Post Quantum Cryptography and its Applications to Bitcoin and Other Cryptocurrencies

Reid Bixler
rmb3yz@virginia.edu

David Evans
evans@virginia.edu

Abstract

Bitcoin was initially created as a means to form an electronic payment system based off of cryptographic proof instead of trust between two parties. The security behind this currency is paramount, because a violation of it would change its inherent value and use. With the introduction of practical quantum computing, the security of Bitcoin will weaken, thus requiring changes to the protocol. The goal of this technical capstone research is the creation of a quantum secure modification to Bitcoin.

1 Introduction

Bitcoin is a decentralized currency relying on a peer-to-peer network to track transactions performed with the currency, very similar to a public ledger available to anybody who wishes to join into the community (Babaioff et al., 2012). This public ledger, otherwise known as the blockchain, offers fast communication in a peer-to-peer network, easy access tot he public, an open-source implementation for modification, and complete transparency of all transactions on the chain (Twesige, 2015). Bitcoin's function as an 'algorithmic' currency is strictly enforced with cryptographic rules in a decentralized manner as opposed to fiat currencies whose value relies on public belief in a nation's government to not oversaturate the market with banknotes (Yermack, 2013). Specifically, Bitcoin and many other cryptocurrency protocols rely on the use of the Elliptic Curve Digital Signature Algorithm (ECDSA) for the public and private keys used throughout the exchange of these currencies (Crossen, 2015). These public and private keys are used to prove ownership of Bitcoins and to transfer the ownership to others through the block chain with signatures generated by these keys. However, ECDSA is strictly used for the exchange of Bitcoins whereas hashing (e.g. SHA256, RIPEMD160), which use one-way functions to create hashes of text, are actually used to create new Bitcoin. (Nakamoto, 2008)

Quantum computing is slowly gaining traction since it was theorized in 1982 by Richard Feynman. It relies on quantum principles and super positioning of particles to perform computational operations significantly faster than classical computers (Feynman, 1982). There already exist quantum algorithms, which use quantum logic gates to solve problems that were once thought computationally unsolvable. One of these algorithms, Shor's Algorithm, is able to easily determine the prime factors of a large number (i.e. integer factorization) within a polynomial time (Shor, 1999). Before this algorithm, integer factorization was considered non-polynomial, also known as NP, such that given a NP problem, one could verify its solutions quickly but it could not be solved within a reasonable time, often taking longer than the expected lifespan of the universe to solve (Ladner, 1975). This is a major problem for the security of many of today's technologies which rely on the hardness of prime factorization.

While quantum computers are only in their infantile stage, the potential security risks behind broken encryption schemes from quantum algorithms would be unfathomable in today's world. If anybody were able to obtain a quantum computer capable of performing significant computations, breaking into many secure systems would be completely feasible. As mentioned earlier, Bitcoin and many other cryptocurrency protocols use ECDSA for the public and private keys to prove ownership of Bitcoins. ECDSA is reliant on elliptic curve cryptography, which requires low computing power for implementation but massive amounts to brute force an attempt to break it. However, a modified Shor's algorithm could notably decrease the computational requirement for a

brute force attempt (Sullivan, 2013). With a working quantum computer capable of significant computation, any Bitcoin addresses that have leftover Bitcoin after a public transaction could be attacked and have their remainder stolen (Pacia, 2014).

2 Alternative Signature Schemes

Digital signatures are used pretty much everywhere on the internet, specifically for the means of Authentication, Integrity, and Non-repudiation. There exist many digital signature schemes that are quantum resistant, but there are some obvious benefits and drawbacks to each. (Buchmann et al., 2004) In this section, we analyze the different types of signatures that could be substituted into Bitcoin to formulate which one we plan to use for the final test implementation.

2.1 Lattice Signatures

A majority of current post-quantum cryptography is heavily reliant on lattices because of the high amount of security obtained while still having much smaller keys and signing/verification times.

Ring Learning with Error is based on the arithmetic of polynomials with coefficients from a finite field. In 2011, Lyubashevsky came up with the application of the Ring-LWE structure for signatures based off of the worst-case hardness of the Shortest Independent Vectors Problem (SIVP) in general lattices. (Lyubashevsky, 2011) In his paper, he looks at the Small Integer Solution (SIS) problem and its application to a signature scheme based off of its provable hardness. The requirements behind the signature are a secret key of $m*k$ matrix of S random integers and a signer must pick an m-dimensional vector y from a distribution D which then is used to compute c such that $c = H(Ay, m)$ which is then used to compute z such that $z = Sc+y$. The signature that the signer will output is (z, c), put there is a probability that a signature will not be outputted, at which point the signer will run the algorithm until a signature is outputted. The signature length and key size can change based off of the signers choosing, but there are certain security implications if those values are too small. Later in the paper, applications of SIS and LWE define a Ring Variant of a SIS signature which is provably improved from the previous two signature schemes. Specifically, the scheme defines a secret key $s_1, \ldots, s_v$ where every coefficient of s_i is chosen uniformly and independently from a predefined range. A public key is created $(a_1 \ldots, a_v, t)$ where each a_i is uniformly random and t is defined as the sum of each multiple of a_i and s_i. A message is signed, which can be verified easily with the public key. This scheme is based on the provable hardness of the ring variant of the SIS problem. This final ring variant is the Ring-LWE Signature provided by Lyubashevsky.

The Goldreich-Goldwasser-Halevi signature scheme was published in 1997 and is based on the Closest Vector Problem (CVP) which is also a lattice based problem (Goldreich et al., 1997). Signers show their message using a lattice, which the verifier then verifies on the different lattice point sufficiently close to the previously designated message point of the same lattice. The GGH signature scheme is more based on an encryption algorithm but the application to signatures is perfectly viable, while difficult.

NTRUSign is based on the GGH signature scheme which was put into publication in 2003, but was revised with parameter recommendations in 2005 (Hoffstein et al., 2005). This signature scheme uses 2 short polynomials in a ring which are then used to place the intended message in a $2N$-dimensional space. One of the problems presented with NTRUSign is the fact that a transcript of signatures can leak information about the private key. After the creators reformed their parameter recommendations in 2005, they suggested that 20^{30} signatures were needed to possibly find the private key, however a recent 2012 analysis of NTRUSign proved that they only needed about a couple thousand signatures to provably find the private key.

In 2013 the BLISS protocol was introduced as a solution to all the previously broken signature schemes introduced (Ducas et al., 2013). BLISS is very similar to Ring-LWE in structure, but has some notable changes which allow for it to (so far) not give away the secret key over consistent usage. However, there were optimizations introduced later in the publication, specifically defining the BLISS signature scheme. Using the BLISS signature scheme for future signatures look to prove quite useful, however compared to current schemes there are noticeable drawbacks. The BLISS scheme noticeably takes more space than RSA or ECDSA, but this is the most applicable and secure implementation of lattice-based cryptography so far. Since that paper has been re-

leased, 2 more modifications have been released which offered improvements in speed and cache protection brought up in BLISS-B and BLZZRD respectively (Ducas, 2014) & (Saarinen, 2017)

2.2 Multivariate Signatures

This type of cryptography is based off of multivariate polynomials over a finite field. One of the benefits of multivariate cryptography is the fact that it is extremely useful in the application of signatures, but all encryption schemes using multivariate cryptography have failed. One of the significant drawbacks of multivariate signatures is the size of the keys, which for the case of Bitcoin, essentially prevents us from using it at all in our desired protocol. However, current research in multivariate signatures includes Cyclic Rainbow and MI-T-HFE (Petzoldt et al., 2010) & (Zhang and Tan, 2015).

2.3 Hash-Based Signatures

Hash signatures have been around since around 1970 ever since Ralph Merkle came up with the idea of the Merkle Trees as an alternative to RSA and DSA. (Becker, 2008)The interesting thing about hash signatures are that the use of hashing is already widely used, but has been proven to be quantum resistant unlike some of the other current encryption methods. This means that using hash-based signatures could prove easier to implement into future technologies simply because people are more used to them. Currently the two general signature schemes are Lamport-Diffie as a single-signing hash or a Merkle-tree, which could be made up of any hashing scheme. (Preneel, 2005) However, while hashing is useful and already widely implemented, there has not been much recent research on hash-based signature schemes which suggests that there may be better alternatives.

2.4 Code-Based Signatures

Code-Based encryption schemes are based on the general hardness of decoding a general linear code. McEliece and Niederreiter encryption algorithms were discovered in the late 20th century, but the problem with code-based signatures is there will be very large key sizes required. (Cayrel and Meziani, 2010) Because of this problem, the code-based cryptography has only seen potential application to encryption schemes and not as much into signature schemes. Similarly to Multivariate signatures, the large key sizes significantly decreases the usability within Bitcoin, which heavily relies on having as small keys as possible.

2.5 Signatures Conclusion

There exist quite a number of signature schemes that have proven to be quantum resistant. In terms of applications, the most likely forms we may use will be Lattice-Based signature schemes. However, the biggest problem that we will see in the implementation of quantum resistant signatures are the large key sizes in comparison to current key sizes. This means that many protocols will need to be changed to work with the new signature schemes, as introduced in the suggestion of Transport Layer Security (TLS) and Internet Key Exchange (IKE).

After analyzing the possible scheme alternatives, we decided upon initially using the BLISS scheme for our post-quantum implementation of Bitcoin due to its smaller key sizes compared to most quantum secure signature schemes as well as the very fast signing and verification times. However, BLISS has been thoroughly researched since its inception in 2013, and there have been potential security flaws that have been fixed through modifications in BLISS-B and BLZZRD. After deciding between these two, we decided upon sticking with the most recent BLZZRD due to the cache-protection and C-implemented software implementation available through the original paper.

3 Risks of the Current Protocol

As stated previously, the ECDSA algorithm is no longer NP-hard using a modification of Shors algorithm. This means that any service currently using the ECDSA algorithm has the potential to be hacked or broken by a quantum computer. In terms of the Bitcoin protocol, what would be the consequences in the breaking of the ECDSA algorithm? The Bitcoin protocol uses ECDSA to generate public and private key pairs which are used to sign transactions and verifying those same transactions. The private key is specifically used to sign any transaction that you choose to put onto the block chain. This means that you can create a transaction designated that you will send b Bitcoin to person P, followed by signing a signature of that transaction with your private key K_r (Buterin, 2013). The Bitcoin network in turn sees that

transaction and verifies your given signature with your publically available public key K_u. As long as the signature can be verified with the public key, then the transaction will most likely go through and onto the block chain. The security of ECDSA relies on the fact that it is provably hard to calculate the private key K_r from the available public key K_u. However, a functioning quantum computer would be able to relatively easily derive the private key from any given public key (Chen et al., 2016). This control of a private key does not give total control over the users Bitcoins, but it does pose risks for some users. Specifically, this poses a risk to users who choose to only stick with one private key, as in users that use the same Bitcoin address repeatedly. However, the Bitcoin protocol protects your Bitcoin address with the usage of multiple hash functions used on the public key to generate the users Bitcoin address. (Howe et al., 2015)

What this generally means for Bitcoin users is that your address will not be the source of your leaked private key (assuming a functioning quantum computer). So all that means is that we dont have to give away our public key? The problem though is that nobody can verify your transaction signatures without your publically broadcasted public key. This means that until you broadcast your public key, your address is perfectly safe from quantum attacks (assuming that the hashing functions SHA256 and RIPEMD-160 remain quantum resistant). However, once your public key is broadcast to verify a transaction, any leftover Bitcoin in that address is at risk of being attacked and robbed by a quantum computer. Once your transaction and public key are available, a maligned entity with a quantum computer can brute force your private key, impersonate you with it, create a transaction transferring any leftover Bitcoin to their own address, and get off scot-free (assuming the Bitcoin community would be unwilling to reverse the transaction). So what is a solution to this? Simply make every Bitcoin address a one-time use address. While this principle is carried out quite often in the Bitcoin block chain, there are still many cases of businesses, organizations, and the like who reuse a Bitcoin address for ease of use. In fact, there exist many Bitcoin addresses on the block chain that are holding hundreds of Bitcoin and have already publically broadcast their public key.

However, there is yet another problem present with the introduction of a quantum computer. The Bitcoin community is built up from a multitude of nodes throughout the world who receive and send Bitcoin transactions to the Bitcoin miners that eventually place your transactions in the block chain. If one of those nodes that you attempt to send to has a quantum computer, they could receive your intended transaction, forge your private key, create a new transaction changing the intended address, and send it on its way to the mining pools. Our previous solution assumes that every Bitcoin node and Bitcoin mining pool are trustworthy and do not have quantum computers. If any receiving node or mining pool is malicious and owns a functioning quantum computer, they can spend your Bitcoin even before your transaction becomes technically available to the public. While the majority of Bitcoin may be safe, any security risk to the protocol must be mitigated as soon as possible, assuming a future of working quantum computers.

4 Required Bitcoin Alterations

The current Bitcoin protocol is implemented in multiple languages but it also has its own Scripting language. (Antonopoulos, 2014) This Scripting language is a number of Opcodes which are *NOT* Turing complete because there is no need for the language to be more complex than it needs to be, but also because a non-Turing complete language wont have loops and therefore wont have the case of the Halting Problem, where it is simple to determine whether or not a program will finish or not, and the introduction of loops can make a program go on infinitely. If the Bitcoin Scripting language doesnt have loops, then there is a guarantee that no program in the Bitcoin software will get stuck on a script that loops on forever. In the past, there were a few Opcodes from the Bitcoin Scripting language that were removed for security reasons. For example, there used to be a simple `OP_RETURN` which had the potential to break the entire structure of Bitcoin because by passing it along with a 1, one could validate any transaction without the public key or signature (Obviously a security flaw in software intended to hold monetary value).

4.1 Script

How does the current Bitcoin Scripting Language (literally known as Script) work? It is a simple stack-based protocol that moves from left to right. What this means is that a user will give a Script as a list of instructions which will move items on and off of a stack. In terms of actual uses, if an Unlocking and Locking script are given, then a final stack with `True` on top (assuming a correct implementation of scripts) will grant access to transferred Bitcoin. The Script will contain, as stated before, opcodes, also known as `Words` which have the ability to manipulate the current stack. The words themselves look like `OP_NOP`, `OP_PUSHDATA1`, etc. but have numbers (opcodes) associated with them such as `97`, `76`, etc. respectively. There are some very complex checks of cryptography implemented into some of these words including public key verification with signatures (`OP_CHECKSIG`), RIPEMD-160 (`OP_RIPEMD160`) and SHA256 (`OP_SHA256`) hashing, and verifying multiple public keys with multiple signatures (`OP_CHECKMULTISIG`). These are the core of the scripting language and are mostly what define the 5 sets of valid Locking and Unlocking Scripts.

What are these Locking and Unlocking Scripts anyway? These are the scripts, that when in combination, validate transactions on the Bitcoin blockchain. The Locking Script is a combination of hashing, the public Bitcoin address, and checks for the signature (Specifically, `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`, in one case). This Locking Script is often called `scriptPubKey`. For each Locking Script, there must be a valid Unlocking script that combines the signature and public key of the sending party (For the previous example, `<sig> <pubKey>` are passed). This Unlocking Script is often called `scriptSig`. With these two scripts, transactions are able to be validated on the stack by verifying the signature and public key hash, and without those the transaction wont be validated. Specifically, the Locking script is a type of lock placed on the outputs of a transaction that wont allow the output to be spent without the verification of the key (i.e. Unlocking Script). Every Bitcoin client intending to verify all the transactions must take the given Locking Script on the inputs run the Unlocking Script (given with the input itself) and execute the scripts together. However, there is not just one way to run Locking and Unlocking Scripts, in fact there are multiple. The one referenced previously is known as a Pay-to-Public-Key-Hash (P2PKH) because it is paying to the hash of a public key. This is predominantly used in the blockchain, but there do exist other payment methods like Pay-to-Public-Key (P2PK) and Multi-Signature.

One thing of note is that P2PK will become completely invalidated if quantum computers become available. The reasoning behind this is that P2PK assumes that the outputs are locked with the Public Key (Like so, `<Public Key A> OP_CHECKSIG`) which in a quantum world would be easy to break and find the private key for. If somebody wants to steal money from this address they would have to simply give the signature from the private key (Like so, `<Signature from Private Key A>`) which would grant access to the stored Bitcoin. At least in the P2PKH instance, the Locking script runs on top of the Bitcoin address (a hashed version of the public key), which would make it hard to find the public key, and therefore hard to break and find the private key. Therefore, up until the Unlocking Script is run (and the Public Key is broadcasted by the intended recipient), any Bitcoin outputs will be safe from quantum-attack *assuming* that the Public Key has not yet been broadcasted.

4.2 BLZZRD Modifications

As mentioned previously, BLZZRD will be our desired post-quantum signature for the Bitcoin protocol. The question is how we can ensure that this signature is being verified alongside the original signature. We want to be able to create Scripts that simultaneously verify the ECDSA signature as well as the BLZZRD signature, and guarantee a failure if either fails (specifically the case when ECDSA succeeds and BLZZRD fails, which would be a sign of a quantum-attack). The largest problem right now is simply the fact that the Script does not yet contain Opcodes for verifying a BLZZRD signature. Until an Opcode for this verification is available, there can be no unlocking or locking scripts that can ensure security against quantum computers. However, in this case, I am going to move forward assuming that there exists a word known as `OP_CHECKBLZZRD` which when given a stack of `<PUB KEY BLZZRD> <SIG`

`BLZZRD>` will return a result of TRUE on the stack (essentially verifying that the public key shown is generated from the same private key of the signature).

We can now go about determining what would be the best course of action for creating a valid Unlocking and Locking Script with the given Opcode. Simply, we will be having our Unlocking script have both our Public Keys and Signatures from ECDSA and BLZZRD, and our Locking Script will have our verifications first of the ECDSA then of the BLISS, and finally comparing those two outputs to determine if they are both true (a simple AND instruction). With this in place, we can ensure that any transaction containing Locked with these Scripts will only be unlocked with a valid and quantum-secure BLISS signature.

4.3 Consequences of BLZZRD

With new Unlocking and Locking scripts, we introduce yet another form of possible transactions in Bitcoin. These transactions will be only useful assuming a BLZZRD private key can be generated and used by the users, which will require a change of software for many Bitcoin clients, assuming they desire the protection from quantum attack. In terms of previous addresses and previous transactions, we cannot end up blocking old Unlocking and Locking scripts, because then you would be deleting old Bitcoin by leaving them in addresses to never be used again. This means that we must be able to use the old Locking and Unlocking scripts, even if they are technically not secure from quantum attack if the transaction does not use change addresses. It will be up to the individual owners of Bitcoin to switch over to the new protocol, with the risk of not switching being the possibility of having your Bitcoin stolen.

In terms of actual requirements set by Bitcoin for a standard transaction, are as follows:

- The transaction must be smaller than 100,000 bytes. Thats around 200 times larger than a typical single-input, single-output P2PKH transaction.
- Each of the transactions signature scripts must be smaller than 1,650 bytes. Thats large enough to allow 15-of-15 multisig transactions in P2SH using compressed public keys.
- The transactions signature script must only push data to the script evaluation stack. It cannot push new opcodes, with the exception of opcodes which solely push data to the stack.

Assuming that we are using BLZZRD-IV, we are going to have a signature size of $6.5KB$, a private key size of $3KB$, and a public key size of $7KB$. This means that adding in any of the BLISS keys or signatures for a single transaction between two parties is completely doable, albeit limiting on the current protocol. As of now, the average single input to single output transaction will require 500 bytes or $0.5KB$. By adding on the public key and signature from BLISS, we will be requiring approximately $14KB$ per transaction (assuming single to single), this would include the 500 bytes from normal ECDSA, $6.5KB$ for the signature, and $7KB$ for the public key. At the current standard, this means that we will only be able to fit around 7 inputs and outputs to a transaction, assuming they are all using the BLZZRD protocol. This is a pretty huge jump in terms of size requirements for Bitcoin (200 to 7 is a factor of nearly 30). In terms of speed requirements, the only thing that we will be verifying is the signature of ECDSA and the signature of BLZZRD. The act of signing in BLZZRD-IV requires about $0.375ms$ while a ECDSA-256 requires about $0.106ms$. Luckily signing is not actually required by a node for a list of transactions, signing is only done by the individual users. Verification for BLZZRD-IV requires $0.032ms$ while a ECDSA-256 requires about $0.384ms$. This means that we can verify about 31,000 transactions in a second with BLZZRD-IV at the same time that we could verify about 2,500 in a second transactions with ECDSA-256. This is extremely useful when integrating into the Bitcoin protocol because then we can know that speed will not be an issue created from this integration. Notably the size will likely become an issue, requiring possibly an increase later in the future, especially considering that there are arguments going on currently about the maximum blocksize of 1MB being too small (only allowing for about 3-4 transactions per second).

5 The BLZZRD Bitcoin Protocol - BLZZCoin

There are currently two alternatives to implement BLZZRD into the Bitcoin protocol, each with their own benefits and drawbacks.

5.1 Burnt BLZZCoin

Technically, this protocol does not rely on BLZZRD, but in order to eventually have post-quantum security, this protocol would exist for intercompatibility issues. The security of this method relies on the fact that hashing will not be easily broken with quantum computers. Technically, anybody can create a random address that passes the checks of a valid address, but actually doesnt hash from a ECDSA public key. You could put together a string of 0s, a really interesting vanity address that obviously would be impossible to find (ex. `1YouWontBeAbleToGetThisAddress...`), or hash some made up public key. But for our case, we want to actually be able to use this Bitcoin once the Bitcoin Community (hopefully) switches over to the BLZZRD protocol. For this we can generate a BLZZRD private and public key, hash our public key, and use that hash just like any other address in our Locking Script.

Burnt BLZZCoin Locking Script:

```
OP_DUP
OP_HASH160
<BlzzrdkeyPubHash>
OP_EQUALVERIFY
OP_CHECKSIG
```

Burnt BLZZCoin Unlocking Script:

```
<BlzzrdSig>
<BlzzrdPubK>
```

Assuming that we only want to transfer our own coin into a locked BLZZRD address, we simply take that address and send money to it. The only way that this address could ever spend money are one of two ways.

1. The Bitcoin Community alters the `OP_CHECKSIG` opcode to allow BLZZRD signature checking. The reason why we cant have a `OP_CHECKBLZZRD` in this case is it would require a fork beforehand, which this method is not relying on. It would be fairly trivial to alter the `OP_CHECKSIG` code to check the length of the signature given to verify if an ECDSA or BLZZRD signature was used.

2. **Assuming** that the Public Hash is a valid address (i.e. starting with 1 or 3, indicating P2PKH and P2SH respectively), however unlikely considering a BLZZRD Public Key Hash would likely not have the same starting bytes as an ECDSA public key hash. Somebody breaks the RIPEMD hash algorithm and produces a hash collision with an ECDSA public key which can be used to validate the transaction.

Benefits:

- This can be done at anytime, even before the Bitcoin Community considers switching to BLZZRD
- Guaranteed security (assuming no hash-break) against quantum attack
- Usable once BLZZRD becomes integrated and requires minimal changes in the protocol

Drawbacks:

- **HIGH POSSIBILITY OF LOSING YOUR BITCOIN**. Obviously this is a huge risk to put all of your faith into the assumption that the Bitcoin Community MAY end up integrating this in this exact way. Even if the Bitcoin Community decides to switch to BLZZRD, but does it another way, then your Bitcoin would still be lost!
- Requires altering an opcode, likely increasing the chance of security flaws. Adding an additional opcode like `OP_CHECKBLZZRD` would be great, but does not allow users of Bitcoin to protect their Bitcoin before the fact.
- If the Bitcoin Community chooses not to do this, then you are just adding an additional transaction on top of the blockchain that is totally useless and takes up unnecessary space, and also permanently locking Bitcoin into the blockchain.

Considerations: This is a very drastic measure for protecting your Bitcoin with BLZZRD, and there are entirely more reasonable methods to protect your Bitcoin, most notably, just putting your Bitcoin into an address that has not been used before AND not spending any of those coins from that address. Assuming that hashing isnt broken, your ECDSA public key does not need to be published to the blockchain until you want to spend coins from an address, therefore protecting

your Bitcoin from quantum attack until the Bitcoin Community implements a more reasonable protection. The only reason that you would prefer Burnt BLZZRD over this is if you think that you REALLY want to guarantee that a quantum attack wont steal your Bitcoin AND you want to leave Bitcoin in this address even after this method became implemented (which is highly unlikely). An additional consideration is that the Bitcoin Community could collectively agree on a protocol before implementation, allowing for the more paranoid users to quickly lock their Bitcoin before a software change gets implemented. This would be a two-step process allowing for better quantum-security while minimizing the amount of wasted and locked Bitcoin.

5.2 Scripted BLZZCoin

The assumption going forward is that a Redeem-Script can have a currently Reserved word/opcode without making it an invalid transaction. To explain why this assumption is necessary, we need to look at the Pay to Script Hash (P2SH). P2SH was added late into the development of Bitcoin to allow the recipients to create their own ways to ensure the security of the Bitcoin that they were receiving. The cool thing about P2SH is that the requirements to unlock funds could be a number of signatures, a password, or even the answer to a complicated math problem. In our case, the necessary key will be our BLZZRD Public Key. However, the main component of our method will be the fact that our script can contain a currently invalid opcode. Technically, as of right now, if a node sees that a transaction is invalid then they will toss it aside and ignore it. However, if the invalid opcode is within a hash, then technically the transaction is valid and becomes truly valid once the opcode becomes valid itself. This method is similar to our Burnt BLZZRD, but has a very interesting benefit of not requiring the alteration of `OP_CHECKSIG`.

Scripted BLZZCoin Redeem Script:

```
<BlzzrdPubK>
OP_EQUALVERIFY
OP_RESERVED1
```

Scripted BLZZCoin Locking Script:

```
OP_HASH160
<20-byte hash of Redeem Script>
OP_EQUAL
```

Scripted BLZZCoin Unlocking Script:

```
<BlzzrdSig>
<Redeem Script>
```

Assuming once again that `OP_RESERVED1` is technically valid as long as it is hashed, then this Bitcoin will be locked into the script address given by the specific `<BlzzrdPubK>` and cannot be unlocked until `OP_RESERVED1` becomes a valid opcode. This is similar to our previous Burnt BLZZRD but has the addition of not requiring the replacement of code in `OP_CHECKSIG`, instead we will rely on the Bitcoin Community to put our assumed `OP_CHECKBLZZRD` into the `OP_RESERVED1`. Once `OP_CHECKBLZZRD` is implemented, our previously locked address becomes spendable with our BLZZRD Signature. As far as my research has found, you must ensure that adding this new opcode doesnt cause transactions that were once invalid to be valid. I am unsure if this means if a P2SH Redeem Script could or could not have an invalid opcode in it. From what I grasp, it means that an unaccepted and invalid transaction from before cannot now become valid transactions. Technically, this method is a completely valid transaction, since the Redeem Script is never publicly released beforehand.

Benefits:

- Similarly to Burnt BLZZCoin, this can be done at anytime, even before the Bitcoin Community considers switching to BLZZRD
- Guaranteed security (assuming no hash-break) against quantum attack
- Usable once BLZZRD becomes integrated
- Doesnt require altering current opcodes

Drawbacks:

- **HIGH POSSIBILITY OF LOSING YOUR BITCOIN**. Obviously this is a huge risk to put all of your faith into the assumption that the Bitcoin Community MAY end up integrating this in this exact way. Even if the Bitcoin Community decides to switch to BLZZRD, but does it another way, then your Bitcoin would still be lost!
- Requires taking up a reserved opcode. Adding an additional opcode like

OP_CHECKBLISS is extremely useful, but seeing as reserved opcodes are rare, it may be risky to give one up for the BLZZRD protocol

- If the Bitcoin Community chooses not to do this, then you are just adding an additional transaction on top of the blockchain that is totally useless and takes up unnecessary space, and also permanently locking Bitcoin into the blockchain.

Considerations: Similarly to the other burnt coin, this is a very drastic measure for protecting your Bitcoin with BLISS. Admittedly, compared to Burnt BLZZRD, this method is more likely to be accepted because there would not be alterations to current opcodes. Depending on if the Bitcoin Community wants more versatility with an opcode by changing OP_CHECKSIG or adding a new opcode over a reserved slot with OP_CHECKBLZZRD will determine what path is best.

5.3 Slow BLZZCoin

Technically, we will not be able to be able to 100% protect our Bitcoin from quantum attack unless we never spend from our final output address (by this I mean you can never have leftover Bitcoin in an address after a transaction, else you risk having it stolen from quantum attack) or use the previously mentioned burn tactics. Obviously, a lot of people still want to be able to constantly use their Bitcoin throughout, and nobody really would like to take the risk of burning their Bitcoin (or at least a majority of it). I suggest a method that allows for the Bitcoin community to slowly integrate BLZZRD into the protocol, admittedly at the cost of speed and immediate security.

Step 1) Message BLZZCoin: A simple way to protect our Bitcoin from quantum attack would be to tell the Bitcoin not to accept transactions from an address without verifying a BLZZRD key. To do this, we can use the 40 additional bytes that can be tacked onto a transaction with OP_RETURN.

Our Locking Script will look like any ordinary P2PKH with OP_RETURN on top:

```
OP_DUP
OP_HASH160
<PubKHash>
OP_EQUALVERIFY
OP_CHECKSIG
OP_RETURN
<Message>
```

Our Unlocking Script will be the normal P2PKH script:

```
<Sig>
<PubK>
```

Our <Message> will contain something along the lines of Please dont accept this transaction without verifying my BLISS public key available at ". This is obviously unlikely to stop many people from accepting the transaction, but in the worst case if your Bitcoin were stolen from quantum attack, then you could 'prove' that the transaction should be invalidated. However, given the Bitcoin community's previous stagnancy on reverting transactions, this will likely provide little protection.

Step 2) Forked BLZZCoin: At this point, we can wait until the Bitcoin Community decides to do a soft or hard fork with BLZZRD included. We cannot assume in this case how they will do it, but the previous two possibilities of burning can be used for this new BLZZRD protection. In this case we have 2 options, replacing a reserved opcode for OP_CHECKBLISS or altering OP_CHECKSIG to allow for BLZZRD signatures. Once this is added, we have the opportunity to replace any P2PKH or P2SH with BLZZRD signatures and BLZZRD public keys. By waiting until a fork happens, we take away any risk that your Bitcoin ends up getting burnt through assumptions in changes of the protocol.

Benefits:

- Can still spend Bitcoin before forking to BLZZRD
- Guaranteed security once Step 2 is achieved
- Very low possibility of losing Bitcoin. Cannot lose Bitcoin through burnt addresses and can only lose it assuming a quantum attack happens before Step 2.
- Step 1 doesnt require altering current opcodes

Drawbacks:

- This requires the Bitcoin Community to eventually fork and make the BLZZRD changes necessary. Until Step 2, your Bitcoin are not guaranteed to be secure with a simple message.

- Taking the risk that your address could be the reason why Bitcoin shifts to BLZZRD. By this, when a quantum computer user decides to attack a Bitcoin address, there will likely be some amount of Bitcoins stolen until the blockchain forks to the BLZZRD or another protocol. If your address has enough Bitcoin to warrant being attacked by a quantum computer, then you are more likely to lose your Bitcoin before the necessary changes take effect.

- Still requires changes to current code or a reserved opcode. Obviously it wont be possible to allow BLZZRD without doing some alteration to the Bitcoin protocol, and it is up to the community to decide which choice is best.

Considerations: One consideration for this method is whether or not the Bitcoin Community should decide to phase out ECDSA entirely. The obvious drawback of phasing out ECDSA is that anybody with old Bitcoin will no longer be able to use them. Even setting some period of time to switch over will still leave some people with Bitcoin stuck in addresses no longer accepted. Essentially, this would be a collective decision by the community to burn a portion of the entire Bitcoin into unspendable addresses. However, if ECDSA is used alongside BLZZRD then we need to start to consider whether or not they will be used together, alternatively, or through some other means. For example, should new standard Locking and Unlocking scripts assume that both ECDSA and BLZZRD are going to be included, or the default be BLZZRD with ECDSA still being available? By maintaining ECDSA, no address will ever be locked out of using their Bitcoin, but they do end up taking the risk of having their Bitcoin stolen in the case of a quantum attack. Also, we need to consider the size of BLZZRD keys and signatures. The limits on transactions are drastically decreased once BLZZRD is added in, and the blockchain will only grow even larger. It makes sense to increase the standard block size from 1MB to something larger, but that is another major problem being currently debated today. Suffice to say, even if BLZZRD secures Bitcoin in technicality, it poses other social and economic problems within Bitcoin.

6 Golang Implementation

This paper also offers an implementation written in Golang for both the BLZZRD packages as well as modifications to the Bitcoin protocol, in this case, the `btcd` implementation (`https://github.com/btcsuite/btcd`). We have decided to go with Golang instead of the native C implementation because of a few reasons. Firstly, there already existed a C implementation of BLZZRD and we thought it would be useful to have an alternative language implementation of it. Secondly, while C is extremely fast, it can be insecure if not coded properly and can often be difficult for people to interpret quickly. Finally, we decided on Golang because of the stable implementation available through `btcd` that works very well and makes for understanding the Bitcoin protocol a bit easier than the core C implementation. The intention of this paper and research was to offer up a potential modification that could technically be feasible in Bitcoin's protocol, not to have the most streamlined and fastest implementation possible. Obviously there is room to speed-up and slim-down our implementation either by cleaning up the current one, or reimplementing it in another language. Our Golang implementation is a transcribed and translated Go version of the C implemented BLZZRD available at `https://github.com/mjosaarinen/blzzrd`. There are some obvious changes between C and Golang, possibly resulting in unintended changes, but none of which have been seen so far in testing. Similarly with the modifications to `btcd`, there may be overhead changes that technically aren't valid on the real blockchain for Bitcoin since all of the tests done were either on the testnet or the simnet provided with the implementation. The translated Go version of BLZZRD and modified Bitcoin protocol are available at urlhttps://github.com/ReidBix/CapstoneResearch.

6.1 Other Implementations

Towards the end of my research, we stumbled upon a very recent article released in late March on a Quantum Resistant Ledger (Coleman, 2017). Interestingly, this is a take on using Lamport-Diffie, as mentioned previously, which relies on hashing as the protection against quantum-attack. Also, on another note, this protocol is essentially would become entirely different cryptocurrency if they chose to build off of their blockchain. The

author has created this blockchain not explicitly for the use of cryptocurrency, and rather is interested in the benefits of the blockchain as a public ledger (Waterland, 2016). As seen from our own research, their keys are significantly larger than that of ECDSA, but also signatures can only be used once safely, which is often not the use case with Bitcoin. This is a very intriguing avenue compared to our own, seeing as we decided to go with lattice-based cryptography as opposed to hash-based. There are potential research possibilities to analyze the differences between the two and possibly see to how these can be combined for a more secure and faster implementation.

7 Conclusion

Protecting Bitcoin from potential attacks of quantum computers may seem like overkill, but when the foundation and value of Bitcoin is reliant on the security of its algorithms, we need to guarantee that those algorithms get replaced if proven to be insecure. Looking through alternative signature schemes, we decided to go with the Bimodal Lattice Signature Scheme (BLISS) with modifications for speed-ups and cache protection (BLZZRD) as our intended post-quantum algorithm replacement for ECDSA. We have analyzed the necessary changes required from Bitcoin, the effects that it will have on the blockchain, and different steps to integrate BLZZRD into Bitcoin. Implementing these changes will definitely be considered drastic and have lasting effects on Bitcoin and the blockchain, which is why so much effort has been put into analyzing such effects. We have gone about offering a software implementation of this improvement in security.

Future Research: There are a number of different research vectors to go about from here. Finding an alternative quantum signature scheme that performs better than BLZZRD would prove to be useful in a future implementation in the Bitcoin protocol. Alternatively, improving BLZZRD and its implementation could result in more speed-up and smaller key sizes. Another venue for research would be finding alternative schemes for integrating a quantum-secure algorithm into Bitcoin without the risk of burning any Bitcoin. Finally, further analyzing the security and implementation of BLZZCoin would decrease potential flaws and provide for a better foundation for the Bitcoin community to go from.

Acknowledgments: I am thankful to Dave Evans for being my technical research advisor throughout this research process and working with me on this during my 4th year. He has been of great help and always was there whenever I seemed to be looking for a solution.

References

A M. Antonopoulos 2014. Mastering Bitcoin: Unlocking Digital Cryptocurrencies. *O'Reilly Media, Inc.*

A Petzoldt, S Bulygin, and J Buchmann. 2010. CyclicRainbowa multivariate signature scheme with a partially cyclic public key. *In International Conference on Cryptology in India*

B Preneel. 2005. Design Principles for Hash Functions Revisited

C Pacia. 2014. Bitcoin vs. the NSAs quantum computer *Bitcoin not Bombs*

D Yermack. 2013. Is Bitcoin a Real Currency? An Economic Approach *National Buereau of Economic Research*

G Becker. 2008. Merkle signature schemes, merkle trees and their cryptanalysis *Ruhr-University Bochum, Tech. Rep.*

J A. Buchmann et al. 2004. Post-Quantum Signatures *IACR Cryptology ePrint Archive*

J Hoffstein, N Howgrave-Graham, J Pipher, J H. Silverman, and W Whyte. 2005. NTRUSIGN: Digital signatures using the NTRU lattice *In Cryptographers Track at the RSA Conference*

J Howe, T Pppelmann, M O'Neill, E O'Sullivan, and T Gneysu 2015. Practical lattice-based digital signature schemes. *ACM Transactions on Embedded Computing Systems (TECS)*

L Chen, S Jordan, Y K. Liu, D Moody, R Peralta, R Perlner, D Smith-Tone 2016. Report on post-quantum cryptography. *National Institute of Standards and Technology Internal Report*

L Coleman. 2017. Quantum Resistant Ledger Readies For Battle Against Quantum Computing, Hires Testers And Seeks Feedback. *Hacked: Hacking Finance*

L Ducas, A Durmus, T Lepoint, and V Lyubashevsky. 2013. Lattice signatures and bimodal gaussians. *In Advances in CryptologyCRYPTO 2013*

L Ducas. 2014. Accelerating Bliss: the geometry of ternary. polynomials *IACR Cryptology ePrint Archive*

L Twesige. 2015. Bitcoin. A Simple Explanation of Bitcoin and Block Chain Technology

M Babaioff, S Dobzinski, S Oren, and A Zohar. 2012. On Bitcoin and Red Balloons *In Proceedings of the 13th ACM Conference on Electronic Commerce* ACM

M J. O. Saarinen. 2017. Arithmetic coding and blinding countermeasures for lattice signatures *Journal of Cryptographic Engineering*

N Sullivan 2013. A (relatively easy to understand) primer on elliptic curve cryptography. *Ars Technica*

O Goldreich, S Goldwasser, and S Halevi. 1997. Public-key cryptosystems from lattice reduction problems. *In Advances in Cryptology-CRYPTO'97: 17th Annual International Cryptology Conference*

P L. Cayrel and M Meziani 2010. Post-quantum cryptography: Code-based signatures. *In Advances in Computer Science and Information Technology*

P Waterland 2016. Quantum Resistant Ledger (QRL). *theqrl.org*

P W. Shor 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303-332.

R E. Ladner 1975. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155-171.

R P. Feynman 1982. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21(6):467-488.

S Crossen. 2015. The Mathematics of Bitcoin.(Doctoral dissertation) *Emporia State University The Department of Mathematics*

S Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system

V Buterin. 2013. Bitcoin Is Not Quantum-Safe, And How We Can Fix It When Needed. *Bitcoin Magazine*

V Lyubashevsky. 2011. Lattice signatures without trapdoors. *In Annual International Conference on the Theory and Applications of Cryptographic Techniques*

W Zhang and C H. Tan. 2015. MI-T-HFE, a new multivariate signature scheme. *In IMA International Conference on Cryptography and Coding*