TECHNICAL NOTE MRL-00XX.0

Evaluating cryptocurrency security and privacy in a post-quantum world

Adam Corbo*, Mitchell "Isthmus" Krawiec-Thayer†, Brandon G Goodell‡
Insight & Monero Research Lab
September 2020

Abstract

We audit some security properties of the Monero protocol in the context of the capabilities and algorithms of quantum computers, including Shor's algorithm, Grover's algorithm, Simon's algorithm, quantum differential cryptanalysis, and quantum access to random oracles. The details of several theoretical vulnerabilities are described: extracting private key from public keys, extracting keys from one-time addresses, ascertaining the true input to ring signatures, violating transaction balancing, unmasking transaction amounts, linking multiple transactions to the same address, decrypting payment identifiers, and hash collisions. We discuss several post-quantum canddiate settings: lattice-based cryptography, multivariate-based cryptography, hash-based cryptography, and supersingular elliptic curve isogeny-based cryptography. An overview of alternative protocols discusses ZK-STARKs, MatRiCT, Raptor-512, and RingRainbow. We examine general tradeoffs and compare candidate protocols based on signature/proof size, public key size, generation time, and verification time. For curious readers, we include toy python implementations of Shor's algorithm, Grover's algorithm and Simon's algorithm, along with our results from commercial quantum hardware; see:
`https://github.com/insight-decentralized-consensus-lab/post-quantum-monero`

1 Introduction

Today's cryptocurrencies, such as Bitcoin, Ethereum, Monero, and Zcash, base their security on computationally intractable problems. This includes a seemingly ubiquitous dependence on the hardness of the discrete logarithm problem, with applications dating at least back to [1]. All of the cryptocurrency protocols listed at least partially rely on the intractability of the discrete logarithm problem as the basis of their security against classical computers (see [2], [3], [4], and [5]). Quantum algorithms exist that can solve this problem in polynomial time given optimally efficient quantum hardware. These algorithms technically can be run on a classical computer, but are inefficient without such hardware access.

We audit the security of blockchain protocols in the context of several quantum algorithms, using Monero as a case study. The noted security vulnerabilities such as key generation are broadly applicable to all elliptic curve cryptography-based coins, and advertising these outcomes as Monero-specific would be a misrepresentation of the scope of the results.

The theory surrounding quantum algorithms and methods has been progressively more developed over the last few decades. We feel that a formal analysis of the security impacts on current protocols and a discussion on long-term migration away from protocols insecure against a quantum-capable attacker are both overdue. This technical note essentially cannot be rigorously exhaustive. More quantum algorithms and abilities are bound to be discovered and other methods may exist that have not been made public.

Cryptographers have developed protocols for use by classical computers that are both resistant to a quantum adversary and may be adaptable to Monero. Each approach has its own benefits, drawbacks, and space/time complexity. We describe and compare some of these in Section 5.

*adamryancorbo@gmail.com
†Isthmus@getmonero.org or IsthmusCrypto@protonmail.com
‡surae@getmonero.org

Speculation on whether practical quantum computers will ever exist, and when they might arrive, is outside of the scope of this technical note. To our knowledge, there are currently no plausibly post-quantum anonymous cryptocurrencies in use today, meaning that only short-to-intermediate term financial privacy is available with current technology. The first protocol to implement privacy features resistant to a quantum adversary will be in a strong position for adoption, even in a pre-quantum world.

1.1 Prerequisites, Preliminaries, and Terminology

We denote concatenation of strings with $||$. For a finite unordered set X with $|X| = n$, note that we can arbitrarily label entries in X and presume $X = [n] = \{1, 2, \ldots, n\}$ without losing any generality. We denote cryptographic hash functions with $\mathcal{H}$.

We say two transactions are *unlinkable* if it is difficult for a non-recipient to discern whether the two transactions have the same recipient or not. We say transactions are *signer-ambiguous* if it is difficult for a non-sender to discern the sender of the transaction. We say transactions are *confidential* if it is difficult for a non-sender, non-recipient to discern the transaction amounts.

When we consider a *classical adversary*, which we abbreviate CA, we mean an adversary only with access to probabilistic Turing machines. When we consider a *quantum adversary*, which we abbreviate QA, we mean an adversary with access to any device(s) that can execute quantum algorithms efficiently and with a sufficient degree of scalability to concretely threaten the security of the cryptographic primitives underlying the Monero protocol. This is not defined by some magic number of qubits or any particular configuration; rather it refers to the capability to leverage methods such as Shor's algorithm (see [6]), Grover's algorithm (see [7]), and Simon's algorithm (see [8]) with sufficient efficiency to tackle the Monero protocol.

In all cases we assume the QA or CA has access to public blockchain data. In some sections, we will note that the attack requires an adversary to have an incomplete database of addresses. These could have been obtained through leaks, OSINT collection, collusion with exchanges/merchants, or any other source.

2 Capabilities of a Quantum Adversary

In this section, we present a brief and informal description of quantum computation, together with a few capabilities of the QA.

A classical bit is a binary digit that takes on one of two values to indicate a logical value. By convention, these are 0 and 1. Qubits, on the other hand, represent many logical values that reduces to a classical bit whenever the qubit interacts with the environment. Indeed, to measure the state of a qubit is equivalent to collapsing it into a classical state. Before measurement, a qubit remains in a superposition of the values 0 and 1. States of qubits are elements of $\mathbb{C}^2$, and we denote orthonormal basis vectors $|1\rangle$ and $|0\rangle$. The superposition $|\psi\rangle$ of a qubit is represented as a linear combination of these basis vectors $|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle$ where $a_0 \in \mathbb{C}$ is the complex scalar amplitude of the state along the direction of $|0\rangle$ in $\mathbb{C}^2$, and a_1 is the amplitude along $|1\rangle$ in $\mathbb{C}^2$.

Amplitudes may be thought of as "quantum probabilities:" the amplitude along an orthonormal basis vector is related to the probability that the collapsed state corresponds to that vector. That is to say, the amplitude along $|0\rangle$ is related to the probability that the collapsed state is 0, and the amplitude along $|1\rangle$ is related to the probability that the collapsed state is 1. We must take care, however. Amplitudes are represented by complex numbers, while traditional probabilities are described by real numbers. Just as the probabilities of a classical system must integrate to 1 under some probability measure μ in order to form a distribution function, the squared magnitudes of state amplitudes in a quantum system must satisfy $\int |a_0|^2 + |a_1|^2 \, d\mu = 1$.

2.1 Violate Discrete Logarithm Hardness with Shor's Algorithm

Let G be an elliptic curve group over some field $\mathbb{F}$. We say the discrete logarithm problem is computationally intractable in G if the following assumption holds.

Assumption 2.1 (Discrete Logarithms in Elliptic Curve Groups)**.** *There does not exist an algorithm $\mathcal{A}$ that takes as input a pair $(g, h) \in G^2$ sampled uniformly at random, runs in polynomial time t, and, with non-negligible probability ϵ, outputs some $x \in \mathbb{F}$ such that $g^x = h$ or $g = h^x$.*

Shor's algorithm, when executed on a quantum computer, can be used with some non-negligible probability ϵ in time t that is polynomial in $|G|$ to solve a related problem, the *hidden subgroup problem*. For a finite abelian group G, a subgroup $H \subseteq G$, a finite set X, and a function $f : G \to X$, we say f *hides* H if, for any $g_1, g_2 \in G$, $f(g_1) = f(g_2)$ if and only if $g_1 H = g_2 H \in G/H$. For example, setting X as the quotient group $X = G/H$, we see that the canonical group epimorphism $f : G \to G/H$ hides H. We can always describe f with $O(\log |G| + \log |X|)$ bits.

It is easy to describe how to break discrete logarithms by breaking the hidden subgroup problem. If $G' = \langle g \rangle$ is any group of order p, we can compute the discrete logarithm of an ostensibly random $h = g^x$ with unknown x using $G = \mathbb{F} \times \mathbb{F}$, $H = \langle (-x, 1) \rangle$, and the function $f : G \to \mathbb{Z}_p$ defined by mapping $(a, b) \mapsto g^a h^b$. This f is a group homomorphism whose kernel is H; finding H is equivalent to computing the discrete logarithm for the generator $X = g^x$. A more in-depth technical outline for running Shor's algorithm for breaking the discrete logarithm hardness assumption is outlined in Section A.1. For more details, see [9] and [10].

It is still thought that there does not exist a PPT algorithm that violates Assumption 2.1. An asymptotic lower bound on classical algorithms solving the discrete logarithm problem is $(\frac{2}{3} + o(1))\sqrt{|G|}$ operations, derived in [11], whereas the average case instance of Shor's algorithm takes $O((\log |G|)^3)$ operations, which is exponentially faster. For a group G such that $|G| \geq 2^{256}$, the CA requires at least 2^{127} operations, which is concretely intractable. On the other hand, there exists some constant c such that the QA requires $c \cdot 2^{24}$ operations, up to lower order terms.

2.2 Unstructured Search with Grover's Algorithm and Hash Pre-images

Some aspects of security rely on the fact that finding the pre-image of a hash digest is difficult. Grover's algorithm can be applied to compute hash digest pre-images: this algorithm finds unordered database entries that satisfy search criteria in $O(\sqrt{n})$ time, where n is the database size. See [7] and Appendix A.2 for details. Grover's algorithm is asymptotically optimal even for quantum adversaries, as shown in [12]. For a QA employing Grover's algorithm, it is possible to find a marked value in an un-ordered set of data. Although this is only a quadratic speedup, it is asymptotically optimal, in that any quantum algorithm takes $\Omega(\sqrt{n})$ time to solve this problem (see [12]). One possible mitigation for this is to double the message lengths used in the hash function, which makes this more difficult to accomplish.

Let $f : [n] \to \{0, 1\}$ be a function which describes whether an index matches the search criteria and to which Grover's algorithm has oracle access. For any non-negative function $f : [n] \to \mathbb{Z}^+$ such that $\sum_{x \in [n]} f(x) > 0$, note that non-negativity implies there must exist some $w \in [n]$ such that $f(w) > 0$. Moreover, the codomain of f is $\{0, 1\}$ so $f(w) = 1$. Grover's algorithm makes approximately $O(\sqrt{n})$ queries to f via oracle access and outputs a solution w such that $f(w) = 1$. Given a hash function $\mathcal{H}$, we can define $f(x)$ to be 1 when $\mathcal{H}(x) = y$ and 0 otherwise. We can then use Grover's algorithm to find hash digest pre-images that fit the necessary parameters.

In many cases, the universe of possible pre-images is stored on the blockchain in public, and so applying Grover's algorithm in these cases in $O(\sqrt{n})$ time can be quite fast. Grover's can still be applied to find arbitrary pre-images that have not yet necessarily been posted publicly, but the search space is significantly larger. In these cases, $O(\sqrt{n})$ is often still a prohibitive hurdle.

For a technical description of Grover's algorithm, see Section A.2. Some bounds on Grover's algorithm can be found in [13], where it is shown that the expected number of queries (one per iteration) to the database Grover's must make is at least $\lfloor \sin(\pi/8)\sqrt{n} \rfloor$. To brute force a 256 bit collision resistant hash digest, essentially trying every

possible input pre-image to a hash function until the known hash digest gets spit out, on a classical computer would take an impractical 2^{255} operations on average. This kind of attack, with the message space of about 2^{256} is highly impractical to attempt and as such is considered one way. On the other hand, a pre-image attack can be done by the QA with Grover's.

For a 256-bit hash digest, a quantum computer applying Grover's algorithm requires at least $\lfloor \sin(\pi/8)2^{128} \rfloor \approx 2^{126.6}$ queries to the database, one for each iteration. Even if each iteration of Grover's algorithm can be done in constant time in at most one picosecond, which is a laughably generous assumption for us to grant the QA, searching a database with 2^{256} entries should take around 10^{26} seconds. The universe is thought to be around 10^{17} seconds old, so this is more than concretely impractical. On the other hand, searching a database with 2^{64} entries requires less than one billion iterations, and a database with 2^{32} entries requires around $25,000$ iterations.

Grover's algorithm is obviously more effective when used to go through a small database instead of searching a cryptographically large space. Indeed, reducing the number of database entries by half reduces the total search time by 75%. If the search space cannot be reduced by much, this use often takes longer than it takes Shor's to break ECC, RSA, etc.

Grover's algorithm can also be used in conjunction with other methods to find the pre-image of hash digests, such as quantum differential cryptanalysis (see Section 2.3).

Monero utilizes the Keccak hash function, which is described as post-quantum secure by [14] and others. However, [15] disputes this assessment, and asserts the existence of attacks on Keccak with greater impact than implied by the purported security of the function. For the sake of thoroughness, our descriptions of Monero's vulnerabilities includes the attack surface that would arise from a quantum-compromised Keccak (even though we hope that it is post-quantum secure).

2.3 Simon's Algorithm, Quantum Differential Cryptanalysis, and Further Capabilities

Simon's algorithm, from [8], can be applied to extract XOR masks from functions under which they are invariant. Since many hash functions are based on iterated XOR masks, this allows for differential cryptanalysis.

Let $f : \{0,1\}^n \to \{0,1\}^n$ be any function that is invariant under some mask a. That is to say, there exists some $a \in \{0,1\}^n$ such that, for every $x, y \in \{0,1\}^n$, $f(x) = f(y)$ if and only if $x \oplus y \in \{0, a\}$. Given oracle access to f, Simon's algorithm makes $O(n)$ queries to the oracle and produces as output the mask a. Like Grover's, this is optimal in the sense that any quantum algorithm must make $\Omega(n)$ queries. This is also exponentially faster than any classical algorithm, which must make $\Omega(2^{n/2})$ queries. See [16] for details.

A tight bound for the probability of success as a function of the number of queries made to the oracle f is given in [17], in which the following heuristic is proposed: Simon's algorithm succeeds in the average case after $n + 3$ queries made to f, and the success probability after $n + k$ queries is roughly $1 - 2^{-k}$.

For a technical description of Simon's algorithm, see Section A.3.

2.3.1 Quantum Differential Cryptanalysis

Besides brute force attacks, there are some other methods a QA or a CA could leverage to reverse or reveal hidden information about the pre-image of a hash digest using differential cryptanalysis techniques such as those first reported in [18], and these techniques may be aided with a quantum computer. Using Simon's algorithm, it's possible to create a system of linear equations that can be used to perform differential cryptanalysis to decrypt an XOR mask, attack symmetric key primitives as in [19], or attack key schedules as in [18]. See [20] for a discussion on the design of block ciphers resistant to differential cryptanalysis.

It is thought that the sponge construction used to create Keccak, the hash function used in Monero, is resistant to a quantum adversary; see [14] for details. It is subjectively unlikely that finding sponge pre-images will be made faster than via application of Grover's algorithm, at least up to lower order terms, except in the case of bad choices of the generating random transformation. Note that side-channel attacks have been presented, e.g. in [21] and [22], with some mitigations presented in the latter. During the NIST competition, a quantum approach based on Grover's

algorithm was presented in [23] to reduce the complexity of finding pre-images of Keccak (and many other SHA-3 candidates).

To summarize, pure brute force attacks on Keccak using Grover's is intractable even for the QA except in certain circumstances with small pre-images, and is thought to be intractable generally even against quantum differential cryptanalysis. By decreasing the pre-image space, this attack can become practical for a QA if not a CA, although still costly.

2.3.2 Quantum Access to Random Oracles and Fiat-Shamir

The Random Oracle Model and the Fiat-Shamir heuristic both suffer criticism even in the classical setting (see [24], [25], [26], and [27] for details). Throughout this technical note, applications of the Fiat-Shamir heuristic to make interactive protocols non-interactive can be improved with respect to the QA by using the Unruh heuristic from [26]. This is costly in terms of proof size and verification time. With quantum access to random oracles, a quantum computer with a particular instantiation of a hash function may compute the hash of a superposition of exponentially many inputs, obtaining a superposition of the hash digests. In short, the QA can gain information about the digests of exponentially many inputs given a polynomial number of queries. Simulation-based proofs of security in the classical setting that rely upon rewinding do not carry over directly. For example, if the adversary makes a query with a superposition of all inputs, the random oracle cannot be simulated "on the fly" by flipping coins at random.

We remark that it is unlikely that the Unruh heuristic will lead to practical drop-in replacements suitable for use in Monero, however. Indeed, Monero signatures are ring signature variants of the Schnorr signatures of [28], which is an interactive identification protocol made non-interactive with the Fiat-Shamir heuristic, which is not thought to be resistant to a quantum adversary. This suggests that Schnorr-like signatures may be forgeable in the presence of the QA. Of course, tampering with the Fiat-Shamir component of these signatures generally requires more quantum resources than computing the discrete logarithm of a key, which can then be used to classically construct signatures at will.

We do not enumerate herein the consequences of quantum access to a random oracle, but we remark that it may cause certain problems to be exponentially easier than in the classical case (see [29] for details). For the vulnerabilities described below, details regarding the random oracle are unnecessary.

3 Technical Overview of Vulnerabilities

If such an adversary were to exist, a number of Monero's fundamental mechanisms would be vulnerable to malicious tampering. Below, we describe how these mechanisms are impacted by various known algorithms. The following should not be considered a comprehensive list.

To provide context, the relevant cryptographic mechanisms are briefly introduced, with some simplifications for clarity and brevity. We refer the reader to [30] for a thorough specification of Monero's protocol, mathematics, and implementation.

3.1 Extracting Wallet Private Keys From On-chain Data

In these sections, we explain how a QA could derive wallet seeds from public information like addresses, sub-addresses, and data intended to be posted publicly on the blockchain. In this section, we assume the QA has a database of some known addresses.

First, we recall a bit about the key generation process in Monero. For a group G of order p with generator g, a Monero-style wallet generation involves two keypairs, one for spending and one for viewing. However, the private spend key k_s is used to deterministically compute all other keys in the Monero Core wallet.

The private spend key $k_s \in \mathbb{Z}_{p-1}$ is a random integer sampled from a pseudorandom number generator. The public spend key K_s is a group element obtained by computing $K_s = g^{k_s}$. The 25-word mnemonic "seed phrase" used to backup a wallet is simply k_s (with a checksum) encoded into a base-1626 dictionary for convenience.

A private view key k_v is generated and the corresponding public key $K_v = g^{k_v}$ is computed. In the Monero Core wallet, the private view key is derived from the private spend key by computing the hash $k_v = \mathcal{H}(k_s)$. Further details are unnecessary for our analysis, and it should be noted that k_s and k_v could be independently generated with a pseudorandom number generator to prevent a QA from exploiting this link in other attacks later, but this is useless as a defense in this section.

3.1.1 Key Extraction from a Primary Address

The wallet's primary public address W is the base-58 encoded concatenation of a network prefix N, both public keys K_v and K_s, and a checksum C. Specifically, $W = N||K_s||K_v||C$. Suppose that a QA knows your primary address. The public spend key K_s and the public view key K_v can be parsed directly from the address W, enabling a QA that learns of any Monero address to apply Shor's algorithm to extract the corresponding private spend key k_s. From this, a CA can compute k_v (if k_v is deterministically derived from k_s as in the Monero Core wallet implementation). Even if a user is not using a view key k_v deterministically derived from k_s (e.g. instead selecting two independent pseudorandom numbers for k_s and k_v), the address is still vulnerable to the QA, who can compute k_v by inverting the map $k_v \mapsto g^{k_v} = K_v$ using Shor's algorithm a second time.

The adversary then essentially owns the wallet: they can derive the remaining keys, view the entire history of the wallet, spend any funds within, and so on. Monero does not claim any security against an adversary that has extracted your private key. Thus even publishing a public key (i.e. posting your address) could be dangerous. Moreover, this extraction can take place at any time in the future, retroactively compromising keys if the wallet W can be found later (e.g. from an internet archive).

One mitigation to reduce the impact of compromised keys is to merely to generate a fresh wallet for each transaction, completely emptying the previous wallet, and to communicate wallet addresses securely to senders, rather than posting the address publicly. This is obviously a poor mitigation. The attacks in this section could, in principle, be mitigated with some quantum-resistant key encapsulation scheme. Such an approach is certain to have far-reaching implications throughout the Monero protocol, and it is not clear what such a protocol may look like in toto. It seems the most practical alternative is to move entirely away from discrete logarithm-based key infrastructures.

3.1.2 Key Extraction From a Sub-Address

Monero enables the creation of many sub-addresses from a single wallet, such that outputs to all addresses can be decrypted by the wallet's main private view key k_s, but the sub-addresses cannot be linked by a CA. Using Shor's algorithm, keys can be extracted from sub-addresses as well. Similar to the last section, suppose that a QA knows your sub-address, for example from OSINT collection or collusion with an exchange.

The ith sub-address W_i contains the i^{th} public spend key $K_{s,i} := K_s \cdot g^{\mathcal{H}(k_v,i)}$ and i^{th} public view key $K_{v,i} := K_{s,i}^{k_v}$ where $\mathcal{H}$ is a hash function. Thus if the QA learns of a sub-address $(K_{s,i}, K_{v,i})$, the QA can apply Shor's algorithm twice to compute $(k_{s,i}, k_{v,i})$, and can apply Shor's algorithm a third time to compute the discrete logarithm of $K_{v,i}$ with respect to $K_{s,i}$, obtaining k_v. The QA can then classically brute force $k_s^{(i)} = k_{s,i} + \mathcal{H}(k_v, i)$ for each i in some small range to find candidate private spend keys $k_s^{(i)}$. With high probability, at least one of these is the spend key corresponding to K_s, which can be checked classically by exponentiating g.

Furthermore, note that sub-addresses use a hash function $\mathcal{H}$. Under the random oracle model, $g^{\mathcal{H}(k_v,i)}$ is a uniformly random group element. Simulation-based proofs of security for schemes using keys based on Monero-style sub-addresses cannot be assumed to remain valid if the random oracle is quantum-accessible, although we have not identified any particular vulnerability.

3.1.3 Key Extraction from One-Time Addresses

Monero one-time keys are computed using the key generation function $f_{otk} : \mathbb{Z} \times \mathbb{Z}_p \times \mathbb{G}^2 \to \mathbb{G}$ mapping (i, r, K_s, K_v) to the one-time key $P = K_s g^{\mathcal{H}(K_v^r || i)}$. Suppose the QA has no information besides public data on the blockchain, i.e. no database of keys collected via open-source intelligence. Then a single one-time address P is not sufficient to compute the private spend key k_s even for the QA. One of Monero's classical security features is that addresses can be safely re-used, due to Monero's one-time addresses, which prevent a CA from linking transactions to the same recipient or identifying the real address behind the one-time address. The manner in which Monero one-time keys are computed may not be secure against a QA.

Monero one-time keys are of the form $(R, P) = (g^r, g^p)$ for some transaction key $R = g^r$ and $P = K_s g^{\mathcal{H}((K_v)^r, i)}$. The discrete logarithm is $p = k_s + \mathcal{H}((K_v)^r, i)$. In the random oracle model, the output of $\mathcal{H}$ is a uniform random variable, so under the random oracle model, p perfectly hides k_s with $\mathcal{H}((K_v)^r, i)$.

We assume the QA intends to analyze a single one-time public key (R, P) they observe on the blockchain; we can assume (without loss of generality) that the QA is not the sender of this transaction, otherwise the QA already knows the recipient (K_s, K_v). The fact that the adversary has a quantum-accessible random oracle is not relevant, since only one P with discrete logarithm $p = k_s + h$ is published where h is some hash digest. Since both k_s and h are unknown to the QA, the QA cannot even applying Grover's algorithm to find a pre-image for h (and, as mentioned in Section 2.2, Grover's applied to a hash digest for a 256-bit input takes too long to be practical anyway).

3.1.4 Key Extraction from a Sample of Multiple One-Time Addresses

If the QA were granted control over the input r, k_v, and i in the hash, then it is possible this sample of several one-time addresses can be used to construct an instantiation of the hidden subgroup problem, which is vulnerable to Shor's algorithm. The hidden subgroup is the kernel of the natural group epimorphism $\pi : \mathbb{G} \to \mathbb{G}/\langle K_s \rangle$. Finding the hidden subgroup provides the generator K_s. Hence, if the QA can obtain a sample of one-time keys $P_1, \ldots, P_n \in \mathbb{G}$ for some $n = O((\log |\mathbb{G}|)^3)$, the QA may be able to obtain K_s. Applying Shor's once more would provide k_s.

However, if the QA does not have control over r, k_v, and i, then the QA does not have access to the random oracle used while executing Shor's. The degree to which the QA may be able to obtain K_s without ultimate control over oracle query inputs is therefore not known.

We have no guarantee that re-use of keys today is secure against key extraction by a future hypothetical QA. If any address or sub-address (say with keypair (K_s, K_v)) has received more than one transaction in the history of the blockchain, then the those one-time keys are the sample of one-time keys used above. Since anybody with knowledge of your address may send multiple outputs to one of your addresses, it is possible that key re-use allows the QA to extract private keys, at least until proven otherwise.

Note that in the current Monero core implementation, the change output for every transaction is sent to index 0 of the current account. Consequently, any account that sends two (or more) transactions with change will result in address reuse, regardless of the number of incoming transactions. However this could be avoided by modifying wallet behavior, for example reserving a subset of subaddresses for one-time change receipt.

3.2 Violate Signer Ambiguity from On-chain

In this section, we describe how the QA can identify which ring member was used in constructing a ring signature, using only on-chain data. We do not assume in this section that the QA has a database of known keys or addresses.

To prevent double-spends, Monero transactions require the publication of all images of the true signing keys used in all ring signatures for the transaction under a one-way function. From on-chain data, a QA can use a ring of public one-time keys and the linkability tag to extract the corresponding private one-time key.

For each transaction input, the signer includes a public linkability tag J and a ring containing output one-time keys, say $\{P_1, \ldots, P_n\}$, where n is the ring size. At the time of this writing, the protocol enforces a ring size of

exactly $n = 11$. The message signer knows the private key p_π corresponding to some public key P_π for one of the ring members, and learned the public keys for the other $n - 1$ decoy keys by sampling them from the blockchain.

Monero uses linkability tags $J := (\mathcal{H}(P))^p$. Under the discrete logarithm assumption, the CA cannot ascertain which ring index π corresponds with the key P_π used by the signer to compute the linkability tag J, although it can compute $\mathcal{H}(P_i)$ for each i. The QA, on the other hand, can also compute the discrete logarithm $\widehat{p}_i$ of J with respect to each $\mathcal{H}(P_i)$. For some index π, $J = (\mathcal{H}(g^{\widehat{p}_\pi}))^{\widehat{p}_\pi}$. The QA concludes that the true signing key was P_π and has, in the course of drawing this conclusion, learned p_π.

This is a rather inefficient application of quantum resources, requiring many executions of Shor's algorithm. If the QA has resources to spare, the QA can merely compute the discrete logarithm of each new P as it is posted to the blockchain with Shor's algorithm and use it to compute the associated linkability tag J. This reduces the signer ambiguity of Monero to anonymity sets with cardinality 1.

All transactions published on the blockchain can be made retroactively non-ambiguous by a QA. A necessary but insufficient modification to protect Monero from the QA is to use linkability tags that are not deterministically computed in the discrete-logarithm setting.

3.3 Violate Transaction Balancing

In this section, it is supposed that the QA has obtained some Monero, the QA can construct transactions at will, and that the QA employs a variant of the Monero protocol where amount Pedersen commitments do not have deterministically computed blinders. Note that the Monero Core implementation uses deterministically computed blinders, but this is not enforced, allowing trickery.

Monero ring confidential transactions in the style of [31] are inspired by Bitcoin-style confidential transactions from [32], swapping usual digital signatures for ring signatures. These transactions are proven to be balanced using range proofs of transaction amounts. In this section, we show how the QA can violate transaction balancing in two different ways, and we mention mitigations for each.

If the Monero community were to learn of the existence of a practical QA before this attack is mitigated, it would be impossible to verify the supply with a classical computer.

3.3.1 Attack: Breaking Binding in Pedersen Commitments

In the event that the QA obtains any Monero, the QA can arbitrarily manipulate the money supply by breaking the binding property of Pedersen commitments. Monero amounts are encoded in perfectly hiding and computationally binding Pedersen commitments from [33], which are included as part of the input keys. The Pedersen commitment scheme uses two basepoints, g_1 and g_2, whose discrete logarithms are unknown with respect to each other. In the Core Monero implementation, $g_1 = g$ (the same as the public key basepoint) and $g_2 = \mathcal{H}(g)$ for some hash function $H : \{0,1\}^n \to G$. To commit to an amount b with a mask y, we compute $C(y, b) = g_1^y g_2^b$. To open some C, we reveal (y, b) and verify that $C = C(y, b)$.

Note that the map taking (y, b) to C is many-to-one: for any (y, b), there are many choices of $(y', b') \neq (y, b)$ such that $C(y', b') = C(y, b)$.

To open some $C = C(y, b)$ to some $(y', b') \neq (y, b)$, a CA has few choices but to brute force search for any (y', b') that will do the trick. For a CA operating in PPT, this is considered intractable. In this sense, the Pedersen commitment scheme is computationally binding. However, the QA can violate computational binding. The QA can apply Shor's algorithm to compute the discrete logarithm of g_2 with respect to g_1 or *vice versa*, say $g_2 = g_1^\gamma = g^\gamma$. To open $C(y, b)$ to a different value $b' \neq b$, the QA can merely compute $y' = y + \gamma(b - b')$ classically. Then $C = C(y, b) = C(y', b')$ yet $b \neq b'$.

This may resolve itself as the following attack. The QA receives a Monero output with amount b and some blinder y, which is a shared secret between the QA and the previous sender, i.e. both know how to open $C(y, b)$. The QA decides to open $C(y, b)$ to the amount $C(y', 2b)$ for some other y' to construct the next transaction, thus doubling their money.

Since the QA knows the value γ such that $g_2 = g_1^\gamma$, and since $C(y,b) = g_1^y g_2^b = g_1^{y+\gamma b}$, the QA solves $y + \gamma b = y' + \gamma(2b)$ for $y' = y - \gamma b$. Then $g_1^{y'} g_2^{2b} = g_1^{y-\gamma b} g_2^{2b} = g_1^y g_2^{2b-b} = C(y,b)$ and the QA can then use y' and $2b$ to classically compute an ostensibly valid transaction.

3.3.2 Mitigation: Restrict Blinders

Blinders for amount Pedersen commitments in the Monero Core implementation are computed deterministically with a hash function. This prevents the QA from selecting an arbitrary blinder y' to open the commitment when sending to a third party.

Specifically, Monero blinders are double hashes of the Diffie-Hellman shared secret K_v^r, i.e. $y = \mathcal{H}(\texttt{pre} \,||\, \mathcal{H}(K_v^r, i))$ for a prefix `pre`. The QA may attempt to violate transaction balancing as in the previous section. However, the attack is successful only if the recipient opens their transaction to an amount different than the original committed value b, such as a negative value. The QA may have a victim's public keys (K_v, K_s) and apply Grover's algorithm to find a pre-image for y', say $y' = \mathcal{H}(x)$, such that $x = \mathcal{H}(\texttt{pre} \,||\, y)$. The QA can then apply Grover's algorithm a second time to find a pre-image for y of the form $y = \mathcal{H}(z \,||\, i)$. The QA can then seek some r such that z is the bitstring representation of K_v^r for the target K_v. This is certainly grossly inefficient.

Instead, the QA can merely compute (k_v, k_s) from the known (K_v, K_s) to own the victim's wallet entirely, rather than trying to pull off a tricky transaction balance scam. Indeed, although the QA can attempt the attack described above quadratically faster than the CA, the repeated application of Grover's to find pre-images without restrictions on the search space drowns this approach with computation time far exceeding the age of the universe. This is especially true if the pre-image for deterministic blinders are long as described in Section 2.2. However, the attacker could employ the attack from Section 3.3.1 to merely send funds to themselves, sequentially doubling (or more) their funds with each transaction. They can open their final commitment as desired, and the attacker can then ultimately use a usual deterministic blinder to send freshly minted money to a third party.

Another alternative is to compute blinders from a verifiable random function and include zero-knowledge proofs (with suitable soundness) that the blinders were computed correctly inside the transaction. This approach has the benefit of preventing the attacker from sequentially multiplying their funds, presuming the verifiable random function is suitability quantum-resistant.

An interesting route for future research may lie in seeking a faster quantum algorithm to violate binding in Pedersen commitments with deterministically computed blinders in the quantum-accessible random oracle model.

3.3.3 Mitigation: Switch Commitments

The money supply of Monero can be protected efficiently by modifying the protocol to use switch commitments instead of Pedersen commitments. These commitments provide a failsafe in the event of the arrival of a quantum computer. Switch commitments are introduced in [34]. Based on ElGamal commitments, switch commitments already contain usual Pedersen commitments, and so they can be considered an extension of the commitment scheme Monero already uses. In this way, modifying the protocol would require minimal changes.

Switch commitments are homomorphic commitments based on ElGamal commitments, allowing arithmetic performed on the committed amounts to act compatibly with arithmetic performed on plaintext amounts. Unlike Pedersen commitments, an opener can convince a verifier that some C commits to a certain value with either a *partial opening* or a *full opening*. If the opener partially opens a commitment, then the scheme is computationally binding and perfectly hiding. If the opener fully opens a commitment, then the scheme is statistically binding and computationally hiding.

Thanks to these properties, the Monero protocol can be modified to use switch commitments with partial openings until it is thought that the arrival of scalable quantum computers is imminent, at which point the protocol can change to full openings. Such changes are not trivial for the Monero codebase, but present the shortest path to mitigating this particular vulnerability.

3.3.4 Attack: Breaking Bulletproof Range Proofs

Bulletproofs, the proving system used to construct range proofs in Monero, are based upon the discrete logarithm hardness assumption. The interactive bulletproof protocol presented in [35] is perfect special honest verifier zero-knowledge and perfectly complete. Hence, even the QA cannot use the transcript of a passing bulletproof to extract the secret data being proved and cannot find a witness that fails to pass verification.

However, bulletproofs have only been proven to satisfy computational witness-extended emulation to our knowledge. It is possible a quantum computer could apply Shor's algorithm to all the basepoints used in bulletproofs, and produce an accepting bulletproof such that the emulator cannot extract a witness. Moreover, problems with the Fiat-Shamir heuristic with respect to the QA detailed in [27] indicate that bulletproofs may suffer unforeseen issues in the presence of the QA.

3.3.5 Mitigation: Quantum-Resistant Range Proofs

Zero knowledge proofs based on problems thought to be hard even for quantum computers may replace bulletproofs in the implementation of Monero range proofs for hardening Monero against the QA. See, for example, [36] for a lattice-based approach. To the knowledge of the authors, no multivariate range proof has been proposed, however.

3.4 Violate Unlinkability

In this section, we assume that the QA has a database of known public wallet addresses. This section should not be confused with the attacks of Sections 3.1. There, the attacker extracts keys, but here the attacker wishes to link one-time keys due to sharing a common recipient. Note that an attacker can use the techniques of Section 3.1 to extract private keys from their database and link transactions directly, subsuming the consequences of this section.

Monero's one-time addresses are intended to prevent key re-use from allowing transactions to be linked by recipient. One-time addresses work by masking recipient public keys. These one-time addresses can be formed either from subaddreses or the primary address, and are computed using hash functions.

Using a combination of Shor's and Grover's algorithm, a quantum computer could reveal the genuine public keys behind each of the one-time addresses being used in a transaction. The trick is to narrow down possible pre-image values to decrease the size of the message space needed to run iterations of Grover's algorithm.

Upon seeing a new transaction relayed on the Monero network, say with one-time key $P = K_s g^{\mathcal{H}(K_v^r||i)}$ and transaction key $R = g^r$ for some unknown K_s, K_v, r, i, the QA can compute r by applying Shor's algorithm. Then, iterating through i, the QA can apply Grover's to look for any pair of keys $(K_v^{(j)}, K_s^{(j)})$ in the database satisfying $P = K_s^{(j)} g^{\mathcal{H}((K_v^{(j)})^r||i)}$. In this way, the QA can link all transactions sent to the same keypair, despite the usage of one-time keys. Although the approach described is very slow due to applying Shor's to each passing transaction, it is still exponentially faster than any similar approach by any CA. Moreover, this approach has no error in the sense that the approach either works or the recipient key is not in the database.

This is not necessarily the best usage of quantum resources. If the QA has a database of keys, the QA can instead dedicate their resources to computing all the corresponding discrete logarithms. Once a key has been broken, the QA has essentially owned the key, and so can easily classically check all incoming transactions just as any other Monero user, without computing the logarithm of each passing transaction key.

Users who always generate new wallets can avoid unlinkability problems. This mitigating measure demonstrates that Monero's one-time keys, intended to solve linkability issues in Monero, do not provide full protection in against a QA.

Replacing the Diffie-Hellman key exchange wholesale with a quantum-resistant key exchange is not sufficient to mitigate the threat presented in this section. Indeed, since the shared secret is hashed, this still allows the QA to apply Grover's to find a pre-image by searching among a known database, even with large key sizes. If the database is small enough, the QA still finds correct solutions very quickly whenever they are to be found. Due to this, mitigations for attacks on unlinkability by the QA are likely to be found among quantum-resistant one-time signature schemes.

3.5 Decrypt Payment Identifiers

Monero transactions optionally[§] come with payment identification that consist of the XOR between a bitstring message and a mask. The mask is generated from the hash of K_v^r, where the transaction key is $R = g^r$. Thus, in any of the previous sections where the QA can compute k_v using Shor's algorithm, the QA can apply Shor's algorithm a second time to R to compute r and compute the mask X directly.

It seems that encrypting payment identifiers with a post-quantum scheme under some post-quantum key infrastructure should be sufficient to prevent the problem of this section with rather conservative changes to the codebase.

3.6 Colliding Transaction Identifiers

The first step of calculating a transaction's identifier $\mathcal{T}$ is splitting up various fields in a Monero transaction into three sets:

1. d_1 contains the the transaction version, inputs, ouptuts, and `extra` fields
2. d_2 contains the signature type, fee, pseudo-output commitments for inputs, encrypted transaction amounts, and output commitments
3. d_3 contains the ring signatures and range proofs

These are hashed to produce the transaction identifier by $\mathcal{T} = \mathcal{H}(\mathcal{H}(d_1)||\mathcal{H}(d_2)||\mathcal{H}(d_3))$. To create a collision that still retains sufficient structure to be correctly parsed, a QA fixes two of the inputs to the final hash function, and searches for a collision in a subfield of the third.

Begin with a valid transaction $T_\circ$ with transaction identifier $\mathcal{T}_\circ$ and select one subfield f for modification, and note the input set containing it as d_f. If a QA can effectively search for preimages with fixed prefix and suffix, then they could craft an alternate payload g such that $g \neq f$ (implying $d_g \neq d_f$) but $\mathcal{H}(d_g) = \mathcal{H}(d_f)$ to cause a collision. Consider modifying the transaction `extra` field in d_1, and note that $\mathcal{H}(\mathcal{H}(d_f)||\mathcal{H}(d_2)||\mathcal{H}(d_3)) = \mathcal{T}_\circ = \mathcal{T}_g = \mathcal{H}(\mathcal{H}(d_g)||\mathcal{H}(d_2)||\mathcal{H}(d_3))$. The transaction with the alternative payload T_g will have the same transaction identifier as the original transaction $T_\circ$, and will be properly formatted as a transaction, but will fail verification.

If an adversarial peer relays a valid block referencing $\mathcal{T}$ along with T_g (the version of the transaction containing the alternative payload), the victim node will mistakenly mark the entire block (and all subsequent blocks on that chain) as invalid. This could be combined with network-layer shenanigans (e.g. sending some nodes $T_\circ$ and other nodes T_g) to cause disruptions.

Note that our previous complaints about Grover's algorithm need not apply in this section. Indeed, even finding a second pre-image g with a small number of bits is sufficient for disruptions, allowing the QA to succeed at this attack more quickly that it seems at first glance. For example, finding a 32-bit pre-image for some $h \in \{0,1\}^{256}$ selected uniformly at random takes, on average, 2 to 3 iterations of Grover's algorithm when such a pre-image exists, and approximately 1 in every 8 digests in $\{0,1\}^{256}$ has a pre-image. Since finding even one bad pre-image is sufficient for trickery, a naive estimate suggests that 24 executions of Grover's is sufficient to find a 32 bit second pre-image (at least in expectation).

Note that if the target datum is required to contain a 256-bit root hash of a Merkle tree, even this does not prevent the attack in this section when this root hash is fixed. Indeed, canonically embedding a fixed string into a message before hashing is equivalent to domain separation, allowing the attacker to find a low-entropy pre-image.

[§]For transaction uniformity, the Core wallet includes encrypted payment IDs on all 2-out transactions, and plaintext payment IDs are ostensibly deprecated. However, neither rule is enforced by consensus, so we continue to observe transactions with no payment IDs, and transactions with plaintext payment IDs. Examples from September 2020 include 1-in/2-out transaction [37] with no payment ID, and transaction [38] with a plaintext payment ID.

3.7 Colliding Block Hashes

Similar to the previous section, the QA gains extra capabilities when playing with block hashes, rather than transaction hashes. Indeed, a block hash $\mathcal{B}$ is produced by hashing block data d, just as in the previous section, allowing the QA to find a pre-image d' just as before. However, the QA can restrict their search to values of d' that correspond to the values of an empty block. Recall that including a prefix on a hash separates the domain of the hash function, leading to a distinct hash functions for each prefix. For example, given a hash function $\mathcal{H}$, one obtains two independent hash functions by computing $\mathcal{H}(0 \mid\mid d)$ or $\mathcal{H}(1 \mid\mid d)$.

In this way, if anyone publishes a block hash $\mathcal{B} = \mathcal{H}(d)$ for some non-empty block d, then the QA can search for some data d' corresponding to an empty block and (with high probability) a different nonce. The QA can then relay the new empty block with block hash $\mathcal{B}$ to peer nodes. Victim nodes will mistakenly think the block at this height is empty, and any transactions later referencing transactions inside this block will be considered invalid by victim nodes. This approach could also be combined with network-layer shenanigans to cause disruptions and potentially consensus faults or chain splits.

It must be noted, however, that a block with data d contains a Merkle root hash of the transaction tree contained inside. In the case of Monero, this is a 256-bit hash digest acting as a pre-image for a block identifier. This suggests that Grover's algorithm is still not sufficient to pull off the trick in this section by replacing a good block with an empty block. Note that if the Merkle tree consists of empty (bad) transactions, then the root hash can be considered a fixed domain separator and the attacker can still attempt to fool a victim node.

However, just as in the previous section, a pre-image with a lower number of bits may be found much more quickly, allowing the QA to carry out other trickery by sending to victim nodes some bad data that ostensibly satisfies hash chain requirements.

3.8 Violate Transaction Confidentiality

Pedersen commitments are perfectly hiding. Given a set of transactions using blinders selected with high entropy, the QA cannot look at one of the Pedersen amount commitments and compute the transaction amount.

However, as discussed in Section 3.3.2, blinders are computed deterministically in Monero. If the QA has a database of known keys, the QA can apply Shor's algorithm to obtain the transaction key r for each transaction, compute K_v^r for each K_v in their database, and compute the blinder y corresponding to each K_v directly.

Note that, given a new transaction, each key K_v in the database can be used to compute a deterministic blinder, and the amount Pedersen commitment in the transaction can be un-blinded with this blinder to compute a plausible transaction amount. This plausible transaction amount is a uniformly random element from $\mathbb{Z}_p$ when K_v is not the recipient key.

In Monero, valid commitments computed by semi-honest parties have amounts on some small set $\{0, 1, \ldots, 2^n - 1\}$ where 2^n is much smaller than the group order p. Each plausible transaction amount, however, is a uniform random variable from the scalar field, and the probability that a plausible amount corresponding to a non-recipient K_v lies is in this small set is exceedingly small.

Even for a database with a very large number of keys, generally only at most one keypair (K_v, K_s) will correspond to a valid transaction amount in the range $\{0, \ldots, 2^n - 1\}$, and this keypair (K_v, K_s) is highly likely to be the recipient of the transaction, revealing both the recipient and amount. The probability of a false positive (a non-matching key that by chance results in a plausible amount within the valid range) is approximately the ratio of the size of the range from the range proof to the size of the group: $\sim 2^{64}/2^{255} = 2^{-191}$.

Of course, given K_v and K_s, the QA can check whether this keypair is the recipient by computing k_v via Shor's. This allows the QA to compute the transaction amount directly as in previous sections.

4 Alternatives to Elliptic Curve Cryptography

4.1 Lattice-based

Lattice-based cryptography stands as the likeliest contender for eventually replacing the now-widespread RSA protocol system, and so should also be looked at as a possible replacement for ECC used in Monero. While RSA and ECC both depend on number-theoretic problems solvable by a quantum algorithm in polynomial time, lattice-based cryptography is grounded in problems proven to be NP-complete or NP-hard. Both complexity classes consist of problems thought to be outside of BQP, an important class of problems decidable by a quantum computer.

Lattice problems are extremely well-studied. The problems are so well-studied that some problems predating Diophantus in the 3^{rd} century C. E. can be reduced to lattice-related problems, for example. Lattice problems have been increasingly featured in cryptography literature, especially since [6]. Lattice-based cryptography is one of the most widely-tested approaches among the extant choices for post-quantum cryptography.

Summarily, the above factors alone may suffice to conclude that lattice-based cryptography is the top contender to harden Monero against a quantum adversary. The flexibility of lattice-based approaches may allow components to be replaced piecemeal, rather than requiring a single migration event.

4.1.1 From a Geometric Point of View

Lattice-based cryptography comes in several flavors, and several can be visualized. The reader can develop some intuition by thought experiments about hiding low-dimensional linear subspaces in a high-dimensional measure space. An arbitrary 0-dimensional object (a point) in a 1-dimensional measure space with a suitable measure (like the real number line) has "length" 0. Similarly, a 0-dimensional point or a 1-dimensional object (like a line or a curve) in a 2-dimensional measure space (like a plane or the surface of a sphere) has "area" (measure) 0. An arbitrary 0-, 1-, or 2-dimensional surface in a 3-dimensional measure space has "volume" (measure) 0.

A similar statement holds when sets are events and the measure is a suitable probability measure. For intuition, it suffices to accept that a low-dimensional event in a high dimensional event space occurs with negligible probability.

Consider a uniformly random selection from the scalar field $\mathbb{Z}_p$ over which $G = \langle g \rangle$ is some cyclic elliptic curve group with generator g and order p. For a given public key g^x, selecting a scalar uniformly at random $y \leftarrow \mathbb{Z}_p$ yields $g^y = g^x$ with probability $1/p$; if G is selected to have order 2^n for some security parameter n, then the probability of successfully guessing the private key this way is a negligible function in n. Indeed, this is analogous to finding a zero-dimensional hidden point in a one-dimensional space.

In particular, a public-key cryptosystem can be built upon ostensibly random, high-dimensional, (public) matrices and the selection of a secret and short basis of some subspace.

For example, the Short Integer Solution problem reduces to presenting to the player with a public matrix A and challenging the player to find some $\mathbf{x}$ in some normed space such that $A\mathbf{x} = \mathbf{0}$ and such that $\mathbf{x}$ has a sufficiently small norm.

4.1.2 Differences Between Lattice-based Cryptography, RSA, and ECC

Crucial qualitative differences lie between lattice-based cryptography and both RSA and ECC.

In particular, unlike RSA and ECC, and like multivariate-based cryptography, lattice-based cryptography hardness assumptions come from provably NP-complete and NP-hard problems, both of which are thought to lie outside of BQP.

As of this writing, no quantum algorithm for solving such lattice problems has promised any substantial performance advantage over the best known classical algorithms, and any such algorithm that can operate in polynomial time can be used to solve any NP problem in polynomial time.

4.1.3 Migration

Lattice cryptography is not the most disruptive among the range of plausible future-oriented methods. Importantly, lattice-based cryptography scales demonstrably well (at least in most cases) and its protocols feature some of the shortest key sizes compared to other post-quantum families. This could make it far less of a strain than certain other options to implement on existing systems.

Yet, while lattice based cryptography holds many promising advantages, it is not without its downsides. For example, though the latticed solution may enable much smaller key lengths compared to some other prospective methods, the lattice-based keys are still larger than those utilized by current RSA protocols, and proving times can also be burdensome.

4.2 Multivariate-based

Multivariate cryptosystems essentially use multivariable polynomial inverses as private keys and the composition of the corresponding multivariable polynomials as public keys. A cryptosystem, then, can be constructed from inversion of the public key. Public keys appear to be random functions, and yet the purported owner can invert the function. Multivariate schemes are usually quick and computationally efficient, which allows their use on devices with smaller computational power like smart cards and RFID chips (see, e.g. [39] and[40]).

The hardness assumption underlying multivariate cryptography is based on an NP-complete problem, which is thought to be outside of BQP.

The security of multivariate protocols is fundamentally reliant upon the problem of solving nonlinear polynomials over a finite field, which is NP-complete. Multivariate quadratic polynomials are linear combinations of monomials with total degree at most 2. That is, with variables $\{x_1, \ldots, x_n\}$, multivariate quadratic polynomials over a finite field $\mathbb{F}$ are linear combinations

$$p(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i x_j + \sum_{i=1}^{n} b_i \cdot x_i + c$$

where $\mathbf{x} = (x_1, \ldots, x_n)^\top$ is a vector of indeterminates over $\mathbb{F}$ and the coefficients $a_{i,j}, b_i, c \in \mathbb{F}$. The name originates from "multi," signifying "more than one," and "variate," standing for "random variables." The sum containing linear terms here can be written with dot product notation $\mathbf{x} \cdot \mathbf{b}$ where $\mathbf{b} = (b_1, \ldots, b_n)^\top \in \mathbb{F}^n$. The sum containing the super-linear terms may also be represented with dot product notation $p = \mathbf{x} \cdot (A\mathbf{x})$ where $W \in \mathbb{F}^{n \times n}$ is the matrix where the $(i,j)^{th}$ entry is $a_{i,j}$. Hence, a quadratic multivariate polynomial can be written $p(\mathbf{x}) = \mathbf{x} \cdot (A\mathbf{x} + \mathbf{b}) + c$. Using this notation, we represent a system of m quadratic multivariate polynomials over R with the following column vector

$$\mathbf{p}(\mathbf{x}) = \begin{pmatrix} \mathbf{x} \cdot \left(A^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \right) + c^{(1)} \\ \mathbf{x} \cdot \left(A^{(2)}\mathbf{x} + \mathbf{b}^{(2)} \right) + c^{(2)} \\ \vdots \\ \mathbf{x} \cdot \left(A^{(m)}\mathbf{x} + \mathbf{b}^{(m)} \right) + c^{(m)} \end{pmatrix}.$$

Checking that some $\mathbf{x}$ is a solution to $\mathbf{p}(\mathbf{x}) = 0$ is as simple as evaluating m polynomials, but finding such a solution given a random p is NP-complete.

Multivariate-based cryptography is founded upon very well-studied problems, just as lattice-based cryptography. Public key multivariate-based cryptosystems use advances from many numerous mathematical fields, including operator theory, nonlinear dynamics, algebra, and complex analysis. As such, multivariate public key cryptography enjoys a plethora of techniques based on foundational mathematics. Like lattice-based cryptography, multivariate problems hold promising new prospects for post-quantum cryptography, making it possible to build fully homomorphic encryption protocols secure against quantum adversaries with great flexibility. Furthermore, multivariate-based protocols promise both compact key sizes as well as short encryption/decryption times.

Judging by the number of papers published on each topic, multivariate-based cryptography has seen relatively less interest than lattice-based cryptography. At the very least, this restricts protocol design for lack of schemes to choose from. Entire transaction protocols with some Monero-like security features are available (see [41], [42], [43]). The relative difference in interest suggests at least a nontrivial yet qualitative gap between the approaches.

Multivariate-based cryptography offers many promising qualities beyond future-oriented cybersecurity. Perhaps the most promising among them is relatively small key and signature sizes in comparison to other post-quantum cryptography schemes. Nonetheless, a relative deficit of quality research material compared to other post-quantum cryptographic schemes might be disadvantageous but also may offer opportunities for future developers and researchers.

4.3 Hash-based

Hash-based post-quantum cryptography involves, as the name suggests, cryptographic protocols based on the principle of generating of keys by the means of secure hash functions. The considerable difficulty of compromising this type of cryptography primarily rests on the pre-image resistance of said hash functions. As detailed above, a hypothetical QA (Quantum Adversary) efficiently implementing Grover's algorithm may remain viable to rendering hash-based schemes vulnerable, but careful selection of hash parameters and the construction of certain tree-based protocols can prevent this and secure schemes against the QA. The extent to which quantum differential cryptanalysis can be leveraged is highly dependent upon protocol architecture and hash function choice.

The XMSS (eXtended Merkle Signature Scheme) protocol is a hash-based post-quantum cryptography scheme currently used in the Quantum Resistant Ledger (QRL). The QRL is a decentralized blockchain cryptocurrency which essentially holds the same functionality as the Bitcoin public ledger, while offering the added bonus of being resistant to quantum adversaries. While hash-based cryptographic protocols are fairly resistant to quantum adversaries, they still carry a few downsides like large key and signature sizes.

4.4 Supersingular Elliptic Curve Isogeny-based Cryptography

Supersingular Elliptic Curve Isogeny based cryptography involves hard problems surrounding the isogenies of elliptic curves. Many current cryptographic protocols involve the difficulty of solving problems relating to elliptic curves, but whereas ECC is based on the difficulty of inverting canonical maps from a field to an elliptic curve group over that field, SSEI-based cryptography uses the difficulty of finding isogenies between supersingular elliptic curves. An isogeny is an epimorphism of algebraic groups with a finite kernel. Another way to look at this class of problems is that it works by finding well-behaved maps between generated elliptic curves. These classes of hard math problems seem to hold promise in being resistant to both quantum and classical adversaries.

5 Alternative Protocols

In this section, we review some alternative schemes that are relevant for benchmark comparisons. Specifically, we present a general proving system, ZK-STARKs, a transaction protocol thought to be quantum-resistant, MatRiCT, and two ring signature schemes that are thought to be quantum-resistant. Although these schemes do not all serve an identical purpose, we summarize their space and time requirements in Table 1.

5.1 ZK-STARKs

All problems in NP have zero-knowledge proofs, shown in [44]. ZK-STARKs (Zero-Knowledge Scalable Transparent ARguments of Knowledge) is a general zero-knowledge proving system for languages in NP. ZK-STARKs require no trusted setup and could be used to replace range proofs and signatures in Monero. Zero-knowledge proofs allows one party to prove statements (such as having enough funds available for a transaction) to a third party without revealing hidden information to the third party (such as the total funds available for a transaction). This type of security

protocol forms a fundamental backbone of the Monero cryptocurrency, and ZK-STARKs provides an alternative to this that is also quantum secure.

ZK-STARKs consists of schemes thought to be secure against the QA and these are well-described in [45].

5.2 MatRiCT

More than one lattice-based implementation of a ring confidential transaction protocol has been proposed, e.g. [41] and [43]. The only protocol that appears practically efficient is MatRiCT (Matrix Ring Confidential Transactions) from [41]. This protocol has impressive features and presents novel sub-schemes, but has at least one critical drawback preventing implementation: the lack of one-time addresses to protect recipients from being exposed.

The Monero protocol is based upon linkable ring signatures to mask transaction senders, a homomorphic commitment scheme to mask transaction amounts, a range proof scheme to prove the masked amount is positive and will not cause overflow/wraparound errors in transaction balance, and one-time addresses to mask transaction recipients. MatRiCT uses the Module-Shortest Integer Solution (M-SIS) and Module-Learning With Error (M-LWE) problems to provide most of these functionalities without Gaussian sampling, including a novel extractable commitment scheme but excepting the one-time addresses.

The proof length of the protocol is around two orders of magnitude shorter than the existing post-quantum proposal, and scales efficiently to large anonymity sets. In particular, a transaction can be generated in time between 250 ms and 3600 ms and can be verified in time between 23 ms and 250 ms depending on anonymity levels and hardware. The public key size for MatRiCT is about 4 kB, which is about the same as for RSA. Transactions comparable to those of Monero take about 25 ms to verify and occupy around 100 kB of space.

With relatively short keys and relatively fast verification times, MatRiCT is a top contender for modifying Monero for QA resistance, although the proposal in [41] lacks any one-time address functionality, exposing recipients of each transaction. This can be mitigated: if recipients never receive more than one transaction to any given public key (i.e. no key re-use), then the security features of MatRiCT are similar to those of Monero. This mitigation is less-than-desirable, though, since users tend to fall back to whatever default mode they are provided. Recall that the author(s) of [46] recommend that keys not be re-used even with Bitcoin, but address re-use is common.

Overall, MatRiCT offers a replacement for RingCT and most of the other mechanisms of Monero that would be vulnerable to a QA, except for one-time addresses. Besides this caveat, MatRiCT (or something like it) is a very promising candidate.

5.3 Raptor-512

Raptor, introduced in [47], is one of the most practical lattice-based (linkable) ring signature protocol. It boasts small (although unfortunately linear) signature sizes of about 1.3 kB per ring member and provides a framework for nearly all the current functionality of security features for Monero.

Raptor operates using so-called "chameleon hash plus" functions as the main building blocks in its construction. Such functions are implementable in the lattice setting and operate as a collision-resistant keyed trapdoor hash function. With the trapdoor, it is possible to easily find collisions for any input. Chameleon hash plus functions are a fundamental building block that distinguishes Raptor from other lattice based protocols and schemes.

Linkable-Raptor-512 is a linkable ring signature based on Raptor with similar signature sizes per ring member as Raptor-512. Linkable-Raptor-512 public keys are 0.9 kB (compare to 64 bytes in Monero).

Signature generation times are claimed to be on the order of 50 ms dependent upon implementation and hardware, and verification times are under 50 ms. Small ring sizes without linkability tags have verification times on the same order as that of Monero, with verification claimed to take place in under 3 ms.

5.4 RingRainbow

Multivariate Cryptography schemes provide some of the smallest signature sizes amongst the post-quantum secure cryptography schemes so far proposed. RingRainbow, proposed in [48], is a multivariate-based ring signature scheme that provides small signature sizes, low verification times, and resistance to the QA. Moreover, RingRainbow is constructed from simple and efficient techniques applied to the Rainbow signature scheme. Multivariate cryptography is quite flexible, especially with respect to ring signatures; see [49], [50], [51], and [52].

We briefly describe the techniques used to construct RingRainbow in this section. The resulting scheme displays perfect anonymity among ring members, short keys, and short, fast-to-verify signatures.

The Rainbow signature scheme works by generating private keys off of two invertible affine functions and a quadratic function. The public key is computed from these private keys such that access to any of the private keys is sufficient to invert the public key. To sign a message M, the hash is computed $h = \mathcal{H}(M)$. The signer uses the private part of their keypair (sk, pk) to compute a signature σ such that $pk(\sigma) = h$; recall that the public key pk is a function and finding pre-images for a random pk is equivalent to violating the hardness underlying the NP-complete multivariate problem.

To construct a ring signature for a message M with ring of public keys $\{pk_1, \ldots, pk_R\}$ using the private key sk_ℓ for some index ℓ, the signer first computes $h = \mathcal{H}(M)$ as before, selects uniformly at random some decoy random values $\{\sigma_i\}_{i \neq \ell}$, and computes $\widehat{h} = h - \sum_{i \neq \ell} pk_i(\sigma_i)$. The signer then uses their private key to compute z_ℓ such that $pk_\ell(z_\ell) = \widehat{h}$. The ring signature is $(z_1, \ldots, z_R)$. To verify the signature, the verifier computes $\sum_i pk_i(z_i)$. In the case that sk_ℓ was used to compute the signature semi-honestly, we split this sum into $pk_\ell(z_\ell) + \sum_{i \neq \ell} pk_i(z_i)$. We note that in the semi-honest case, $pk_\ell(z_\ell) = \widehat{h} = h - \sum_{i \neq \ell} pk_i(z_i)$, so $\sum_i pk_i(z_i) = h = \mathcal{H}(M)$. Moreover, computing a solution to $\sum_i pk_i(z_i) - h = 0$ without knowledge of the corresponding sk_i is hard, so even a quantum computer cannot forge a signature.

This ring signature scheme, when formalized appropriately, is provably post-quantum and classically secure, with the smallest signature sizes of all the post-quantum methods so far outlined.

A RingRainbow signature takes up only 43 bytes of space (the smallest of all signatures so far presented here), with a public key size of about 47 kB. Generation time takes about 13 ms and each signature can be verified in less than 25 ms. As shown, the main feature that makes RingRainbow stand out is its small signature size.

6 Key Size/Verification Time Table

Table 1 displays the shortest key sizes and verification times given by each protocol, including RSA and ECC for reference. The hardware used for finding the necessary generation and verification times have clock rates that range from 20 MHz to 3 GHz. For RSA, verification time for slow hardware (under 20 MHz) for a 1 kB signature is 2 s, while at faster hardware (over 3 GHz) it is 13 ms. Most of the other key generation and verification times are estimated for hardware operating at about 3 GHz.

	Sig/Proof Size (kB)	**Public Key Size** (kB)	**Gen Time** (ms)	**Verify Time** (ms)	**Usage**
Raptor-512	1.2	0.9	29 - 57	3 - 50	Ring Signature Scheme
RingRainbow	0.043 - 0.43	47 - 460	13 - 3100	10 - 31000	Ring Signature Scheme
Monero	0.25	0.064	50 - 70	10 - 20	Transaction protocol
MatRiCT	0.65 - 6.5	4 - 10	240 - 3500	20 - 220	Transaction Protocol
ZK-STARK	0.064 - 100	10 - 120	100 - 60000	50 - 70	Proving system

Table 1: Comparison of cryptography protocols and their relevant storage and time requirements. Time to verify and generate relevant artifacts for the different schemes should be taken as ballpark estimates, as related to the other cryptographic protocols and will vary widely for different hardware. Best and worst case as such are shown. Storage requirements for the ring signature schemes display signature and public key sizes for anonymity sets containing 5 to 50 members.

7 Conclusions

The development of scalable quantum computers capable of efficiently using many qubits to implement Shor's algorithm poses an existential threat to most cryptocurrencies and banking systems, including Monero. This is mostly due to reliance upon the assumption that computing discrete logarithms in elliptic curves is difficult, which is not a safe assumption if quantum computers are leveraged. This threat is also due to certain vulnerabilities in the practical construction of decentralized ledgers, since crafting messages that cause hash collisions can disrupt the security of transaction and block identifiers. To worsen matters, recent advances in qubit control has reduced the expected number of qubits required for such hijinks.

The ideas presented herein should not be construed as a comprehensive list of vulnerabilities or techniques that a QA might one day leverage to undermine the functionality of Monero or other cryptocurrencies. We note that retroactive deanonymization puts today's Monero users at the hands of tomorrow's (quantum or classical) adversaries. If practical quantum computers that can break Monero's encryption arrive at any point in the future, then users' lifelong transaction history will become public for ingestion by the AdTech industry, stalkers, criminals, and governments. It is irrelevant which party publishes a deanonymized copy of the Monero blockchain first - the universal evaporation of privacy is irreversible.

Most of the functionality in the Monero protocol and Core implementation is vulnerable to a quantum adversary. Thankfully, for many of the threats, there exist schemes viably considered as secure against a quantum adversary and efficient enough to be used for eventual structural replacements in Monero. The alternatives to elliptic curve cryptography listed in this technical note offer schemes and protocols that can be effectively utilized to replace much of the functionality and security mechanisms of Monero.

Since the majority of Monero security features are based on elliptic curve cryptography, if there's anything that should be gleaned from this technical note, it's that someday these features should be based on another form of cryptography that is not as vulnerable to Shor's algorithm.

Contrary to the common misconception that all post-quantum cryptography is impractical in terms of storage or verification time, Table 1 demonstrates some alternatives are approaching practical implementation. Since the attacks laid out herein are generally retroactive and present an eventual threat to the privacy of Monero users, we recommend that performance tradeoffs are weighed carefully against the implications for today's users if their private financial data are later revealed. The timeline for which these mechanisms should be replaced with post-quantum secure variations is a debate best left to the Monero community at large. We recommend that the community take recent advancements in the field of quantum computing into account during this discussion, noting that Shor's algorithm, Grover's algorithm, and Simon's algorithm have already been experimentally realized on actual quantum computers.

Some of the mitigating measures presented herein are "low-hanging fruit" in terms of implementation. For

example, switch commitments are only one group element larger than Pedersen commitments and require only a small modification to bulletproof range proofs. Thus switch commitments are an ideal, low-friction candidate for protecting the monetary supply of Monero against the threat of quantum computers.

On the other hand, some of the attacks presented here result in failures of crucial cryptographic underpinnings, such as the extraction of private keys from public keys, which catastrophically breaks the key generation methods utilized by essentially every cryptocurrency. A more drastic modification to the Monero codebase would be required to handle some of these issues. These attacks are also retroactive, so failing to address these problems during the quantum grace period (our current era where we understand the capabilities of quantum computers, but cannot yet construct them at practical cryptography-breaking scale) may leave today's Monero users exposed at an unknown later date. Migration cannot begin after practical quantum computers have arrived; the ledger is only meaningful if the migration is completed before a quantum adversary enters the scene. We urge the Monero community to begin considering migration to a post-quantum secure key infrastructure and transaction protocol at some future date.

8 Acknowledgements

We greatly appreciate the thorough review and insightful observations from Sarang Noether. Example transactions were provided by Neptune from Noncesense Research Lab. We especially thank the dozens of community members who sponsored this research through the Monero CCS. Additional thanks to Insight for offering a space to pursue open-source privacy tech research.

Thanks also for helpful edits and technical advice from Aleksey Calvin, Kunal Marwaha, Cheyenne Nelson, and `#monero-research-lab`.

Additional personal thanks to Judy Young, Tim Corbo, Curtis Robinson, Jesse Korson, Jennifer Ayala and Cameron Wells. And of course a big thanks also goes to all our dogs, cats, friends, relatives, and partners who supported us during this project.

A Technical Implementation of Quantum Algorithms

The introduction to this technical note outlines some of the capabilities a theoretical quantum adversary could leverage in order to introduce vulnerabilities into the current Monero security infrastructure. This appendix serves as a more in depth technical look into the algorithms that can be run efficiently on quantum hardware in order to provide these capabilities. The three algorithms provided in this appendix include Shor's, Grover's and Simon's algorithm. While these three do provide some of the largest security risks to Monero as outlined, the capabilities of a quantum computer are not limited to them. Not included in this group are many algorithms that can be used in QDC (Quantum Differential Cryptanalysis) such as the Bernstein-Vazirani algorithm and others. The main threat factor from QDC would be in reversing or revealing hidden information about the pre-image of a hash digest.

In the following, we present casual descriptions of textbook implementations of the quantum algorithms presented in this technical note. Example code and results in the appendices utilize the open source qiskit python library, and quantum hardware available through the IBM Quantum Experience.

A.1 Shor's Algorithm Technical Implementation

Shor's algorithm violates the RSA hardness assumption (see [53]) and the elliptic curve discrete logarithm hardness assumption (see [9]), both with polynomial time complexity.

Elliptic curves over finite fields form abelian groups. For a (not necessarily cyclic) group $\mathbb{G}$ with some generator $g \in \mathbb{G}$, g^n is the n-fold application of the group operation with g, i.e.

$$g^n = \underbrace{g \cdot g \cdot \ldots \cdot g}_{n}. \tag{1}$$

If $g' = g^x$ for some $g \in \mathbb{G}$ and a field element x, we say x is the discrete logarithm of g' with respect to g. $\mathbb{G}$ is cyclic, i.e. $\mathbb{G} = \langle g \rangle$, such that $g^p = \mathrm{id}_{\mathbb{G}}$, then every element $g' \in \mathbb{G}$ can be written $g' = g^x$ for some $x \in \mathbb{Z}_p$.

The discrete logarithm hardness assumption is that, for an elliptic curve group, it is intractable for an algorithm to output x given only g and $g' = g^x$. The following is a description of Shor's algorithm as applied to elliptic curves, primarily based on [9].

After initializing the qubits to zero, a unitary transformation is applied to the registers such that the quantum state of the system can be described as

$$|\psi\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{y=0}^{2^n-1} |x, y, g^x (g')^y\rangle \,. \tag{2}$$

To compute $g^x (g')^y$, first we repeatedly square the group elements g and g', thus getting the multiples $P_i = g^{2^i}$ and $Q_i = (g')^{2^i}$. We then compute the products $P_i Q_i$ and write

$$g^x (g')^y = \prod_i g^{x_i} (g')^{y_i}. \tag{3}$$

where x_i and y_i are the i^{th} bits in the binary expansions of x and y, respectively. These group elements can be classically precomputed and $|x, y\rangle$ can be described with a single qubit. This is done by using the semi-classical quantum Fourier transform from [54], and is analogous to the factoring algorithm used in breaking RSA. Thus, by using an accumulator register R, we can represent $|x, y, g^x (g')^y\rangle = |x, y, R\rangle$ describing $|x, y\rangle$ with a single qubit.

From these steps, the problem reduces to multiplying a fixed pre-computed classically known point P_i with a superposition of points. This product essentially shifts the discrete logarithm for each component in the superposition by the same quantity. This leads to the necessity of unitary transformations U_{P_i}, U_{Q_i} acting on a state $|S_i\rangle$, which represents a point on the elliptic curve, i.e. $U_{P_i} : |S\rangle \to |S \cdot P_i\rangle$ and $U_{Q_i} : |S\rangle \to |S \cdot Q_i\rangle$.

These steps are applied n times to g^x and n times to $(g')^y$ where $n \approx log_2(q)$. This series of steps decomposes the discrete logarithm quantum algorithm into a sequence of group shifts by constant classically known elements. After these are done, the measured output will correspond to a distribution with peaks around scalars (e.g. private keys) equal to Nk/q and Ndk/q where $N = 2^n$. Multiplying by q/N provides k and dk, where d is the discrete logarithm value we are trying to find.

Shor's algorithm executed by a quantum computer is exponentially faster than any known classical algorithm to compute discrete logarithms.

```
"""The following is python code utilizing the qiskit library that can be run on extant quantum
hardware using 5 qubits for factoring the integer 15 into 3 and 5. Using period finding,
for a^r mod N = 1, where a = 11 and N = 15 (the integer to be factored) the problem is to find
r values for this identity such that one can find the prime factors of N. For 11^r mod(15) =1,
results (as shown in fig 1.) correspond with period r = 4 (|00100>) and r = 0 (|00000>).
To find the factor, use the equivalence a^r mod 15. From this:
(a^r -1) mod 15 = (a^(r/2) + 1)(a^(r/2) - 1) mod 15.In this case, a = 11. Plugging in the two r
values for this a value yields (11^(0/2) +1)(11^(4/2) - 1) mod 15 = 2*(11 +1)(11-1) mod 15
Thus, we find (24)(20) mod 15. By finding the greatest common factor between the two coefficients,
gcd(24,15) and gcd(20,15), yields 3 and 5 respectively. These are the prime factors of 15,
so the result of running Shor's algorithm to find the prime factors of an integer using quantum
hardware are demonstrated. Note, this is not the same as the technical implementation of Shor's
algorithm described in this section for breaking the discrete log hardness assumption,
though the proof of concept remains."""

# Import libraries
```

```
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from numpy import pi
from qiskit import IBMQ, Aer, QuantumCircuit, ClassicalRegister, QuantumRegister, execute
from qiskit.providers.ibmq import least_busy
from qiskit.visualization import plot_histogram

# Initialize qubit registers
qreg_q = QuantumRegister(5, 'q')
creg_c = ClassicalRegister(5, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.reset(qreg_q[2])
circuit.reset(qreg_q[3])
circuit.reset(qreg_q[4])

# Apply Hadamard transformations to qubit registers
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])

# Apply first QFT, modular exponentiation, and another QFT
circuit.h(qreg_q[1])
circuit.cx(qreg_q[2], qreg_q[3])
circuit.crx(pi/2, qreg_q[0], qreg_q[1])
circuit.ccx(qreg_q[2], qreg_q[3], qreg_q[4])
circuit.h(qreg_q[0])
circuit.rx(pi/2, qreg_q[2])
circuit.crx(pi/2, qreg_q[1], qreg_q[2])
circuit.crx(pi/2, qreg_q[1], qreg_q[2])
circuit.cx(qreg_q[0], qreg_q[1])

# Measure the qubit registers 0-2
circuit.measure(qreg_q[2], creg_c[2])
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[0], creg_c[0])

# Get least busy quantum hardware backend to run on
provider = IBMQ.load_account()
device = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= 3 and
                                    not x.configuration().simulator and x.status().operational==True))
print("Running on current least busy device: ", device)

# Run the circuit on available quantum hardware and plot histogram
```

```
from qiskit.tools.monitor import job_monitor
job = execute(circuit, backend=device, shots=1024, optimization_level=3)
job_monitor(job, interval = 2)

results = job.result()
answer = results.get_counts(circuit)
plot_histogram(answer)

#largest amplitude results correspond with r values used to find the prime factor of N.
```

Github repo for this code can be found at this link:

https://github.com/hamburgerguy/Quantum-Algorithm-Implementations/blob/master/Shor.py

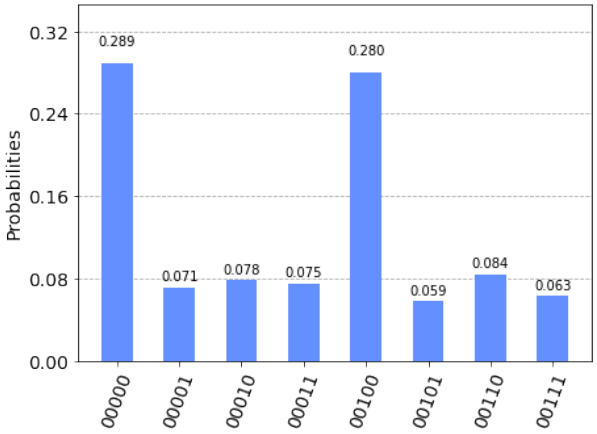

Figure 1: Histogram output from above code of Shor's algorithm to factor 15 on ibmq-16-melbourne quantum computing hardware. Results correspond with theory.

A.2 Grover's Algorithm Technical Implementation

Given a set of data, a targeted implementation of Grover's Algorithm can procure from this data a specific labeled instance x_0 which can be for example, the confidential pre-image of a hash digest. These are the steps one would take to do so:

1. Initialize the qubits.
$$|\psi\rangle = |0\rangle^{\otimes n}$$
2. Put the qubits in an equal state of superposition.
$$|s\rangle = H^{\otimes n}|0\rangle^n$$
3. Apply an oracle reflection $|U_f\rangle$, which is a unitary transform, to the marked instance x_0 of the qubits.
4. Apply an additional reflection,
$$U_s = 2|s\rangle\langle s| - \mathbb{1}$$
such that this maps the state to $U_s U_f|s\rangle$.
5. Repeat steps 3 and 4 until the state of the system can be described as $|\psi_t\rangle = (U_s U_f)^t|s\rangle$. This takes approximately $\sqrt{n} =$ t times, where n is the number of entries in the data set. The qubits are measured, and the corresponding amplitude corresponds to the equivalent classical bits of the target entry.

```
"""Python implementation of Grovers algorithm through use of the Qiskit library to find the value 3 (|11>)
out of four possible values. """

#import numpy and plot library

import matplotlib.pyplot as plt
import numpy as np

# importing Qiskit
from qiskit import IBMQ, Aer, QuantumCircuit, ClassicalRegister, QuantumRegister, execute
```

```
from qiskit.providers.ibmq import least_busy
from qiskit.quantum_info import Statevector

# import basic plot tools
from qiskit.visualization import plot_histogram

# define variables, 1) initialize qubits to zero
n = 2
grover_circuit = QuantumCircuit(n)

#define initialization function
def initialize_s(qc, qubits):
    '''Apply a H-gate to 'qubits' in qc'''
    for q in qubits:
        qc.h(q)
    return qc

### begin grovers circuit ###

#2) Put qubits in equal state of superposition
grover_circuit = initialize_s(grover_circuit, [0,1])

# 3) Apply oracle reflection to marked instance x_0 = 3,  (|11>)
grover_circuit.cz(0,1)
statevec = job_sim.result().get_statevector()
from qiskit_textbook.tools import vector2latex
vector2latex(statevec, pretext="|\\psi\\rangle =")

# 4) apply additional reflection (diffusion operator)
grover_circuit.h([0,1])
grover_circuit.z([0,1])
grover_circuit.cz(0,1)
grover_circuit.h([0,1])

#If running on actual quantum hardware

# Load IBM Q account and get the least busy backend device
provider = IBMQ.load_account()
device = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= 3 and
                                    not x.configuration().simulator and x.status().operational==True))
print("Running on current least busy device: ", device)

# 5) measure the qubits
grover_circuit.measure_all()

#If run on qubit simulator
qasm_simulator = Aer.get_backend('qasm_simulator')
```

```
shots = 1024
results = execute(grover_circuit, backend=qasm_simulator, shots=shots).result()
answer = results.get_counts()

#If run on actual quantum hardware

from qiskit.tools.monitor import job_monitor
job = execute(grover_circuit, backend=device, shots=1024, optimization_level=3)
job_monitor(job, interval = 2)

results = job.result()
answer = results.get_counts(grover_circuit)
plot_histogram(answer)

#highest amplitude should correspond with marked value x_0 (|11>)
```

Github link for running Grover's algorithm on quantum hardware:

https://github.com/hamburgerguy/Quantum-Algorithm-Implementations/blob/master/Grover.py

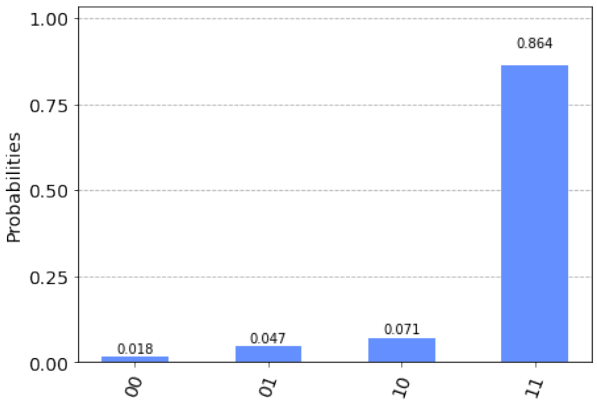

Figure 2: Histogram output from above code of Grover's algorithm run on ibmqx2 quantum computing hardware. Results correspond with theory.

A.3 Simon's Algorithm Technical Implementation

Simon's problem is, given a function $f : \{0,1\}^n \to \{0,1\}^n$ that is known to be invariant under some n-bit XOR mask a, determine a. In other words, given $f(x) = f(y)$ if and only if $x \oplus y \in \{0, a\}$, compute a.

This can be used to set up a system of linear equations that can be used to find the outputs and find the XOR mask of certain functions, which has numerous applications in cryptography.

1. Initialize two n-qubit registers to 0.
$$|\psi_1\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes n} \tag{4}$$

2. Apply a Hadamard transform to the first register.
$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle^{\otimes n} \tag{5}$$

3. Apply a query function Q_f.
$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle \tag{6}$$

4. Measure second register, causing the first register to become
$$|\psi_4\rangle = \frac{1}{\sqrt{2}} \left(|x\rangle + |y\rangle\right). \tag{7}$$

5. Apply Hadamard transform to the first register.
$$|\psi_5\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{z\in\{0,1\}^n} \left[(-1)^{x\cdot z} + (-1)^{y\cdot z}\right] |z\rangle \tag{8}$$

6. Measure the first register, which gives an output if
$$(-1)^{x\cdot z} = (-1)^{y\cdot z}. \tag{9}$$

The output will correspond to a string z, this string will map to $b \cdot z = 0 \pmod 2$ which from Gaussian elimination can be used to find the XOR mask to the function $f(x)$.

```
"""Qiskit code for running Simon's algorithm on quantum hardware for 2 qubits and b = '11'. """

# importing Qiskit
from qiskit import IBMQ, BasicAer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, execute

# import basic plot tools
from qiskit.visualization import plot_histogram
from qiskit_textbook.tools import simon_oracle

#set b equal to '11'
b = '11'

#1) initialize qubits
n = 2
simon_circuit_2 = QuantumCircuit(n*2, n)

#2) Apply Hadamard gates before querying the oracle
simon_circuit_2.h(range(n))

#3) Query oracle
simon_circuit_2 += simon_oracle(b)

#5) Apply Hadamard gates to the input register
simon_circuit_2.h(range(n))

#3) and 6) Measure qubits
simon_circuit_2.measure(range(n), range(n))

# Load saved IBMQ accounts and get the least busy backend device
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= n and
                                    not x.configuration().simulator and x.status().operational==True))
print("least busy backend: ", backend)

# Execute and monitor the job
from qiskit.tools.monitor import job_monitor
shots = 1024
job = execute(simon_circuit_2, backend=backend, shots=shots, optimization_level=3)
job_monitor(job, interval = 2)

# Get results and plot counts
```

```
device_counts = job.result().get_counts()
plot_histogram(device_counts)

#additionally, function for calculating dot product of results
def bdotz(b, z):
    accum = 0
    for i in range(len(b)):
        accum += int(b[i]) * int(z[i])
    return (accum % 2)

print('b = ' + b)
for z in device_counts:
    print( '{}.{} = {} (mod 2) ({:.1f}%)'.format(b, z, bdotz(b,z), device_counts[z]*100/shots))

#the most significant results are those for which  b dot z=0(mod 2).

'''b = 11
11.00 = 0 (mod 2) (45.0%)
11.01 = 1 (mod 2) (6.2%)
11.10 = 1 (mod 2) (6.4%)
11.11 = 0 (mod 2) (42.4%)'''


Github link to this code:

https://github.com/hamburgerguy/Quantum-Algorithm-Implementations/blob/master/Simon.py
```

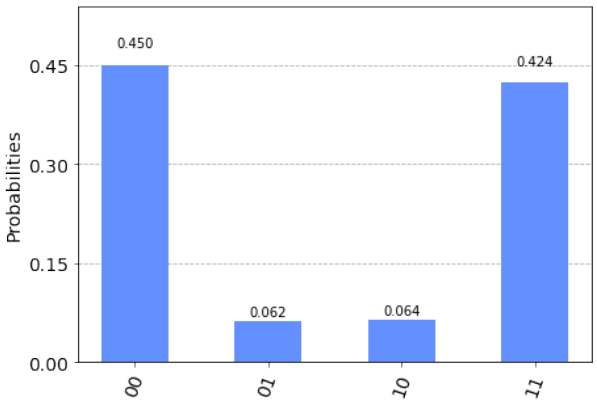

Figure 3: Output from above code of Simon's algorithm run on ibmqx2 quantum computing hardware. Results correspond with theory.

References

[1] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[2] Andreas Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, Inc., 2017.

[3] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum*. O'Reilly Media, Inc., 2018.

[4] SerHack. *Mastering Monero*. Lernolibro LLC, 2018.

[5] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, version version 2020.1.14. `https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf`, 2020.

[6] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[7] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[8] Daniel R. Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[9] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[10] Michele Mosca. Quantum algorithms. *arXiv preprint arXiv:0808.0369*, 2008.

[11] Steven D Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78(1):51–72, 2016.

[12] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[13] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[14] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, and Christian Schaffner. Quantum preimage, 2nd-preimage, and collision resistance of sha3. *IACR ePrint*, 302:2017, 2017.

[15] Daniel J. Bernstein. "quantum attacks against blue midnight wish, echo, fugue, grøstl, hamsi, jh, keccak, shabal, shavite-3, simd, and skein. `cr.yp.to/papers.html#quantumsha3`, 2010.

[16] Pascal Koiran, Vincent Nesme, and Natacha Portier. A quantum lower bound for the query complexity of simon's problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1287–1298. Springer, 2005.

[17] Xavier Bonnetain. Tight bounds for simon's algorithm.

[18] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[19] Thomas Santoli and Christian Schaffner. Using simon's algorithm to attack symmetric-key cryptographic primitives. *arXiv preprint arXiv:1603.07856*, 2016.

[20] Shengbao Wu and Mingsheng Wang. Security evaluation against differential cryptanalysis for block cipher structures. *IACR Cryptol. ePrint Arch.*, 2011:551, 2011.

[21] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Cryptol. ePrint Arch.*, 2020:371, 2020.

[22] Nathaniel Graff. Differential power analysis in-practice for hardware implementations of the keccak sponge function. 2018.

[23] Daniel J. Bernstein. Quantum attacks against blue midnight wish, echo, fugue, grøstl, hamsi, jh, keccak, shabal, shavite-3, simd, and skein.

[24] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[25] Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the fiat-shamir paradigm. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 102–113. IEEE, 2003.

[26] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.

[27] Dominique Unruh. Post-quantum security of fiat-shamir. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–95. Springer, 2017.

[28] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[29] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 41–69. Springer, 2011.

[30] Koe, Kurt M. Alonso, and Sarang Noether. *Zero to Monero: Second edition v2.0.0.* 2020.

[31] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[32] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

[33] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[34] Tim Ruffing and Giulio Malavolta. Switch commitments: A safety switch for confidential transactions. In *International Conference on Financial Cryptography and Data Security*, pages 170–181. Springer, 2017.

[35] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[36] Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *Annual International Cryptology Conference*, pages 147–175. Springer, 2019.

[37] Monero mainnet transaction c62c6707f5f12939456b3df3839b6b8a6102fe331bdaaf3241ae2d7f5fdf4c2b. `https://xmrchain.net/tx/C62C6707F5F12939456B3DF3839B6B8A6102FE331BDAAF3241AE2D7F5FDF4C2B`, block 2180052 on 2020-09-05.

[38] Monero mainnet transaction 6d2bcb7a61b002cf8ad695f459cf56546c55809d719d96d48ae48ea74c98be21. `https://xmrchain.net/tx/6D2BCB7A61B002CF8AD695F459CF56546C55809D719D96D48AE48EA74C98BE21`, block 2176830 on 2020-09-01.

[39] Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp, and Christopher Wolf. Time-area optimized public-key engines: Mq-cryptosystems as replacement for elliptic curves? In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 45–61. Springer, 2008.

[40] Anna Inn-Tung Chen, Ming-Shing Chen, Tien-Ren Chen, Chen-Mou Cheng, Jintai Ding, Eric Li-Hsiang Kuo, Frost Yu-Shuang Lee, and Bo-Yin Yang. Sse implementation of multivariate pkcs on modern x86 cpus. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 33–48. Springer, 2009.

[41] Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Matrict: efficient, scalable and post-quantum blockchain confidential transactions protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 567–584, 2019.

[42] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1. 0). In *Australasian Conference on Information Security and Privacy*, pages 558–576. Springer, 2018.

[43] Wilson Alberto Torres, Veronika Kuchta, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, and Jacob Cheng. Lattice ringct v2. 0 with multiple input and multiple output wallets. In *Australasian Conference on Information Security and Privacy*, pages 156–175. Springer, 2019.

[44] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[45] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[46] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system.

[47] Xingye Lu, Man Ho Au, and Zhenfei Zhang. Raptor: a practical lattice-based (linkable) ring signature. In *International Conference on Applied Cryptography and Network Security*, pages 110–130. Springer, 2019.

[48] Mohamed Saied Emam Mohamed and Albrecht Petzoldt. Ringrainbow–an efficient multivariate ring signature scheme. In *International Conference on Cryptology in Africa*, pages 3–20. Springer, 2017.

[49] Albrecht Petzoldt. Efficient key generation for rainbow. In *International Conference on Post-Quantum Cryptography*, pages 92–107. Springer, 2020.

[50] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann. A multivariate based threshold ring signature scheme. *Applicable Algebra in Engineering, Communication and Computing*, 24(3-4):255–275, 2013.

[51] Murat Demircioglu, Sedat Akleylek, and Murat Cenk. Efficient gemss based ring signature scheme. *Malaysian Journal of Computing and Applied Mathematics*, 3(1):35–39, 2020.

[52] Mohamed Saied Emam Mohamed, Albrecht Petzoldt, and C RingRainbow. Efficient multivariate ring signature schemes. *IACR Cryptol. ePrint Arch.*, 2017:247, 2017.

[53] Edward Gerjuoy. Shor's factoring algorithm and modern cryptography. an illustration of the capabilities inherent in quantum computers. *American journal of physics*, 73(6):521–540, 2005.

[54] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998.