & \text{if}(x_2 = 0, y_2 = 0) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set flag for $(x_2, y_2) = O$
x_3	f_1, f_2, x_1, x_2	$\|0\rangle \mapsto \begin{cases} \|x_2\rangle, & \text{if}(f_1 = 1) \\ \|x_1\rangle, & \text{if}(f_2 = 1) \end{cases}$	$\begin{cases} \|x_2\rangle, & \text{if}(f_1 = 1) \\ \|x_1\rangle, & \text{if}(f_2 = 1) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set x_3 if either (x_1, y_1) or (x_2, y_2) is O
y_3	f_1, f_2, y_1, y_2	$\|0\rangle \mapsto \begin{cases} \|y_2\rangle, & \text{if}(f_1 = 1) \\ \|y_1\rangle, & \text{if}(f_2 = 1) \end{cases}$	$\begin{cases} \|y_2\rangle, & \text{if}(f_1 = 1) \\ \|y_1\rangle, & \text{if}(f_2 = 1) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set y_3 if either (x_1, y_1) or (x_2, y_2) is O
x_3	f_1, x_2	$\|0\rangle \mapsto \|-x_2 \mod p\rangle, \text{if}(f_1 = 0)$	$\begin{cases} \|x_2\rangle, & \text{if}(f_1 = 1) \\ \|x_1\rangle, & \text{if}(f_2 = 1) \\ \|-x_2 \mod p\rangle, & \text{otherwise} \end{cases}$	Compute $-x_2$ term in $x_3 = \lambda^2 - x_1 - x_2$
x_2	x_1	$\|x_2\rangle \mapsto \|x_2 - x_1 \mod p\rangle$	$\|x_2 - x_1 \mod p\rangle$	Compute denominator of λ
a_1	y_1, y_2	$\|0\rangle \mapsto \|y_1 + y_2 - p\rangle$	$\|y_1 + y_2 - p\rangle$	For checking $(x_1, y_1) + (x_2, y_2) = O$
y_2	y_1	$\|y_2\rangle \mapsto \|y_2 - y_1 \mod p\rangle$	$\|y_2 - y_1 \mod p\rangle$	Compute numerator of λ
f_3	x_2, a_1	$\|0\rangle \mapsto \|1\rangle, \text{if}(x_2 = 0, a_1 = 0)$	$\begin{cases} \|1\rangle, & \text{if}(x_1 = x_2, y_1 + y_2 = p) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set flag for $(x_1, y_1) + (x_2, y_2) = O$

Target Register	Control Bits	Quantum Operation	Value After	Remarks
f_4	x_2, y_2	$\|0\rangle \mapsto \|1\rangle, \text{if}(x_2 = 0, y_2 = 0)$	$\begin{cases} \|1\rangle, & \text{if}(x_1 = x_2, y_1 = y_2) \\ \|0\rangle, & \text{otherwise} \end{cases}$	Set flag for identical coordinates
x_3	f_3, x_1	$\|x_3\rangle \mapsto \|x_3 + x_1\rangle, \text{if}(f_3 = 1)$	$\begin{cases} \|x_2\rangle, & \text{if}(f_1 = 1) \\ \|x_1\rangle, & \text{if}(f_2 = 1) \\ \|0\rangle, & \text{if}(f_3 = 1) \\ \|-x_2 \mod p\rangle, & \text{otherwise} \end{cases}$	Set $x_3 = 0$ if $(x_1, y_1) + (x_2, y_2) = O$
y_2	f_4, x_1	$\|0\rangle \mapsto \|3x_1^2 + a \mod p\rangle, \text{if}(f_4 = 1)$	$\begin{cases} \|3x_1^2 + a \mod p\rangle, & \text{if}(f_4 = 1) \\ \|y_2 - y_1 \mod p\rangle, & \text{otherwise} \end{cases}$	Numerator of λ for identical coordinates
x_2	f_4, y_1	$\|0\rangle \mapsto \|2y_1 \mod p\rangle, \text{if}(f_4 = 1)$	$\begin{cases} \|2y_1 \mod p\rangle, & \text{if}(f_4 = 1) \\ \|x_2 - x_1 \mod p\rangle, & \text{otherwise} \end{cases}$	Denominator of λ for identical coordinates
a_2	x_2	$\|0\rangle \mapsto \|x_2^{p-2} \mod p\rangle \|junk\rangle$	$\begin{cases} \|(2y_1)^{-1} \mod p\rangle \|junk\rangle, & \text{if}(f_4 = 1) \\ \|(x_2 - x_1)^{-1} \mod p\rangle \|junk\rangle, & \text{otherwise} \end{cases}$	Compute modular inverse for denominator of λ
λ	y_2, a_2	$\|0\rangle \mapsto \|y_2 \cdot a_2 \mod p\rangle$	$\begin{cases} \|(3x_1^2 + a) \cdot (2y_1)^{-1} \mod p\rangle, & \text{if}(f_4 = 1) \\ \|(y_2 - y_1) \cdot (x_2 - x_1)^{-1} \mod p\rangle, & \text{otherwise} \end{cases}$	Compute λ
x_3	f_1, f_2, f_3, x_1	$\|x_3\rangle \mapsto \|x_3 + \lambda^2 - x_1 \mod p\rangle, \text{if}(f_1 = 0, f_2 = 0, f_3 = 0)$	$\begin{cases} \|x_2\rangle, & \text{if}(f_1 = 1) \\ \|x_1\rangle, & \text{if}(f_2 = 1) \\ \|0\rangle, & \text{if}(f_3 = 1) \\ \|\lambda^2 - x_2 - x_1 \mod p\rangle, & \text{otherwise} \end{cases}$	Final calculation of x_3
x_1	x_3	$\|x_1\rangle \mapsto \|x_1 - x_3 \mod p\rangle$	$\|x_1 - x_3 \mod p\rangle$	Prepare for y_3 calculation
y_3	$f_1, f_2, f_3, x_1, y_1, \lambda$	$\|0\rangle \mapsto \|\lambda \cdot x_1 - y_1 \mod p\rangle, \text{if}(f_1 = 0, f_2 = 0, f_3 = 0)$	$\begin{cases} \|y_2\rangle, & \text{if}(f_1 = 1) \\ \|y_1\rangle, & \text{if}(f_2 = 1) \\ \|0\rangle, & \text{if}(f_3 = 1) \\ \|\lambda(x_1 - x_3) - y_1 \mod p\rangle, & \text{otherwise} \end{cases}$	Final calculation of y_3

After obtaining the result, the ECC addition circuit must be uncomputed to return all ancilla to $|0\rangle$.
In the wide variant uncomputation is performed after all $aG + bQ$ calculations, which requires extra qubits proportional to the number of intermediate additions. In the compact variant each addition is uncomputed immediately, enabling efficient qubit reuse.

The compact variant performs the addition as $|xx\rangle\,|yy\rangle\,|0\rangle\,|0\rangle -> \cdots -> |xx + yy\rangle\,|yy\rangle\,|0\rangle\,|0\rangle$ while ensuring ancilla return to $|0\rangle$ by the following sequence:

1. $|xx\rangle\,|yy\rangle\,|0\rangle\,|0\rangle$
2. $|xx\rangle\,|yy\rangle\,|0\rangle\,|xx + yy, junk\rangle$ (ECC addition)
3. $|xx\rangle\,|yy\rangle\,|xx + yy\rangle\,|xx + yy, junk\rangle$ (copy result to third register)
4. $|xx\rangle\,|yy\rangle\,|xx + yy\rangle\,|0\rangle$ (uncompute ECC addition to return fourth register to $|0\rangle$)
5. $|xx + yy\rangle\,|yy\rangle\,|xx\rangle\,|0\rangle$ (SWAP to move result)
6. $|xx + yy\rangle\,|-yy\rangle\,|xx\rangle\,|0\rangle$ (map $|yy\rangle \mapsto |-yy\rangle$ to prepare subtraction)
7. $|xx + yy\rangle\,|-yy\rangle\,|xx\rangle\,|xx, junk\rangle$ (ECC addition to recompute $|xx\rangle$)

8. $|xx + yy\rangle\ |-yy\rangle\ |0\rangle\ |xx, junk\rangle$ (clear third register)
9. $|xx + yy\rangle\ |-yy\rangle\ |0\rangle\ |0\rangle$ (uncompute to clear fourth register)
10. $|xx + yy\rangle\ |yy\rangle\ |0\rangle\ |0\rangle$ (restore $|-yy\rangle$ to original)

Note that for additions where doubling or inverse-based addition cannot result in the point at infinity O (add_{normal}, add_{nodbl}), we omit unnecessary flags and conditional branches to reduce the number of quantum gates.

Subcircuits

The implementation contains a set of reusable subcircuits used throughout the ECC arithmetic:

- Increment: $|x\rangle_n \mapsto |x+1\rangle$ — arbitrary-width increment.
- Constant addition: $|x\rangle_n \mapsto |x+a\rangle$ — implemented from increment building blocks.
- Register addition: $|x\rangle_n |y\rangle_m \mapsto |x\rangle |y+x\rangle$ — built from controlled increments.
- Modular constant addition: $|x\rangle_{n+1} \mapsto |x+a \bmod p\rangle$ — implemented such that carries are returned to $|0\rangle$.
- Modular register addition: $|x\rangle_n |y\rangle_{n+1} \mapsto |x\rangle |y+x \bmod p\rangle$ — with carry cleared.
- Modular negation: $|x\rangle_{n+1} \mapsto |-x \bmod p\rangle$.
- Set inverse into another register: $|x\rangle_n |0\rangle_n \mapsto |x\rangle |-x \bmod p\rangle$.
- Modular squaring: $|x\rangle_n |0\rangle_{n+1} \mapsto |x\rangle |x^2 \bmod p\rangle$ — implemented by controlled adds of classically precomputed constants $2^{i+j} \bmod p$.
- Combined modular squaring and addition/multiplication primitives used to build higher-level routines (multiplication, exponentiation, modular inverse, etc.).

Some circuits are specialized for constant vs register operands; when $|y\rangle = |0\rangle$ the two variants produce the same result, but the constant-operated variant often uses fewer gates and is preferred when applicable.

Scale of the Quantum Circuit

The table below summarizes the number of qubits, quantum gates and depth for each ECC bit size.
The number of quantum gates is counted as one for each MCX or SWAP gate; after transpilation the real gate counts on hardware are much larger, but these numbers capture circuit scale for simulator runtime and resource estimation.

ecc bits	qbits (compact)	gates (compact)	depth (compact)	qbits (wide)	gates (wide)	depth (wide)
3	47	3,377	3,078	76	1,720	1,542
4	71	12,971	12,548	120	6,522	6,283
5	93	47,986	46,860	223	24,010	23,412
6	117	138,286	136,175	366	62,292	40,528
7	138	338,976	335,086	618	164,926	124,419
8	163	825,873	817,349	923	387,780	229,635
9	230	1,641,514	1,634,177	1462	784,956	468,072
10	235	2,287,611	2,276,484	1635	1,115,400	749,091
11	280	4,992,898	4,968,256	2206	2,391,440	1,358,478
12	329	6,027,756	6,010,187	2899	2,901,368	1,651,805

As shown, the compact variant increases qubit count slowly while the wide variant grows rapidly.
However, the wide variant allows more parallel execution, so as the ECC bit size increases, the circuit depth increases more slowly compared to the number of gates.

In practice the wide variant would be favorable for hardware execution, but with the current error rates even a 3-bit ECC instance is difficult to run error-free on available devices.

As a reference, the table below shows the scale of quantum gate counts after transpilation when running the circuits on real hardware (ibm_fez).

bits	type	qbits	gates	depth	sx gate	cz gate	rz gate	x gate
3	compact	47	1,012,397	545,758	536,619	253,993	220,752	1,021
3	wide	76	505,461	270,058	267,887	127,056	109,939	567
4	compact	71	4,167,068	2,246,123	2,217,253	1,048,828	897,179	3,794
4	wide	120	2,080,814	1,117,623	1,107,430	524,356	447,024	1,990
5	compact	93	17,462,887	9,461,983	9,318,314	4,412,259	3,720,215	12,081

Beyond these entries we could not execute further runs because we exceeded the hardware limits on qubit count or gate count.