TECHNICAL NOTE MRL-00xx

Evaluating and developing Monero security in a post-quantum world

Adam Corbo, Mitchell Krawiec-Thayer (@isthmus)*, Brandon G Goodell†
Insight, Monero Research Lab

Aug 2020

Abstract

This technical note renders a thorough analysis of Monero's extant weaknesses before a theoretical quantum adversary. Extrapolating from its survey of weaknesses, the discussion herein extends into technical descriptions of plausible solutions, with a focus on their respective practical and theoretical trade-offs. Revisit this when nearing completion

1 Introduction

1.1 Prerequisites and preliminaries

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.

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 a spectrum of 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 $\mathbb{C}^2$; 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 $\int |a_0|^2 + |a_1|^2 \, d\mu = 1$.

Whereas a classical adversary, which we abbreviate with CA, uses computers that operate only with bits, a theoretical quantum adversary (abbreviated QA) has a computer that operates with qubits. The CA adversary can only be expected to complete algorithms using probabilistic Turing machines in polynomial time (PPT algorithms) whereas the QA adversary can solve bounded error, quantum, polynomial (BQP) time algorithms. The algorithms from Sections 1.2.1, 1.3, and 1.4 are BQP algorithms and require qubits to execute efficiently, these algorithms when run with qubits solve certain problems exponentially faster than any CA.

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.

* Isthmus@getmonero.org
† surae@getmonero.org

1.2 The Quantum Adversary's Capabilities

1.2.1 Violate Discrete Logarithm Hardness with Shor's Algorithm

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 answer oracle queries made to f with $O(\log|G| + \log|X|)$ bits. Before Shor's algorithm, it was thought that the following hardness assumption was valid.

Assumption 1.1 (Hidden Subgroup Problem)**.** *There does not exist an algorithm $\mathcal{A}$ with oracle access to f that can output a generating set for H in polynomial time.*

Shor's algorithm for discrete logarithms is a BQP algorithm capable of solving the hidden subgroup problem, violating Assumption 1.1. It is still thought that Assumption 1.1 is valid when restricted to PPT algorithms, such that there does not exist a PPT algorithm that violates Assumption 1.1.

Indeed, if $G' = \langle g \rangle$ is any group of order p, we can compute the discrete logarithm of an ostensibly random $X = g^x$ using $G = \mathbb{Z}_{p-1}^2$, $H = \langle (X, g^{-1}) \rangle$, and the function $f : G \to \mathbb{Z}_p^*$ defined by mapping $(a, b) \mapsto g^a X^b$. 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 [10] and [6].

1.3 Utilize Grover's Algorithm to Find Pre-Images of Hash Functions and Unstructured Search Capabilities

1.3.1 Grover's Algorithm

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 [4] for details, since we do not specify the algorithm here. It is shown in [1] that Grover's algorithm is asymptotically optimal even for quantum adversaries.

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 H, we can define $f(x)$ to be 1 when $H(x) = y$ and 0 otherwise and 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.

1.3.2 Difficulty of Finding the Pre-Image of Hash Digests

Some aspects of security rely on the fact that finding the pre-image of a hash digest is difficult. For a QA, as stated in the last subsection, employing Grover's algorithm, it is possible to find a marked value in an un-ordered set of data in approximately the square root of the number of possible entries to look through. One possible mitigation for this is to double the key lengths used in the hash function, which makes this more difficult to accomplish.

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 on average 2^{255} operations.

This kind of attack, with the state space of about 2^{256} is highly impractical to attempt and as such is considered one way.

Using Grover's algorithm, one could attempt this same attack by setting up the Oracle circuit used in Grover's algorithm with the same hash function in order to mark when the appropriate pre-image would spit out the known hash digest. If you take into account the state space of all possible inputs for a 256 bit collision resistant hash function, this would still take approximately 2^{128} operations of Grover's algorithm to generate an output that corresponded with the appropriate pre-image for the known hash digest. While still significantly better at attempting this attack than a classical computer, for standard clock speeds this would still require a time period greater than the age of the universe to attempt.

If the state space is lowered, for instance if we know one or more of the factors used to generate the hash digest, or can narrow down the possible pre-images in some way, then this attack becomes more feasible for a quantum computer to attempt. Depending on how effective this is at reducing the search space, it will still in most cases take longer than it takes Shor's to break ECC, RSA, etc.

Grover's algorithm can also be used in conjunction with other methods, such as QDC (Quantum Differential Cryptanalysis), to find the pre-image of hash digests. (See 1.4.2).

1.4 Simon's Algorithm, Quantum Differential Cryptanalysis, and Further Capabilities.

1.4.1 Simon's Algorithm

Simon's algorithm, from [13], 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. It is shown in [5] that Simon's algorithm is asymptotically optimal.

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

1.4.2 Quantum Differential Cryptanalysis

Besides brute force attacks, there are some other methods a QA could leverage to reverse or reveal hidden information about the pre-image of a hash digest. 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, or attack symmetric key primitives [12]. For block ciphers relying on a Feistel scheme, which if they are used in the construction of a hash function, would be vulnerable to this advantage a QA could leverage.

Attacks of this nature, namely using an efficient quantum algorithm (Such as the Bernstein-Vazirani algorithm [14], which is beyond the scope of this technical note) to set up a system of linear equations, and then solving these classically and use them to perform differential cryptanalysis, are a bit more difficult to describe than the brute force Grover's algorithm attack. Attacks of this nature require a deep case by case look into the internal architecture of each specific hash function used, (Keccak, chacha20 in the case of Monero) Besides Simon's algorithm and the Bernstein-Vazirani algorithm, other algorithms exist that could be used to perform differential cryptanalysis on the block ciphers used in the construction of each hash function.[**?**]

As far as this technical audit could surmise, Keccak (the hash function used in Monero) is secure against currently known methods employing Simon's or the Bernstein-Vazirani algorithm to perform differential cryptanalysis and find information about or reverse the pre-image of the Keccak hash function. That being said, there could be other methods using these algorithms that were not covered during this audit, with or without using Grover's algorithm in tandem.

For that matter, there are other algorithms as well that can be used to perform quantum differential cryptanalysis other than B-V or Simon's, it is unknown if others exist that aren't as well known as of this writing.

Finding or designing a hash function whose internal architecture is provably resistant to all possible methods of Quantum Differential Cryptanalysis (henceforth QDC), is outside the scope of this technical audit. It could be possible Keccak is already secure against all possible forms of QDC, but until such a proof is found, we will assume for the purposes of this audit that it might be possible for a future QA to utilize such an advantage in an extreme case, maybe in combination with Grover's algorithm, to find the pre-image of a known hash digest.

To summarize, pure brute force attacks on Keccak using Grover's is intractable for a QA. However, by decreasing the state space this attack can become possible to implement for a QA if not a CA, though possibly still difficult. There are a few methods to do this, which could include QDC. Since it is unknown whether or not keccak specifically is susceptible to QDC, it might still be possible for a QA to find the pre-image of a known hash digest even without decreasing the possible state space.

1.4.3 Further Capabilities

The capabilities of a QA as outlined in this article primarily center around the advantages that can be leveraged by independently utilizing Shor's, Grover's, Simon's or some combination of these algorithms to attack the security features present in Monero. The further methods of QDC are not extensively explored, such as using Simon's or the B-V algorithm, but also should not be considered a complete list of algorithms one could attempt QDC with. A QA also inherently has the added capability of being able to generate true randomness rather than relying on a PRNG, which potentially could be useful as well.

2 Technical Overview of Vulnerabilities

For the purposes of probing potential vulnerabilities, we assume a theoretical QA capable of efficiently leveraging the above-detailed quantum-empowered algorithms (as well as, prospectively, other algorithms not yet discovered). If such an adversary were to exist, a number of Monero's core mechanisms would be vulnerable to the plausible implementation of such algorithms. Below, we describe how these mechanisms are impacted by various known algorithms. The following should not be considered a comprehensive list.

2.1 Deriving Wallet Seeds

In this section, we explain how an efficient QA can derive wallet seeds from public information like addresses, sub-addresses, and data intended to be posted publicly on the blockchain like pairs of one-time keys with the same recipient.

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 $k_v = H(k_s)$. 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.

The wallet's primary public address A is the base-58 encoded concatenation of a network prefix N, both public keys K_v and K_s, and a checksum C. Specifically, $A = N||K_s||K_v||C$.

2.1.1 Key Extraction From Addresses

The public spend key K_s and the public view key K_v can be parsed directly from the address A, 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 Core implementation). Even if a user is not using a view key k_v deterministically derived from k_s, a user who is using two independent pseudorandom numbers for k_s and k_v 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. In this way, even publishing a public key (i.e. posting your address) could be dangerous.

2.1.2 Key Extraction From Sub-Addresses

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 subaddresses cannot be linked by a CA. Using Shor's algorithm, keys can be extracted from sub-addresses as well.

The ith sub-address A_i contains the i^{th} public spend key $K_{s,i} := K_s \cdot g^{H(k_v,i)}$ and i^{th} public view key $K_{v,i} := K_{s,i}^{k_v}$. Thus if an adversary learns of a subaddress, a QA can apply Shor's algorithm to $K_{v,i}$ to compute the discrete logarithm with respect to $K_{s,i}$, obtaining k_v. Using k_v the QA can classically compute $H(k_v, i)$ for each allowable i in the core implementation. The adversary can then brute force compute $K_{s,i} \cdot g^{-H(k_v,i)}$ through some small set of possible indexes i and obtain possible group elements, one of which is certain to be the $K_{s,i}$ (in the core implementation).

From this, a CA can compute k_v (if k_v is deterministically chosen using k_s as in the core implementation) or a QA can compute k_v by inverting the map $k_v \mapsto g^{k_v} = K_v$.

2.1.3 Key Extraction From A Single One-Time Address

A single one-time address P is not sufficient to compute the private spend key k_s.

Key extraction from a single one-time key is not possible since one-time keys are perfectly hiding. Indeed, the one-time keys in Monero are of the form (R, P) for some transaction key $R = g^r$ and for $P = K_s \cdot g^{H((K_v)^r,i)}$. The discrete logarithm, then, is $k_s + H((K_v)^r, i)$. Given any P, given any k_s, and given any k_v, there exists an r such that $P = g^{k_s+H((K_v)^r,i)}$.

In this way, P information-theoretically hides k_s. This means that, if only one output has ever been sent to (K_s, K_v), then at most 1 one-time key $g^{k_s+H((K_v)^r,i)}$ appears on the blockchain. Even the QA, who can compute the discrete logarithm $k_s + H((K_v)^r, i)$, cannot determine k_s without additional information.

2.1.4 Key Extraction From A Pair of One-Time Addresses

In the previous section, we explained why a single one-time address is insufficient for key extraction. In this section, we describe how 2 one-time keys can be used to extract the spend key.

One of Monero's classical security features is that addresses can be safely re-used, due to Monero's one-time 'stealth' addresses, which prevent a CA from linking transactions to the same recipient or identifying the real address behind the stealth address.

However, in this sense, Monero is not secure against a QA: re-use of keys today allows a future hypothetical QA to extract the private spend key. If any address or subaddress (say with keypair (K_s, K_v)) has received more than one transaction in the history of the blockchain, then the methods in this section can be applied by a QA with strictly public data to extract the keypair (k_v, k_s).

Since creating a transaction is permissionless, this allows anybody with knowledge of your address to send multiple outputs to your public keys, in the process making your private keys extractable to any QA at any time in the future.

Monero transactions are published with transaction keys $R = g^r$ and one-time (so-called "stealth") keys $P = K_s \cdot g^{H(K_v^r, i)}$. Say that (R, P) and (R^*, P^*) are two pairs of keys for a Monero transaction made to the same address, i.e. $P = K_s \cdot g^{H((K_v)^r, i)}$ and $P^* = K_s \cdot g^{H((K_v)^{r^*}, i^*)}$ for some i, i^*. A classical computer can compute $P' = P \cdot (P^*)^{-1}$ easily. The QA can apply Shor's algorithm to P, P^*, and P' obtaining the discrete logarithms $p = k_s + H((K_v)^r, i)$, $p^* = k_s + H((K_v)^{r^*}, i^*)$, and $p' = H((K_v)^r, i) - H((K_v)^{r^*}, i^*)$, respectively. The QA can then classically compute $p' + p_2 = p' - p_1 = k_s$.

From this, a CA can compute k_v (if k_v is deterministically chosen using k_s as in the core implementation) or a QA can compute k_v by inverting the map $k_v \mapsto g^{k_v} = K_v$.

2.2 Violate Signer Ambiguity

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. In this section, we show how the QA can use a ring of public one-time keys and a key image known to be computed by one of the ring members to extract the corresponding private one-time key.

For each transaction input, the signer includes a public key image 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, it is necessary that all rings have exactly $n = 11$ ring members to be considered valid. 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 key images $J := (H(P))^p$. 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 key image J. The QA, on the other hand, can compute $H(P_i)$ for each i and compute the discrete logarithm $\widehat{p}_i$ of J with respect to each $H(P_i)$. For some index π, $J = (H(g^{\widehat{p}_\pi}))^{\widehat{p}_\pi}$. The QA concludes that the true signing key was P_π and has, in the course of drawing this conclusion, learned p_π.

2.3 Violate Transaction Confidentiality

Since Pedersen commitments are perfectly hiding, and each transaction ostensibly uses fresh randomness, it should seem that even the QA cannot look at an arbitrary Pedersen commitment to a transaction amount and compute the transaction amount. However, the situation is not so nice.

2.3.1 Attack

The transaction amount is restricted to a small set $[2^N]$ for some $N \in \mathbb{N}$. Recall that we commit to an amount b with a mask y by computing $C(y, b) = g_1^y g_2^b$, we see that there are at most 2^N choices of b. This is selected to be small enough so that users can brute-force search for the transaction amount when they see a new transaction such that they can compute the blinder y (i.e. the transactions addressed to them). The QA can then apply Shor's algorithm 2^N times to $g_2^{-b^*} C(y, b)$ for each $b^* \in [2^N]$ to obtain 2^N possible choices of the blinder y. This allows the QA to compute a Monero transaction amount with $O(2^N)$ applications of Shor's algorithm.

2.3.2 Mitigation

Since it is thought that the timeline for Shor's algorithm to reliably compute the discrete logarithm for 32 byte keys is even longer than the timeline for the arrival of the QA, one reasonable short-term mitigation for this violation of privacy is to increase N substantially and to use some quantum-secure method of transmitting the transaction amount, perhaps via secure sidechannel.

2.4 Violate Transaction Balancing

Monero ring confidential transactions in the style of [7] are inspired by Bitcoin-style confidential transactions from [9], 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 and we mention a mitigation.

2.4.1 Attack

Each Monero transaction is defined by some input anonymity sets (called rings) of possible input keys, some output keys, a non-negative plaintext fee, a range proof of the transaction amount including the fee, and a ring multisignature on the input rings. Monero amounts are encoded in perfectly hiding and computationally binding Pedersen commitments from [8], which are included as part of the input keys. A Monero transaction is considered valid if the anonymity sets are subsets of old output keys on the blockchain, the ring multisignature is valid (to authenticate whoever broadcast the transaction), and the range proof is valid (to ensure that transaction balances).

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 $g_2 = H(g)$ for some hash function $H : \{0,1\}^n \to G$. To commit to an amount b with a mask y, we compute $C(y,b) = g_1^y g_2^b$.

Note that this map 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, a 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 resolves itself as the following attack. The QA receives a Monero output with amount b. The QA constructs (i) a range proof for use in a new transaction using amount b and the usual blinder as if they wish to spend all of b, but (ii) also uses the above approach to select a different blinder y' and a different amount $b' < b$ for the recipient to open.

2.4.2 Mitigation

We identify two mitigations for the above attack.

First, if blinders are deterministically computed using random oracles from public data from the blockchain, then the QA the is restricted from selecting y' freely. Depending on specific implementation recommendations, however, the computation for blinders in commitments may still be vulnerable to the QA, so we omit details.

Second, the idea of switch commitments are introduced in [11]. These commitments are homomorphic, so they can be used (similarly to Pedersen commitments) to the degree necessary for usage in a currency protocol. They already contain usual Pedersen commitments, and so they can be considered an extension of the commitment scheme Monero already uses.

These commitments have the following properties. 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.

Switch commitments in Monero would require (i) a single additional group element in transactions, (ii) a slight modification to range proofs resulting in no significant change to transaction size or verification time, and (iii) the computation of an additional scalar by the recipient to open commitments.

Note that when the QA violates the binding property of commitments, they can violate the supply of Monero, which is intended to be a money with a known, fixed supply secure against malicious tampering. On the other hand, when the QA violates the hiding property, they can violate the amount privacy afforded by the confidential transactions of Monero.

A protocol that does not balance transactions cannot function as a currency, whether transactions are confidential or not. A currency without confidential transactions can still function as a currency, however. In this way, we could regard the monetary supply as hierarchically more important to secure than user privacy.

2.5 Violate Unlinkability

The integration of one-time addresses into Monero's functionality has provided transactions with an extra layer of protection. In their essence, one-time stealth addresses work by masking the public keys used in each transaction. These one-time stealth addresses can be formed either from subaddreses or the primary address. Using a combination of Shor's and Grover's algorithm, a quantum computer could reveal the genuine public keys behind each of the stealth addresses being used in a transaction.

Since this mechanism relies on a QA using Grover's algorithm to find the pre-image of a hash digest used to generate the stealth address from spend and view keys, it is somewhat more robust against a QA compared to the other vulnerabilities mentioned, so the trick is to narrow down possible pre-image values to decrease the state space needed to run iterations of Grover's algorithm. For a 256 bit hash digest, even for a quantum computer to find the pre-image using a pure brute force attack this would require approximately 2^{128} iterations of Grover's algorithm to output a corresponding amplitude, which at standard clock speeds would require a runtime much greater than the lifetime of the universe.

If Grover's algorithm was instead used to go through all known public view/spend key pairs until it found a match that output the stealth address with a known r (from Shor's) or perhaps used to go through a smaller state space somehow (methods shall be described below) then this functionality is vulnerable to a QA.

If all transactions made by a public key are generated off a new subaddress that is different each time, this could provide an extra layer of security that might make it much more difficult

If Alice wants to send a transaction to Bob, a one time stealth address is formed as such: First, generating a random integer r to be used only once in this transaction, $K_0 = \mathcal{H}_n(rK_v)G + K_s$ Where K_0 is the one-time stealth address Alice submits this and g^r (the transaction public key) to the network. Once submitted to the network, Bob can see it is meant for him by multiplying the transaction public key using his private view key k_v, since $rk_vG = rK_v$. From this, $K_s' = K_0 - \mathcal{H}_n(rK_v)G = K_s$. This is considered secure classically, however a quantum adversary could break this feature as such: Find r, by factoring g^r using Shor's. Using Grover's, find a K spend and view public key pair amongst existing public keys such that $K_0 = \mathcal{H}_n(rK_v)G + K_s$ for a given r. From this, extracting private keyss is simple, using Shor's to invert $k \mapsto g^k = K$ as described in section 2.1.1.

If a subaddress was used to generate the stealth address instead of a public key address, then this would add an extra layer of security. However, since these are also secured by DLP, it would be possible to find the associated public key address used to generate the relevant subaddress attached to the relevant stealth address from this in much the same manner as shown.

2.6 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.

2.7 Immutability

2.7.1 Block immutability

M question - can Grover's algorithm be used to forge a block with target hash and desired arbitrary payload?

Can you circumvent the pq-PoW hardness of RandomX by brute forcing other parts of the block (for example tx_extra in the miner coinbase)

I have a bit more clarity now about how this might work.

(here, we answer the question: if in the future RandomX is the pq-bottleneck and you can Grover's effectively, what are your options?)

The attacker actually do a proof of work! Investing the full amount of time/energy with with a classical computer

So if you a bunch of block information B and you boil it down to a digest b to hash with RandomX, so we try a bunch of nonces:

$RandomX(b, n_1) \rightarrow$ fail to pass difficulty

$RandomX(b, n_2) \rightarrow$ fail to pass difficulty

$RandomX(b, n_3) \rightarrow$ fail to pass difficulty

$RandomX(b, n_4) \rightarrow$ fail to pass difficulty

$RandomX(b, n_5) \rightarrow$ fail passes difficulty ... winning block

Now, I don't broadcast anything. I just stick (b, n_5) in my back pocket as winning inputs to RandomX. Then, whenever I feel like it, I use Grover's algorithm to find a B preimage for the b that I previously solved.

You don't have to use QA tricks on RandomX if you can pull off QA tricks on its inputs.

Actually, heh, I don't think you have to do any proof of work at all! You could just pick random previously solved blocks, take their (n, b) pair, and make your own $B \rightarrow b$ mapping with Grovers. (of course that would be sloppy, and Noncesense would eventually notice the RandomX nonce collision and start poking around.)

Does this make any sense?

2.7.2 Transaction immutability

M - will finish this up later notes - Txns have no PoW

Grover's to find primage of alternate txn that produces the same txid

Fluffy blocks are a risk if QAs are forging transactions. More broadly, the fact that the block PoW input only ingests transaction IDs instead of transaction data makes this an attack surface

3 Alternatives to Elliptic Curve Based Cryptography

3.1 Lattice-based

3.1.1 From a geometric point of view

3.1.2 Differences between lattice-based cryptography, RSA, and ECC

3.1.3 Migration

3.2 Multivariate-based

3.3 Hash-based

3.4 Supersingular Elliptic Curve Isogeny-Based Cryptography

4 Alternative protocols

4.1 ZK-STARKS

4.1.1 Possible usage

4.2 MatRiCT

4.3 L2RS

4.4 RingRainbow

5 Key Size/Verification Time Table

6 Conclusion

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. The main security concern not included in this group might be from 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.

1. **A.1 Shor’s Algorithm Technical Implementation**

   Shor’s algorithm violates the RSA hardness assumption [3] and the elliptic curve discrete logarithm hardness assumption [10], both with polynomial time complexity.

   Given that some of the largest security assumptions of which Monero is based off of can be solved efficiently using a specialized elliptic curve implementation of Shor’s algorithm, the specific implementation will be shown as such:

Elliptic curves over finite fields form abelian groups. For discrete logarithms over elliptic curves, We consider an elliptic curve, E, over $GF(p)$, where p is a large prime. The base of the logarithm is a point P on the elliptic curve E, whose order is another (large) prime q, such that $qP = 0$. We want to compute the discrete logarithm, d, that lies on another point Q on the elliptic curve such that $Q = dP$.

We apply the following transformation to a set of qubits, such that the quantum state can be described as:

$$|\psi\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{y=0}^{2^n-1} |x, y, xP + yQ\rangle \tag{1}$$

Thus we need a method of computing (large) integer multiples of group elements. This can be done by the standard "double and add technique". This is the same technique used for the modular exponentiation in the factoring algorithm, although there the group is written multiplicatively so it's called the square and multiply technique. To compute $xP + yQ$, first we repeatedly double the group elements P and Q, thus getting the multiples $P_i = 2^i P$ and $Q_i = 2^i Q$. We then add together the P_i and Q_i for which the corresponding bits of x and y are 1, $xP + yQ$ can then be written as

$$xP + yQ = \sum_i x_i p_i + \sum_i y_i Q_i \tag{2}$$

The multiples as such of P_i and Q_i can be computed classically beforehand, creating conditions that can be used describe the state shown in equation (2) for $|x, y\rangle$ using just a single qubit, which drastically cuts down the qubit requirements. This is done by using the semiclassical quantum Fourier transform [2] and is analogous to the factoring algorithm used in breaking RSA. Thus, we can represent $|x, y, xP + yQ\rangle = |x, y, O\rangle$ with O simply being the "accumulator" register, with $|x, y\rangle$ being replaced by the single qubit as mentioned before.

We are left being required to carry out a number of steps whereby we add a fixed (classically known) point P_i (or Q_i) to a superposition of points. We are working in the cyclic group generated by P, thus the effect of a fixed addition is to "shift" the discrete logarithm of each element in the superposition by the same amount. Thus we need unitary transformations U_{P_i} and U_{Q_i} which acts on any basis state $|S_i\rangle$ representing a point on the elliptic curve, as $U_{P_i} : |S\rangle \rightarrow |S + P_i\rangle$ and $U_{Q_i} : |S\rangle \rightarrow |S + Q_i\rangle$.

Applying these steps n times to P and n times to Q where n is approximately $n = log_2 q$. This sequence of steps decomposes the discrete logarithm quantum algorithm into a sequence of group shifts by constant classically known elements. After these are done, the resultant measured output will correspond to a distribution with peaks around points equal to Nk/q and Ndk/q where $N = 2^n$. To obtain the relevant values, k and dk which are ultimately what will be used to solve the DLP, we multiply the observed output values by q/N.

This version of Shor's algorithm for solving the DLP for elliptic curves for 256 bit security can be done efficiently at standard clock speeds in under a minute, which for a classical computer using the most efficient currently known algorithm would take over ten-thousand years to find the same result. As such, it should be obvious why this is a considerable advantage a QA (Quantum Adversary) would have over a CA (Classical Adversary).

2. 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:

(a) Initialize the qubits

$$|\psi\rangle = |0\rangle^{\otimes n}$$

(b) Put the qubits in an equal state of superposition.

$$|s\rangle = H^{\otimes n}|0\rangle^{n}$$

(c) Apply an oracle reflection $|U_f\rangle$ to the marked instance x_0 of the qubits.

(d) Apply an additional reflection,

$$U_s = 2|s\rangle\langle s| - \mathbb{1}$$

Such that this maps the state to $U_s U_f|s\rangle$ Repeat steps 3-4 approximately $\sqrt{N}$ = t times, where N is the number of entries in the data set.

(e) Once the state of the system can be described as $|\psi_t\rangle = (U_s U_f)^t|s\rangle$. you measure the qubits, the corresponding amplitude will correspond to the equivalent classical bits of the target entry.

3. 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. first you initialize two n-qubit registers to 0

$$|\psi_1\rangle = |0\rangle^{\otimes n}|0\rangle^{\otimes n} \tag{3}$$

2. Apply a Hadamard transform to the first register

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle^{\otimes n} \tag{4}$$

3. Apply a query function Q_f

$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle \tag{5}$$

4. Measure second register, causing the first register to become

$$|\psi_4\rangle = \frac{1}{\sqrt{2}}\left(|x\rangle + |y\rangle\right) \tag{6}$$

5. Apply Hadamard transform to the first register

$$|\psi_5\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{z\in\{0,1\}^n} \left[(-1)^{x\cdot z} + (-1)^{y\cdot z}\right]|z\rangle \tag{7}$$

6. Measure the first register, which gives an output if

$$(-1)^{x\cdot z} = (-1)^{y\cdot z} \tag{8}$$

The output will correspond to a string z, this string will correspond to $b \cdot z = 0 \pmod 2$ which from Gaussian elimination can be used to find the XOR mask to the function f(x). This can be run exponentially faster than any equivalent classical algorithm.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

[11] 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.

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

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

[14] Huiqin Xie and Li Yang. Using bernstein–vazirani algorithm to attack block ciphers. *Designs, Codes and Cryptography*, 87(5):1161–1182, 2019.