SHOR'S ALGORITHM AND SECP256K1

SHAVER PHAGAN

September 1, 2026

1. Introduction

1.1. **Abstract.** The purpose of this writeup is to describe Shor's algorithm for the discrete logarithm problem on the secp256k1 curve used by the Bitcoin network and to summarize the general state of affairs regarding Bitcoin's quantum issue.

My goal is to describe Shor's algorithm in a way that lays bare the sequence of ideas necessary to implement the algorithm with a quantum circuit, while minimizing prerequisites as much as possible. My perspective is a mathematician's and emphasizes certain aspects that might be routine for computer scientists, like encoding a group law in a quantum computational register (cf. Section 4). Familiarity with abstract algebra is assumed when discussing technical details, because it is essential to quantum algorithms. I also assume familiarity with modular arithmetic and finite fields, as covered in *Programming Bitcoin*. Those interested in a bird's eye view may safely skip Sections 2, 3, and 4. These sections provide technical details, but they are not strictly necessary for developing a meaningful general idea of how Shor's algorithm for secp256k1 breaks the hardness of its discrete logarithm problem. This is covered in Section 5, which is written so that hopefully the general shape of the process will be revealed to any determined reader, even as the complexity of Shor's algorithm increases in the course of the exposition.

When discussing Bitcoin protocol in Section 6, my objective has been to highlight which aspects of transaction and blockchain protocol are *actually vulnerable* to Shor's algorithm, and then also highlight how rapidly the community is addressing these issues by covering a couple of pertinent, concrete examples.

1.2. **Mathematical Upshot.** The solution to the discrete logarithm problem for secp256k1 is encoded in phases associated to its regular representation, which can be extracted with quantum phase estimation.

From a quantum algorithmic perspective, the central idea is that the discrete logarithm problem for secp256k1 can be reformulated as a phase estimation problem, which quantum–but not classical–computers are able to handle efficiently. The heart of the solution is the quantum Fourier transform, which can be efficiently implemented by a quantum circuit. The representation theory of cyclic groups offers a clarifying perspective, on account of the fact that secp256k1 is itself a finite cyclic group. For secp256k1, the penultimate quantum phase estimation step amounts to applying the inverse quantum Fourier transform to an auxiliary register that records the effect of a certain quantum gate on the average v of the unit eigenvectors of the regular representation of secp256k1.

The incredible thing is that the gate is essentially doing a sequence of bit flips on a register configured to v, yet because of superposition, the auxiliary register retains detailed information about phases related to the underlying permutation representation, and the phases in turn encode the solution to the discrete logarithm problem. The quantum Fourier transform is used to extract the solution from the auxiliary register.

1.3. **Protocolary Upshot.** Every known potential quantum attack on the Bitcoin network requires access to public keys, so the only vulnerabilities are points of public key exposure. Notably, these include transaction broadcasts and legacy locking scripts, like the P2PK script. Hash-based protocols, like Bitcoin's proof-of-work scheme, are considered quantum safe, because Grover's algorithm–the best known quantum algorithm for hash grinding–provides only a quadratic speedup over classical algorithms. In particular, quantum computers are not believed to break preimage resistance of cryptographic hash functions like SHA-256.

The Bitcoin community is making substantial progress on addressing the quantum issue, and in principle quantum safe transaction protocols already exist, albeit with more technical overhead than would be ideal to cover the full spectrum of use cases. We discuss this and other efforts in some detail in Section 6. My hope is that this writeup will provide some clarity on what the vulnerability *actually is*, without contributing to the FUD that unfortunately already surrounds this issue. Indeed, there is substantial variation between estimates, but it seems the emergence of cryptographically relevant quantum computers could very well be a decade or more away. At the rate the community is moving, we have good reason to expect the Bitcoiners will have the quantum issue sorted out well before Q-Day.

1.4. **Acknowledgements.** I would like to thank Jeff Gotts for his support and encouragement, and also for many stimulating conversations that guided this work. I would also like to acknowledge the tremendous help that learnmeabitcoin.com and Conduition's blog were to me as I familiarized myself with Bitcoin protocol.

2. Representation Theory of Cyclic Groups

Any group we discuss is finite. An **abelian group** is a set A, endowed with a group law $\oplus$ given by an associative commutative binary operation

$$\oplus : A \times A \to A : (a, b) \mapsto a \oplus b$$

admitting a unique identity element $0 \in A$ such that $a \oplus 0 = 0 \oplus a = a$ for all $a \in A$, as well as a unique additive inverse $-a$ for each $a \in A$, defined by the rule $a \oplus (-a) = (-a) \oplus a = 0$. We will denote by V the set

$$\bigoplus_{a \in A} \mathbb{C}a = \{z_1 a_1 + \cdots + z_n a_n \mid z_i \in \mathbb{C}, a_i \in A\}$$

of all formal $\mathbb{C}$-linear combinations of elements of A. "Formal" here just means that A is a basis for V. Furthermore, we equip V with the following action by A. For $b \in A$, $v \in V$, define

$$b.v = z_1(b \oplus a_1) + \cdots + z_n(b \oplus a_n),$$

where

$$v = z_1 a_1 + \cdots + z_n a_n.$$

Notice that the action of A on V amounts to permuting coordinates according to the group law. The vector space V equipped with this permutation action of A is called the **regular representation** of A. We will be very interested later on in the regular representation of secp256k1, which is an especially nice sort of abelian group: a cyclic group.

A **cyclic group** is an abelian group C that is generated by a single element c, so that

$$C = \{0, c, 2c, ..., (n-1)c\},$$

where $2c = c \oplus c$, $3c = c \oplus c \oplus c$, etc. It follows from Lagrange's Theorem, a fundamental theorem in finite group theory, that

$$m_1 c = m_2 c$$

when $m_1 \equiv m_2 \bmod n$. In particular, $nc = 0$. For the rest of this section V will denote the regular representation of C, so that

$$V = \bigoplus_{j=0}^{n-1} \mathbb{C}(jc),$$

and the permutation action of $mc \in C$ on an element of V is given by

$$mc.\left(\sum_{j=0}^{n-1} z_j(jc)\right) = \sum_{j=0}^{n-1} z_j((m+j)c).$$

Take a moment and try to visualize the cyclic permutation of the coordinates. For instance, if we write a vector in V as a column using matrix notation, the action of c goes like this:

$$c.\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} z_{n-1} \\ z_0 \\ \vdots \\ z_{n-2} \end{bmatrix}.$$

Eigenvectors of the regular representation of a cyclic group will be central players in our later discussion of Shor's algorithm for secp256k1, so we will devote some space here to discuss them.

Recall that a linear isomorphism U of a complex vector space is **unitary** if it preserves the Hermitian inner product. In particular, the eigenvalues of a unitary transformation all have complex modulus equal to 1. Now, the regular representation of C on V determines linear isomorphisms U_b of V by the rule

$$b.v = U_b v$$

for $b \in C$ and $v \in V$. Furthermore, each U_b is a unitary matrix, because it only permutes the basis elements of V and therefore does not affect its Hermitian inner product. For $k \in \{0, ..., n-1\}$, let

$$v_k := \frac{1}{\sqrt{n}} \sum_{r=0}^{n-1} e^{2\pi i r k/n} rc.$$

Each v_k is a complex unit vector. Furthermore,

$$\begin{aligned} mc.v_k &= \frac{1}{\sqrt{n}} \sum_{r=0}^{n-1} e^{2\pi i r k/n}(r+m)c \\ &= \frac{1}{\sqrt{n}} \sum_{r=0}^{n-1} e^{2\pi i (r-m) k/n} rc \\ &= e^{-2\pi i m k/n} v_k, \end{aligned}$$

so v_k is an eigenvector of U_{mc} with eigenvalue $e^{-2\pi imk/n}$. Notice the eigenvalue amounts to a phase shift that records mk mod n. In particular, if $b.v_k = e^{-2\pi imk/n}v_k$, then we can deduce that $b = mc$ if k is invertible modulo n. Extracting information about m in this manner gets at the heart of Shor's algorithm for the discrete logarithm problem. One can check that $\{v_0, ..., v_{n-1}\}$ is in fact an orthonormal basis for V, so the v_k are *all* of the unit eigenvectors associated to the regular representation. One last remark about the v_k: their (normalized) average is remarkably simple, on account of the identity

$$\frac{1}{\ell}\sum_{k=0}^{\ell-1} e^{2\pi ijk/\ell} = \begin{cases} 1 & \text{if } \ell \mid j \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Indeed, using Equation (1), we calculate

$$v = \frac{1}{\sqrt{n}}\sum_{k=0}^{n-1} v_k = \sum_{r=0}^{n-1}\left(\frac{1}{n}\sum_{k=0}^{n-1} e^{2\pi irk/n}\right) rc = 0c, \tag{2}$$

so the average of the unit eigenvectors of the regular representation of C is equal to the identity element of C. This is relevant to Shor's algorithm for the discrete logarithm problem on secp256k1, as it allows us to *actually configure* a quantum circuit to *implement* the algorithm.

3. Quantum Registers, the Fourier Transform, and Phase Estimation

Recall that a theoretical bit is just an element of the cyclic group of order two

$$\mathbb{Z}/2\mathbb{Z} = \{0, 1\},$$

whereas a physical bit is the state of something like a transistor, whose states are described mathematically by $\mathbb{Z}/2\mathbb{Z}$. A classical register stores the state of some system in bit strings, eg. 1100011101101, in which case one often says that the register is in state 1100011101101. Just as there is a distinction between theoretical and physical bits, there is some disambiguation required for the term "qubit." Furthermore, there are quantum registers, which store the state of a some system in entangled qubits, eg. $|1100011101101\rangle, \frac{1}{\sqrt{2}}|101\rangle + \frac{e^{\pi i/2}}{\sqrt{2}}|111\rangle$.

Theoretical qubits are mathematical objects modeling certain dynamics of physical objects (eg. trapped ions, photons, or electrons), which may be leveraged for data processing in what is now called "quantum computing". The aforementioned physical objects are, in the context of quantum computing, **physical qubits**. Physical qubits are the physical objects whose states are theoretically described by theoretical qubits. Here, "theoretical" means ignoring environmental noise. In reality, physical qubits are highly influenced by environmental noise, so one has to build in redundancy by distributing the logical state across many physical qubits to make a **logical qubit**. A logical qubit is the physical analog of a transistor for the purpose of quantum programming: a programmer can configure and manipulate a logical qubit and rest assured that its state will be as predicted by the theoretical qubit, just like a transistor can be confidently engaged and disengaged by a programmer, with the resulting state being as predicted by arithmetic modulo 2.

A theoretical qubit is a unit vector in a two dimensional complex vector space. The two dimensions correspond to orthonormal basis vectors $|0\rangle$ and $|1\rangle$, and the unit normalization comes from the probabilistic interpretation of quantum mechanics: basis vectors are observable states, the probability that *some state* will be observed upon measurement is equal to 1, and this probability

is exactly the squared modulus of a vector describing the state of a quantum system. Concretely, a qubit is a vector

$$u \in \mathbb{C}^2 := \mathbb{C}\left|0\right\rangle \bigoplus \mathbb{C}\left|1\right\rangle$$

of the form $u = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$ with $|\alpha|^2 + |\beta|^2 = 1$. One forms a quantum register by entangling logical qubits, which is described mathematically by tensoring copies of $\mathbb{C}^2$. The basis of the tensor product consists of all pairwise products of elements from the constituent bases, and the state of the quantum register is still just a unit vector in the resulting complex vector space. For instance, the computational basis of

$$\mathbb{C}^2 \bigotimes \mathbb{C}^2$$

is $\{\left|0\right\rangle \otimes \left|0\right\rangle, \left|0\right\rangle \otimes \left|1\right\rangle, \left|1\right\rangle \otimes \left|0\right\rangle, \left|1\right\rangle \otimes \left|1\right\rangle\}$, which we abbreviate $\{\left|00\right\rangle, \left|01\right\rangle, \left|10\right\rangle, \left|11\right\rangle\}$. Similarly, tensoring n copies of $\mathbb{C}^2$ gives a complex vector space of dimension 2^n, with basis $\left|0 \cdot 0\right\rangle, ..., \left|1 \cdots 1\right\rangle$, where the expression inside the ket $\left|\cdot\right\rangle$ varies over the binary representations of the numbers $0, ..., 2^n - 1$. Notice that the basis states of our register are essentially just bit strings. This means we can embed a dataset into a quantum register by choosing an appropriate binary labeling of its constituents, then identifying a datum with the basis vector in a quantum register determined by its binary label. For instance, if we give a, b, c the binary labels $00, 01, 11$, respectively, then we can embed the set $\{a, b, c\}$ into a 4-dimensional quantum register, writing $\left|a\right\rangle := \left|00\right\rangle, \left|b\right\rangle := \left|01\right\rangle, \left|c\right\rangle := \left|11\right\rangle$. This is done without further comment at various points in the exposition, and the idea is refined in Section 4 when we discuss embedding a group law in a quantum register. We use quantum registers to record phase estimation, which in turn facilitates Shor's algorithm for secp256k1.

For quantum registers to be useful, we need to be able to manipulate and measure them, so we will briefly describe how to do both, starting with measurement. Let's suppose our register $\mathcal{R}$ consists of a single qubit in the state

$$\mathcal{R} = \alpha\left|0\right\rangle + \beta\left|1\right\rangle.$$

Then, the Laws of Quantum Mechanics dictate that when we measure the register, the result will be $\left|0\right\rangle$, with probability $|\alpha|^2$, or $\left|1\right\rangle$, with probability $|\beta|^2$. In particular, the outcome of measurement must be either $\left|0\right\rangle$ or $\left|1\right\rangle$. Similarly, if a register $\mathcal{R}$ consists of two subregisters $\mathcal{R}_X, \mathcal{R}_Y$, i.e.

$$\mathcal{R} = \mathcal{R}_X \otimes \mathcal{R}_Y,$$

and $\mathcal{R}$ is configured to the state

$$\mathcal{R} = \sum_{i=0}^{n}\sum_{j=0}^{m} \alpha_{ij} x_i \otimes y_j, \quad \sum_{i,j} |\alpha_{ij}|^2 = 1,$$

where $x_1, ..., x_n$ is a computational basis for X and $y_1, ..., y_m$ is a computational basis for Y, then a measurement of $\mathcal{R}$ will be a state $x_i \otimes y_j$ with $\alpha_{i,j} \neq 0$, a measurement of $\mathcal{R}_X$ will be a state x_i with $\alpha_{i,j} \neq 0$ for *some* j, and a measurement of $\mathcal{R}_Y$ will be a state y_j with $\alpha_{i,j} \neq 0$ for *some* i. To manipulate a quantum register, we apply a quantum gate, just like we apply a logic gate to manipulate a classical register. The effect of a quantum gate on a register is given by image of its input state under a unitary transformation U modeling the gate. That is, if $\mathcal{R} = v$, and we apply the gate to $\mathcal{R}$, the resulting state of the register is $\mathcal{R} = Uv$. Recall that unitary transformations preserve the Hermitian inner product, and, therefore, the complex modulus of complex vectors. This means that unitary matrices are compatible with the normalization condition for qubits in a quantum register. In other words, if a complex vector is a valid state for a quantum register (ie. a unit vector), so is its image under a unitary transformation. We often conflate a quantum gate

and its representative unitary operator.

The **phase** of a complex number u with $|u| = 1$ is the real number $\varphi \in [0, 1)$ such that $u = e^{2\pi i\varphi}$. Given such u, the problem of determining φ with high confidence is called **phase estimation**. Quantum computers have an exponential advantage over classical computers in various phase estimation problems, and this is what facilitates Shor's algorithms for breaking various cryptographic schemes.

The mathematical operation at the heart of quantum phase estimation is called the **quantum Fourier transform**. The quantum Fourier transform $\mathcal{F}$ is a linear isomorphism $\mathbb{C}^m \to \mathbb{C}^m$ defined by the rule

$$\mathcal{F}(|j\rangle) = \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} e^{2\pi ijk/m} |k\rangle . \tag{3}$$

In particular, this is a unitary transformation that can be efficiently configured as a quantum gate acting on a quantum register. Since $\mathcal{F}$ is an isomorphism, it has an inverse $\mathcal{F}^{-1}$, which can also be efficiently implemented with a quantum circuit. This fact turns out to be handy for phase estimation. We discuss the mathematical properties of $\mathcal{F}$ and how they facilitate Shor's algorithm through phase estimation, but we do not describe the underlying quantum circuit. Those interested in how one *builds* a quantum circuit that *does* the quantum Fourier transform can consult the excellent treatment in Nielsen and Chaung. We discuss in Section 4 how to embed secp256k1 into a quantum register, so that a quantum phase estimation circuit can be used to solve its discrete logarithm problem.

While the quantum Fourier transform is arguably the "star of the show," we need to discuss two more types of quantum gates before we can begin developing the quantum phase estimation, First, there is the Hadamard gate, which takes

$$|0\rangle \mapsto \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle .$$

Using sequences of Hadamard gates, one can perform the transformation

$$|0\cdots0\rangle \mapsto \frac{1}{\sqrt{2^n}} (|0\rangle + |1\rangle) \otimes \cdots \otimes (|0\rangle + |1\rangle) = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle . \tag{4}$$

Given a register $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2$ and a gate U acting on $\mathcal{R}_2$, one can form an $\mathcal{R}_1$-**controlled** U **gate** acting on $\mathcal{R}$ that, roughly speaking, acts on $\mathcal{R}_2$ according to the respective superposition of states of the constituent qubits of $\mathcal{R}_1$. SAY WHAT THE RESULT IS!!! Quantum phase estimation uses a register $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2$, where $\mathcal{R}_2$ is acted upon by a unitary transformation U with an eigenvalue whose phase is of interest, and $\mathcal{R}_1$ is initially configured to the ground state $\mathcal{R}_1 = |0\cdots0\rangle$. Configuring $\mathcal{R}_2$ with eigenvectors of U, one uses Hadamard gates to put the auxiliary register $\mathcal{R}_1$ in state (4), so that $\mathcal{R}$ is poised for the application of $\mathcal{R}_1$-controlled U^k gates. Applying these controlled U^k gates stores meaningful information about phases of the eigenvalues in the auxiliary register $\mathcal{R}_1$, which can be extracted by applying the inverse quantum Fourier transform to $\mathcal{R}_1$. We may now give some indication of the quantum phase estimation algorithm.

Suppose a phase φ may be written $\varphi = a_1/2 + \cdots + a_t/2^t$ with $a_i \in \{0,1\}$ for each i, and we have a register $\mathcal{R}_2$ configured to the corresponding eigenvector, that is

$$\mathcal{R}_2 = |u\rangle\,,$$

and

$$U\,|u\rangle = e^{2\pi i\varphi}\,|u\rangle\,.$$

If we entangle $\mathcal{R}_2$ with another register $\mathcal{R}_1 = |0\cdots0\rangle$ consisting of t qubits configured to their ground state, then we have a new register

$$\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 = |0\cdots0\rangle\,|u\rangle\,.$$

By putting the qubits in $\mathcal{R}_1$ in superposition using Hadamard gates, then applying a series of controlled powers of U to $\mathcal{R}_2$, one transforms the register $\mathcal{R}$ to the state

$$\mathcal{R} = \frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}|k\rangle\,U^k\,|u\rangle = \frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}e^{2\pi i\varphi k}\,|k\rangle\,|u\rangle = \frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}e^{2\pi i(2^t\varphi)k/2^t}\,|k\rangle\,|u\rangle\,. \tag{5}$$

Please take a moment to compare Equations (3) and (5). Note that $2^t\varphi$ in the right hand side of (5) occupies the position of j in the right hand side of (3). By applying the inverse Fourier transform $\mathcal{F}^{-1}$ to the first register, we put $\mathcal{R}$ in the state

$$\mathcal{R} = |2^t\varphi\rangle\,|u\rangle\,,$$

so that by measuring the first register and dividing the result by 2^t, we recover φ exactly. Now, an arbitrary phase φ does not admit a finite binary representation, but it is a real number that satisfies the inequality $0 \le \varphi < 1$, so we can write

$$\varphi = \sum_{i=1}^{\infty}\frac{a_i}{2^i}$$

for some sequence $(a_i)_{i=1}^{\infty}$ with $a_i \in \{0,1\}$ for each i. We can use the same approach as before to get $\mathcal{R}$ into the state

$$\mathcal{R} = \frac{1}{\sqrt{2^T}}\sum_{k=0}^{2^T-1}e^{2\pi i(2^T\varphi)k/2^T}\,|k\rangle\,|u\rangle\,, \tag{6}$$

for some reasonably large T, but now $2^T\varphi$ is not an integer like j is in Equation (3). Nevertheless, by having enough qubits in $\mathcal{R}_1$ and applying the inverse Fourier transform to the first register of (6), we are able to transform $\mathcal{R}$ to the state

$$\mathcal{R} = |\overline{2^T\varphi}\rangle\,|u\rangle\,,$$

where $\overline{2^T\varphi}$ denotes a T-bit integer whose first t bits agree with the first t bits of $2^T\varphi$ with high probability. We can make the probability higher by increasing T, ie. by increasing the number of qubits in $\mathcal{R}_1$. By measuring the first register, we can recover the first t bits of φ with high probability. In particular, $\left(\overline{2^T\varphi}\right)/2^T$ is a good approximation of φ in the sense that $\left|\left(\overline{2^T\varphi}\right)/2^T - \varphi\right| < 2^{-t}$ with very high probability, so we can write $\varphi \approx \left(\overline{2^T\varphi}\right)/2^T$.

Hopefully, the reader is convinced that the quantum Fourier transform is a promising tool for phase estimation. The fact that the Fourier transform can be efficiently accomplished by a quantum gate ultimately allows quantum computers to do efficient phase estimation. I have, however, left one detail out of the discussion above: in practice, the eigenvector $|u\rangle$ often cannot be configured

within a register. However, one is able to get around this by taking certain averages of eigenvectors, because these averages turn out to be configurable. The quantum Fourier transform in turn is robust enough to extract meaningful information about the phase from these averages. We describe this carefully for secp256k1 in Section 5.

Remark 3.1. In Section 5, we will need to estimate the phase of a complex eigenvalue $\lambda = e^{2\pi i\psi}$, where $\psi \geq 1$. In this case, it is important to keep in mind that $e^{2\pi i\psi} = e^{2\pi i\{\psi\}}$, where $\{\psi\}$ is the fractional part of ψ given by

$$\{\psi\} = \psi - \lfloor \psi \rfloor,$$

where $\lfloor \psi \rfloor$ is the floor function rounding a real number down to the nearest integer. Indeed, the phase of λ is $\{\psi\}$, and

$$\mathcal{F}^{-1}\left(\frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} e^{2\pi i\psi k} |k\rangle |u\rangle\right) = |\overline{2^t\{\psi\}}\rangle\,.$$

In other words, for any real number $\psi \geq 1$, quantum phase estimation for $\lambda = e^{2\pi i\psi}$ allows us to estimate the fractional part of ψ. As before, larger choices of t result in better estimates for $\{\psi\}$. Note for positive integers $s > \ell$, we know that

$$\{s/\ell\} = s'/\ell$$

for $0 \leq s' < \ell$ such that $s \equiv s' \bmod \ell$, by the Euclidean algorithm. This observation will become important later when we estimate phases arising from integer ratios modulo the order of secp256k1.

4. Encoding a Group Law in a Computational Register

To actually implement Shor's algorithm for secp256k1 on a quantum computer, we need to encode its group law within a quantum register. The idea is to identify group elements with base states in the register, so that the quantum gate corresponding to addition by a group element simply permutes the basis states. Note that permuting basis states amounts to doing bit flips on their binary labels. In other words, for a finite abelian group A, we need to embed the regular representation

$$V = \bigoplus_{a \in A} \mathbb{C}a$$

within a quantum register $\mathcal{R}$, so that the action of $a \in A$ on V can be realized by a quantum gate U_a acting on $\mathcal{R}$. To do this in practice, we need an *efficient* oracle that computes the product $a \oplus b$ when given $a, b \in A$, and we need an *efficiently computable* injective function

$$|\cdot\rangle : A \to \mathcal{R} : a \mapsto |a\rangle$$

from A to the basis states of the quantum register. This is because the register is a priori blind to the group law. In particular, if our register $\mathcal{R}$ is configured to the state $|b\rangle$, we determine the result of applying the U_a gate to $\mathcal{R}$ by first querying the oracle for the sum $a \oplus b \in A$, then computing its corresponding basis state $|a \oplus b\rangle$. From this, we can conclude $U_a |b\rangle = |a \oplus b\rangle$ and configure the gate U_a accordingly.

If our group A is the secp256k1 curve, the existence of the efficient oracle is a consequence of the group law for elliptic curves, which we describe in Section 5. To build $|\cdot\rangle$, we can use the fact that any point of secp256k1, besides its identity element 0, is uniquely represented by an ordered pair in the Cartesian product $\mathbb{F}_p \times \mathbb{F}_p$. For the purpose of finding $|\cdot\rangle$, we can think of $\mathbb{F}_p \times \mathbb{F}_p$ as a $p \times p$ grid of p^2 points. By numbering the points on the grid, one obtains an injective map $\mathbb{F}_p \times \mathbb{F}_p \to \mathbb{Z}/p^2\mathbb{Z}$.

By choosing the numbering carefully, one can cook up an *efficiently computable* map. For instance, we can visualize the grid as the set of points $\{(i,j)\}_{i,j=0}^{p-1}$ in the plane $\mathbb{R}^2$. We partition the grid by its intersections with lines of slope -1. Tracing these lines downward from the left until tracing the main diagonal, then tracing from the right upward until running out of points will yield a labeling that looks something like

$$\iota : \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{Z}/p^2\mathbb{Z} : (x,y) \mapsto \begin{cases} n_{x+y} + x, & x+y \leq p-1 \\ n_p + n_{2p-2-x-y} + p - 1 - x, & x+y > p-1 \end{cases}$$

where $n_i = i(i+1)/2$ is the i^{th} triangular number. The function ι can be easily computed on a classical computer. Here is a chart that illustrates the idea when $p = 5$.

y					
4	10	24	20	17	15
3	6	11	23	19	16
2	3	7	12	22	18
1	1	4	8	13	21
0	0	2	5	9	14
	0	1	2	3	4
			x		

We can therefore obtain an efficiently computable embedding $|\cdot\rangle$ of secp256k1 into a computational register by sending a point (x,y) of secp256k1 to its image under ι, then sending $\iota(x,y)$ to the basis state corresponding to the binary representation of $\iota(x,y)$. That is

$$|\cdot\rangle : \text{secp256k1} - \{0\} \to \mathcal{R} : (x,y) \mapsto |\iota(x,y)\rangle\,,$$

so if $P = (x,y)$ and $P \neq 0$, we set $|P\rangle := |\iota(x,y)\rangle$, and we extend $|\cdot\rangle$ to all of secp256k1 by setting $|0\rangle := |0\cdots0\rangle$.[1]

5. secp256k1 and Shor

Let's review the definition of the secp256k1 curve, along with the properties that have made it suitable for digital signature algorithms. This curve, which we will denote by E, consists of all pairs (x,y) with $x, y \in \mathbb{F}_p$ satisfying the elliptic equation

$$y^2 = x^3 + 7,$$

where $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ is the size of the base field $\mathbb{F}_p$, along with the identity element 0, which is often called the "point at infinity." When we say we are "given a (nonidentity) point $P \in E$," we mean we are given such a pair of coordinates (x,y), and $P = (x,y)$. We say in this case that (x,y) are the **coordinates** of P. It turns out that E is a cyclic group of (extremely)

[1]The point $(0,0)$ is not on secp256k1, since 7 is not congruent to 0 modulo p.

large prime order ℓ. Specifically, there is a known generating point G of order ℓ, which is expressed in hexadecimal as

$$\ell = FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE$$
$$BAAEDCE6\ AF48A03B\ BFD25E8C\ D0364141.$$

This is on the order of magnitude of 2^{256}, which is only about a thousand times less than the estimated number of atoms in the entire universe. This means E is huge, and every point $P \in E$ is of the form $P = mG$ for some non-negative integer m. Here, mG means G added to itself $m-1$ times according to the group law, $\oplus$. For example, $2G = G \oplus G$, and $5G = G \oplus G \oplus G \oplus G \oplus G$. The group law for E is as follows:

The sum $(x_1, y_1) \oplus (x_2, y_2)$ of $(x_1, y_1), (x_2, y_2) \in E$ is equal to

$$\begin{cases} 0, & \text{if } x_2 = x_1 \text{ and } y_2 = -y_1 \\ \left(-2x + \dfrac{9x^4}{2y}, -y + \left(x + 2x - \dfrac{9x^4}{2y}\right)\dfrac{3x^2}{2y}\right), & \text{if } x_1 = x_2 = x \text{ and } y_2 = y_1 = y, \\ \left(-x_1 - x_2 + \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2, -y_1 + \left(2x_1 + x_2 - \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2\right)\dfrac{y_2 - y_1}{x_2 - x_1}\right), & \text{otherwise.} \end{cases}$$

Note that knowledge of x_1, x_2, y_1, y_2 allows one to efficiently compute $(x_1, y_1) \oplus (x_2, y_2)$. This facilitates efficient signing and verification for ECDSA, because both involve adding points on an elliptic curve according to its group law. On the other hand, the algebraic complexity of the expressions in the group law, along with the enormity of E, also serves a cyptographic purpose: it scrambles the coordinates, so that even if we know the coordinates (x, y) of a point P in E, it is computationally infeasible to determine m such that $P = mG$ using classical algorithms. Put simply, we cannot efficiently divide an element of E by G. At least, despite tremendous effort, no one has figured out how to do so. In fact, to recover m from knowledge of the coordinates of $P = mG$, even the most efficient known classical algorithms (eg. Pollard's ρ) achieve only a quadratic speedup over the naive approach of iteratively adding $-G$ until getting back to G, which is an intractable procedure that requires an amount of time comparable to ℓ when m is large and random (as is the case in ECDSA). In summary, the coordinates of the points in E appear to be pairs of random elements in $\mathbb{F}_p$, with no clear relation to the coordinates of G, and determining their relations by brute force is infeasible with classical computation, even for the world hegemon. The reader who wants to feel the truth of this should ponder for a moment just how massive ℓ is.

Now, suppose we are given a point $P \in E$, and we are given the apparently daunting task of determining m such that $P = mG$. This is the **discrete logarithm problem** for P with base G. We already know from the preceding discussion that determining m by inspecting the coordinates of P is infeasible, so we will try to do something that feels more sophisticated by appealing to the representation theory of the cyclic group E. We saw in Section 2 that if U is the unitary transformation

$$U\,|jG\rangle = |-P + jG\rangle\,,$$

then the vector

$$v_k := \frac{1}{\sqrt{\ell}} \sum_{r=0}^{\ell-1} e^{2\pi i r k/\ell}\,|rG\rangle \tag{7}$$

is an eigenvector of U with eigenvalue $e^{2\pi imk/\ell}$. Since $Uv_k = e^{2\pi imk/\ell}v_k$, we could determine the eigenvalue by comparing the entries of v_k and Uv_k, assuming we could efficiently configure U and v_k in a computational register to perform the matrix multiplication. From there, since we know ℓ, k, and $e^{2\pi imk/\ell}$, we could find m with some arithmetic mod ℓ. The issue is, we cannot configure v_k. Indeed, the sum in Equation (7) involves a number of terms that is just a few orders of magnitude less than the total number of atoms estimated to be in the entire universe, and each term would have to be correctly configured in order to get v_k in the register. This is infeasible, so we cannot hope to extract m using a single eigenvector in a computational register, regardless of whether that register is classical or quantum. What we will try instead is a standard procedure in Fourier analysis: since we can't extract information about m from a *specific* v_k, we will try to extract information from the *average of all* v_k.

The initial calculation is promising. Indeed, by Equation (2) in Section 2, we know

$$\frac{1}{\sqrt{\ell}}\sum_{k=0}^{\ell-1} v_k = |0\rangle\,,$$

so the average is just $|0\rangle$: a basis state in our register. On the other hand, the v_k are exceedingly large linear combinations of basis states. Therefore, we can easily configure the average, whereas configuring v_k is impossible for practical purposes like signature forgery. Now, if we entangle a register $\mathcal{R}_2$ configured to $|0\rangle$ with a register $\mathcal{R}_1$ consisting of t qubits configured to the ground state, we will have a register

$$\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 = |0\cdots 0\rangle\,|0\rangle\,.$$

The idea now is to manipulate $|0\rangle$ with $\mathcal{R}_1$-controlled powers of U, so that the eigenvalues $e^{2\pi imk/\ell}$ are recorded in the first register. As in Section 3, by running $\mathcal{R}_1$ through Hadamard gates, then running R_1-controlled powers of U-gates on $\mathcal{R}_2$, we develop a gate that transforms $\mathcal{R}$ to the state

$$\mathcal{R} = \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1} |j\rangle\,U^j\,|0G\rangle\,.$$

Take a moment and reflect on the process so far. The initial configuration of our register $\mathcal{R}$ consisted of two entangled registers $\mathcal{R}_1$ and $\mathcal{R}_2$, initialized to the basis states $|0\cdots 0\rangle$ and $|0\rangle$, respectively. Basis states are always configurable in a physical quantum register, and we actually *know* what $|0\rangle$ *is*, assuming we have embedded the group law of secp256k1 into $\mathcal{R}_2$ reasonably efficiently, as in Section 4. Therefore, $\mathcal{R} = |0\cdots 0\rangle\,|0\rangle$ is a state we can *actually configure* within a reasonably robust quantum computer. Furthermore, the Hadamard gates and $\mathcal{R}_1$-controlled gates can also *really be configured* within a quantum circuit. Making precise sense of realizing the effect of the controlled powers of U-gates still requires the embedding from Section 4, but the upshot is that one can *actually configure*[2] the circuit we are describing, and it will run in an amount of time that makes an attack on secp256k1 feasible. We will now pick up where we left off in the last paragraph.

Let us investigate the state of our register a bit deeper by calculating

$$\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1} |j\rangle\,U^j\,|0\rangle = \frac{1}{\sqrt{2^t\ell}}\sum_{k=0}^{\ell-1}\sum_{j=0}^{2^t-1} |j\rangle\,U^j v_k = \frac{1}{\sqrt{2^t\ell}}\sum_{k=0}^{\ell-1}\left(\sum_{j=0}^{2^t-1} e^{2\pi ijmk/\ell}\,|j\rangle\right)v_k.$$

[2]Modulo the incredible engineering task of developing cryptographically relevant quantum computers.

We put parenthesis around the contribution of the first register to the last expression, in order to highlight the fact that information about m is encoded in the phases of the eigenvalues that have been recorded there, and this contribution looks a lot like it has come from a Fourier transform (cf. Equation (3)). In fact, if we apply an inverse Fourier transform to $\mathcal{R}_1$, the resulting state of our register $\mathcal{R}$ is

$$\mathcal{R} = \frac{1}{\sqrt{\ell}} \sum_{k=0}^{\ell-1} |\overline{2^t\{mk/\ell\}}\rangle\, v_k,$$

so that measuring the first register gives $\overline{2^t\{mk/\ell\}}$, from which we deduce an approximation $\{mk/\ell\} \approx \overline{2^t\{mk/\ell\}}/2^t$ for *some* k (cf. Remark 3.1). However, there is an issue...we don't know *which* k, so we can't say anything meaningful about m.

As it turns out, we can address this without too much additional effort. If instead of U, in the discussion above, we had used the linear transformation V defined by the rule $V|mG\rangle = |(m-1)G\rangle$, the result of measuring $\mathcal{R}_1$ after running our quantum circuit would have been $\overline{2^t k/\ell}$, from which we derive an approximation $k/\ell \approx \overline{2^t k/\ell}/2^t$ for *some* k. Now, if we could run the circuits for U and V and guarantee the result gave estimates of $\{mk/\ell\}$ and k/ℓ for the *same* k, then we could obtain a good estimate of m. Indeed, in this case we determine k by multiplying our estimate of k/ℓ by ℓ, then we find m by dividing m' by k mod ℓ, where $m' \equiv mk \bmod \ell$. Note that we *know* ℓ, so there is no need to appeal to the continued fractions algorithm used in Shor's order-finding algorithm for RSA. Thankfully, we can configure a circuit that estimates phases for the eigenvalues of U and V in tandem with one-another, and measuring its auxiliary register gives exactly what we need. The idea is essentially to have a large auxiliary register $\mathcal{R}'$ split into two parts $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2$. One then captures eigenvalue data for V, U in $\mathcal{R}_1, \mathcal{R}_2$, respectively, by implementing the circuit described above with $\mathcal{R}_1$-controlled powers of V-gates and $\mathcal{R}_2$-controlled powers of U-gates run in tandem. We conclude this section by sketching the details.

Configure registers $\mathcal{R}_1 = |0\cdots0\rangle$ and $\mathcal{R}_2 = |0\cdots0\rangle$ consisting of t qubits each, along with a third register $\mathcal{R}_3 = |0\rangle$. Entangling these registers, we initialize a register

$$\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3 = |0\cdots0\rangle\,|0\cdots0\rangle\,|0\rangle\,.$$

We use the auxiliary registers $\mathcal{R}_1$ and $\mathcal{R}_2$ to record the eigenvalues of V and U, respectively. We choose t as large as we need to estimate the phases to our desired precision (cf. Section 3). The auxiliary registers are put in superposition using Hadamard gates as before, then we run $\mathcal{R}$ through a sequence of $\mathcal{R}_1$-controlled powers of V-gates and a sequence of $\mathcal{R}_2$-controlled powers of U-gates, each acting on $\mathcal{R}_3$. The result is

$$\mathcal{R} = \frac{1}{2^t\sqrt{\ell}} \sum_{k=0}^{\ell-1} \sum_{r=0}^{2^t-1} \sum_{j=0}^{2^t-1} |r\rangle\,|j\rangle\, V^r U^j v_k$$

$$= \frac{1}{2^t\sqrt{\ell}} \sum_{k=0}^{\ell-1} \sum_{r=0}^{2^t-1} \sum_{j=0}^{2^t-1} |r\rangle\,|j\rangle\, e^{2\pi i k(r+mj)/\ell} v_k$$

$$= \frac{1}{2^t\sqrt{\ell}} \sum_{k=0}^{\ell-1} \left(\sum_{r=0}^{2^t-1} e^{2\pi i k r/\ell}\,|r\rangle \right) \left(\sum_{j=0}^{2^t-1} e^{2\pi i k m j/\ell}\,|j\rangle \right) v_k.$$

By applying an inverse Fourier transform to each of the first two registers, we put $\mathcal{R}$ in state

$$\mathcal{R} = \frac{1}{\sqrt{\ell}} \sum_{k=0}^{\ell-1} |\overline{2^t k/\ell}\rangle \, |\overline{2^t \{km/\ell\}}\rangle \, v_k,$$

so measuring the first two registers gives

$$\mathcal{R}_1 = |\overline{2^t k/\ell}\rangle$$

and

$$\mathcal{R}_2 = |\overline{2^t \{mk/\ell\}}\rangle \,,$$

from which we derive estimates $k/\ell \approx \overline{2^t k/\ell}/2^t$ and $\{mk/\ell\} \approx \overline{2^t \{mk/\ell\}}/2^t$ for the *same* k. From here, we can find m as suggested in the previous paragraph.

6. Toward Quantum Safe Bitcoin Protocol

Known quantum attacks against the Bitcoin network would require accessing the public key that unlocks a UTXO. Public keys are exposed via the input or output scripts of a transaction broadcast. "On spend" attacks involve extracting a spender's public key from the unlocking script in a transaction broadcast, running Shor's algorithm to find the spender's private key, and then broadcasting an altered transaction with a forged signature–perhaps with a different recipient than the original–before the original transaction is confirmed and incorporated into the blockchain. Even after a legitimate transaction has been confirmed, and the UTXOs have been updated, those UTXOs whose locking scripts contain the owner's public key are still susceptible to "at rest" attacks, which simply extract the public key (a.k.a. $P = mG$ from Section 5) from the UTXO sheet, then run Shor's algorithm for the discrete logarithm problem on secp256k1 to derive the private key (a.k.a. m from Section 5). The quantum attacker is then free to spend the UTXO as they wish. The idea is that on spend attacks extract public keys from input scripts that are not available in the UTXO sheet, and therefore must be completed in a relatively small amount of time, whereas at rest attacks simply extract public keys from relatively static UTXO data, and can therefore operate over longer time frames.

We discuss these vulnerabilities concretely with two well-known transaction types: P2PK and P2PKH. Let's review their locking and unlocking scripts.

P2PK Unlocking Script	**P2PK Locking Script**
OP_PUSH Signature	OP_PUSH OP_CHECKSIG Public Key

P2PKH Unlocking Script	**P2PKH Locking Script**
	OP_DUP
OP_PUSH	OP_HASH
Signature	OP_PUSH
OP_PUSH	Public Key Hash
Public Key	OP_EQUALVERIFY
	OP_CHECKSIG

Now, P2PKH transactions are immune to at rest attacks, because the locking script of the output

UTXO does not contain the owner's public key, but rather the hash of the owner's public key: quantum computers do not break preimage resistance in cryptographic hash functions like SHA-256, so hashing keeps the public key hidden from a quantum attacker. However, a P2PKH transaction broadcast reveals the spender's public key in the unlocking script, so P2PKH is vulnerable to on spend attacks. On the other hand, P2PK transactions are immune to on spend attacks, because the spender does not reveal their public key in the unlocking script. However, the locking script reveals the owner's public key, so P2PK transactions are vulnerable to at rest attacks. Notice there is a sense in which P2PKH scripts are more quantum secure than P2PK scripts, because the former has less exposure to a quantum attacker than the latter.

Levy recently developed a transaction protocol, QSB, that eliminates all known quantum vulnerability in transaction scripts. Roughly speaking, this is accomplished by replacing public keys and elliptic curve signatures with hash-to-sig puzzles and Lamport signatures, both of which are quantum safe. Its elegance is not in hiding ECC public keys, but rather in making knowledge of them irrelevant for proving ownership. Instead, QSB uses Lamport signatures to prove ownership, as Lamport public keys are (quantum safe) hashes of private key data. Miraculously, this protocol does not require consensus change. Indeed, there is now a QSB transaction on the blockchain. The main drawback of QSB is the relatively high cost (estimates are around $100 per transaction) of off-chain computation needed to solve the hash-to-sig puzzles, which is incurred by honest spenders of UTXOs with QSB locking scripts. However, QSB appears to be a promising option for those with significant bitcoin holdings who wish to proactively shield themselves from a quantum threat. For instance, let's say we own a UTXO with a standard locking script–P2PKH, say–and we want to move it to a quantum safe address. All we have to do is broadcast a transaction whose input script is the P2PKH unlocking script that proves our ownership of the UTXO and whose output script is a QSB locking script. After confirmation, the resulting UTXO is quantum safe, and now if we want to spend it, we will need to spend about $100 to solve the hash-to-sig puzzles involved in finding a valid input script to unlock the UTXO.

QSB seems promising for proactive Bitcoiners who are willing to migrate their holdings to post-quantum addresses before Q-day, but what recourse does the network have for those addresses that are not quantum safe upon Q-day's arrival? Won't quantum attackers quickly pillage the exposed portion of the UTXO sheet? With a softfork, the Bitcoin network can update consensus rules to require additional proof of ownership beyond ECC signatures in order to spend UTXOs that do not have quantum safe locking and unlocking scripts. The right softfork could give honest spenders the ability to move such UTXOs to quantum safe addresses, while at the same time preventing on-spend and at-rest attacks by a quantum computer. A softfork like this is called a **rescue protocol**. The idea is to require the spender of a UTXO to supply information about it, which is inaccessible to a quantum attacker. Examples include the public key whose hash is stored in a P2PKH locking script, or the seed (xpub or xpriv) used to generate (through hardened derivation) a key pair whose public key is stored in a P2PK locking script. Information about a UTXO that is not accessible to a quantum attacker is known as a **knowledge asymmetry**. All rescue protocols rely on knowledge asymmetry, and there are several proposals already in existence. For instance, DropKick is a recently announced rescue protocol, according to which an honest "quantum procrastinator" would rescue their UTXO by depositing the hash of their knowledge asymmetry and a quantum safe public key Q on the block chain (this is the **commitment**), waiting ten days, then broadcasting a standard signed transaction, along with some data binding the transaction to

the quantum safe commitment with a quantum safe signature, verifiable with Q. In doing so, the spender is able to prove the UTXO is theirs and update it to a quantum safe address, all while remaining consistent with legacy consensus rules and protecting their UTXO from quantum threats.

Exciting progress is being made on developing quantum safe Bitcoin protocols, but there is still work to be done. For instance, a successful rescue protocol will require a softfork with consensus. Furthermore, most quantum safe Bitcoin protocol proposals rely on hash functions, which are arguably the most trusted quantum-hard one-way functions. However, there are other such functions. Indeed, post quantum cryptography is a blossoming field in its own right, and it will be exciting to see how ideas from this discipline influence the long term development of Bitcoin protocol.

Email address: shaverphagan@gmail.com