Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Making Bitcoin Quantum Resistant

Author:
Dragos Ilie (dii14)

Supervisor:
Prof. William Knottenbelt

Second Marker:
Prof. Kin Leung

June 18, 2018

Abstract

Quantum computers are expected to have a dramatic impact on numerous fields, due to their anticipated ability to solve classes of mathematical problems much more efficiently than their classical counterparts. This particularly affects cryptographic systems involving integer factorisation and discrete logarithms, such as the Rivest-ShamirAdleman or Elliptic Curve cryptosystems.

Bitcoin is a decentralised digital currency system, which was introduced by the pseudonymous Satoshi Nakamoto in 2008. Its conception was enabled by the use of two cryptographic primitives: hash functions and public-key cryptography.

In this paper we consider the threats a quantum-capable adversary could impose on Bitcoin, which currently uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign transactions and ensure immutability.

We then propose a simple protocol update, structured as a commit–delay–reveal scheme, which allows users to securely move funds secured by the now vulnerable ECDSA into a quantum-resistant digital signature scheme. The transition protocol functions even if ECDSA has already been compromised. While our scheme requires modifications to the Bitcoin protocol, these can be implemented as a soft fork.

Acknowledgements

I would like to start by thanking Research Associate Iain Stewart of the Cryptocurrency Research Centre at Imperial College London, for numerous clarifications and discussions, without which this work would not have been possible. He consistently contributed to deepen my understanding on the subject while allowing me to focus on whichever aspect of the work I wanted.

I would also like to express my sincere gratitude to my thesis supervisor Prof. William Knottenbelt for his continuous support, for pushing me to publish the theoretical part of this work in the Royal Society Open Science Journal, and most importantly, for encouraging me to pursue a PhD. degree. His availability in helping me was really far beyond expectations.

Furthermore, I would like to thank PhD. students Alexei Zamyatin and Sam Werner with whom I have co-authored the aforementioned journal paper. Together, we gracefully overcame the various challenges encountered, thus producing an elegant solution.

Finally, I must convey my absolute appreciation to my parents and girlfriend for their inexhaustible support and unceasing encouragements throughout this venture.

Contents

Section 1: Introduction

In this paper we approach the issue of integrating Bitcoin in a post-quantum world. One of the aspects, for which Bitcoin is regularly praised is the security guarantees it offers. However, with recent developments in the quantum computing research area, it appears that one of the core cryptographic primitives, on which the whole security model relies, is highly vulnerable. However, the system was designed in such a modular and extensible fashion, that it can be smoothly recovered with little modifications to the existing system.

In this paper we offer a protocol update called QRWit (QuantumResistantWitness), as inspired by SegWit [40]. We propose a three stage, commit–delay–reveal, scheme that allows Bitcoin users to safely spend their non-quantum-resistant funds and secure them under a quantum-resistant cryptosystem.

1.1 Problem and Motivation

Bitcoin was introduced by the pseudonymous Satoshi Nakamoto in 2008 [47]. It can be described as a set of standards, ideologies, protocols and technologies that form the basis of a decentralized digital currency system. The units of digital money are called bitcoins and their ownership can be transferred to other participants through transactions. Transactions are signed by the sender using ECDSA, an algorithm based on the elliptic curve public-key cryptosystem. Users communicate to each other, broadcasting transactions, via a peer-to-peer network characterised by the lack of a central authority governing the state of the system. Participants maintain a list of all historic transactions, aggregated in blocks, in a distributed public ledger called the blockchain. Blocks are linked via the hashes of their predecessors, thereby providing strong guarantees for the immutability of the transaction history. Miners, a subset of the participants, validate transactions and solve a computationally expensive puzzle called Proof-of-Work (PoW) to create blocks. For their services, they are rewarded with fresh (minted) units of the underlying cryptocurrency and with transaction fees. The dynamically changing set of pseudonymous participants establishes agreement on the current state of the system by considering only the longest chain of blocks (main chain), thus monetizing mining only on top of the main chain.

Quantum computers (QCs) theoretically appeared about 40 years ago, but relatively recent breakthroughs have placed the idea in the public eye once again. One such breakthrough, with a direct impact on Bitcoin's security, is Peter Shor's polynomial time quantum algorithm [60], which in its subsequently generalised form can break ECDSA. Although the early generations are not scalable enough to affect Bitcoin, various alternatives for the architecture of QCs are being considered, tested

and implemented [69, 22, 68]. As more entities enter this growing research area, it seems increasingly likely that powerful QCs will emerge in the near future. A sudden improvement in the approach towards scaling might lead to a powerful QC appearing virtually overnight.

To this end, the Bitcoin community must be prepared to transition to a quantum-resistant signature scheme. Post-quantum cryptography is an area of research with substantial history and many different approaches are still being pursued and considered. Although such cryptosystems exist theoretically, not many satisfy the low-bandwidth, space efficiency, and scalability requirements of Bitcoin. Furthermore, even if a suitable quantum-resistant signature scheme is found and deployed in Bitcoin, the issue of safely transferring ECDSA protected funds to the new signature scheme remains. In particular, the scenario in which the transition has to be done in the presence of quantum-capable adversaries must be taken into consideration and solutions to overcome this problem must be developed as soon as possible.

QRWit comes as response to exactly this issue. Moreover, while we appreciate many Bitcoin believers are confident that QCs will not affect Bitcoin any time soon, the need for QRWit remains as ECDSA secured funds should be recoverable even in the distant future when QCs will become everyday gadgets. As a digital currency system, Bitcoin has to guarantee backwards compatibility such that users who leave their funds untouched for extended periods of time can still recover them at any point they wish.

1.2 Contribution

On these grounds, a protocol that allows Bitcoin users to safely consume funds backed by ECDSA even in the presence of a quantum-capable attacker, should be proposed. Consequently, we propose QRWit, a commit–delay–reveal protocol for the secure transition from Bitcoin's current signature scheme to a quantum-resistant analogue, applicable even if ECDSA has already been compromised. Independent of quantum computing, QRWit can be generally applied to react to the appearance of any other vulnerabilities rooted in Bitcoin's public-key cryptography. Furthermore, unlike other proposals, we emphasise the necessity of a substantial delay phase to provide strong guarantees against adversarial chain reorganisations.

We would like to note that the theoretical work we present here has already been redacted in a paper co-authored with members of the Cryptocurrency Research Centre of Imperial College London [62]. The paper has already been accepted for publication in a special issue of the Royal Society Open Science journal on Blockchain technology and is due to appear shortly after this work is submitted.

In support of our theoretical work and analysis, we have developed a prototype implementation for QRWit, which we describe in detail in Section 5. We will show how the changes we propose can be implemented as a soft fork using a similar approach as, for example, SegWit [40].

1.3 Objectives

At the beginning of the project, we set the following goals. To clear away any suspense and to calm the reader's curiosity, we will also mention in what proportion we managed to achieve them.

1. Develop a scheme that can safely transition ECDSA secured funds to quantum resistance. This goal was certainly achieved as we will show throughout this paper.

2. Publish a paper on this scheme. We set this goal because we noticed that there are no other papers or well defined proposals for how to solve this dire problem. As mentioned, we achieved this, as our paper [62] was accepted for publication.

3. Optimise the scheme to maximise usability (i.e. flexibility, costs, transition time). This goal was partly achieved. We managed to create a completely flexible scheme that allows consumption and creation of any type of transaction outputs. Furthermore, we believe the costs are minimal as no scheme involving less than two transactions could offer the same security guarantees. On the other hand, we recently discovered a further modification to our scheme that would allow for the transition time to be user configurable, thus allowing for even more flexibility. Therefore, we can say that the current implementation of QRWit can be optimised further as we explain in Section 8.

4. Give a prototype implementation of QRWit in Bitcoin. This goal was also partly achieved. The current implementation successfully implemented all the new protocol updates, but we did not manage to fully update the relaying mechanism to ensure communication with un-upgraded clients. Although this is not a hard requirement of the Bitcoin consensus protocol as upgraded nodes represent a majority so the consensus is still established, it would have been a nice addition to our scheme.

The remainder of this paper is organised as follows. Section 2 presents the background knowledge necessary to understanding the core problem and the solution. In particular, we give a somewhat formal and extensive introduction to elliptic curve cryptography, quantum computing, Bitcoin, and post-quantum cryptography. In Section 6, we present all the alternative proposals of which we became aware while working on this project. Having covered the basic theory, we are in a position to fully describe the impact QCs have on Bitcoin (i.e. vulnerabilities uncovered, attack models and their feasibility), in Section 3. Consequently, in Section 4 we propose a protocol for the transition from Bitcoin's ECDSA to a quantum-resistant signature scheme and address certain concerns that the community might have (e.g. costs, flexibility, time frame). The specific prototype implementation demonstrating the workings of our protocol is given in Section 5 and its effectiveness, correctness, and scalability is assessed in Section 7. Furthermore, we present some extension possibilities and specific optimisations that will improve the overall usability of the scheme we suggest, in Section 8. Finally, we conclude this report with Section 9.

Section 2: Background

In this section we briefly present the basic concepts needed to understand the rest of the material. We will use these building blocks to explain the problem our work is aiming to solve and, in doing so, we will construct a clear image of the threats that quantum computing is posing to Bitcoin.

Firstly, we will take a look at elliptic curve cryptography which is the mechanism that ensures Bitcoin users maintain control of their funds and which is the main vulnerability from the point of view of a quantum-capable attacker. Secondly, we will go over the main aspects of Bitcoin that are relevant to understanding the problem and our solution. Then, we will cover the basics of quantum computing and give two examples of quantum algorithms that are of interest in the context of this paper. Finally, we will present some alternatives to elliptic curve cryptography that are believed to be secure even in the face of quantum computers.

2.1 Elliptic Curve Cryptography

Public-key cryptography is the first cryptosystem that relies on number theory rather than simple substitutions or permutations. Such cryptosystems are intrinsically asymmetric, requiring the use of two keys to encrypt and decrypt, thus providing a solution to the problem of secret key sharing. The concept was publicly introduced in 1976 by Whitfield Diffie and Martin Hellman [24] and one year later Ron Rivest, Adi Shamir and Len Adleman leveraged the intractability of factoring large integers to realize the first practical implementation: the RSA cryptosystem [55].

Elliptic curve cryptography (ECC) is a form of public-key cryptography developed in 1985 by Neal Koblitz and Victor Miller [37]. While it provides the same functionality as RSA, ECC's security relies on another hard mathematical problem: the elliptic curve discrete logarithm problem (ECDLP). At the moment, the most efficient known classical algorithms for solving ECDLP have fully exponential runtime complexity [44] which is an improvement over the subexponential-time algorithms that can factor large integers [39]. Thus, the same level of security can be achieved with smaller keys in elliptic curve systems than in RSA. As smaller key sizes imply a more efficient use of power, bandwidth, and storage, many applications, including Bitcoin, make use of ECC.

The remainder of this section is based on Darrel Hankerson's, Alfred Menezes', and Scott Vanstone's book "Guide to Elliptic Curve Cryptography" [30].

2.1.1 Elliptic Curve Arithmetics

Definition 1 (Abelian Group)
An abelian group $(G, \star)$ consists of a set G with a binary operation $\star : G \times G \to G$ satisfying the following properties:

1. ***Closure**: $\forall a, b \in G$ we have that $a \star b \in G$*
2. ***Associativity**: $a \star (b \star c) = (a \star b) \star c \ \forall a, b, c \in G$.*
3. ***Existence of an identity**: $\exists e \in G$ such that $a \star e = e \star a = a \ \forall a \in G$.*
4. ***Existence of inverses**: $\forall a \in G \ \exists b \in G$, called the inverse of a, such that $a \star b = b \star a = e$.*
5. ***Commutativity**: $a \star b = b \star a \ \forall a, b \in G$. This property is required for a group to be abelian.*

Remark 1 (Exponentiation)
For any group, we will denote exponentiation to mean repetitive applications of the group operation on the same element. i.e. $g^t = \underbrace{g \star \cdots \star g}_{t}$.

Definition 2 (Order of a Group)
*When G is a finite set, the group is called a finite group and the number of elements in set G is called the **order of the group**.*

Definition 3 (Cyclic Subgroups)
If $(G, \star)$ is a finite group of order n and $g \in G$, then the smallest positive integer t such that $g^t = 1$ is called the order of g and the set

$$\langle g \rangle = \{g^i : 0 \leq i \leq t - 1\}$$

*of all powers of g is itself a group under the same operation as G, and is called the **cyclic subgroup** of G generated by g.*

Definition 4 (Cyclic Groups)
*If G has an element g of order n, then G is said to be a **cyclic group** and g is called a generator of G.*

Definition 5 (Field)
A field is a set F together with two operations, addition (denoted by $+$) and multiplication (denoted by $\cdot$), that satisfy the usual arithmetic properties:

1. *$(F, +)$ is an abelian group with (additive) identity denoted by 0.*
2. *$(F \setminus \{0\}, \cdot)$ is an abelian group with (multiplicative) identity denoted by 1.*
3. *The distributive law holds: $(a + b) \cdot c = a \cdot c + b \cdot c \ \forall a, b, c \in F$.*

As for groups, the field is said to be finite, if the set F is finite. The order of the field is the number of elements in set F.

Theorem 1 (The order of a finite field is a prime power)
There exists a finite field F of order q if and only if q is a prime power, i.e. $q = p^m$ where p is a prime number called the characteristic of F and m is a positive integer.

Theorem 2 (Finite field isomorphism)
For any prime power q, there is essentially only one finite field of order q.

Informally, this means that any finite fields of order q has the same structure of elements (maybe the labelling differs). Hence, we say that any two finite fields of order q are isomorphic and denote such a field by F_q.

Definition 6 (Subfields and Extension fields)
A subset k of a field K is a subfield of K if k is itself a field with respect to the operations of K. In this instance, K is said to be an extension field of k.

Definition 7 (Elliptic Curve)
An elliptic curve E over a field K is defined by an equation

$$E : y^2 + a_1xy + a_3y = x_3 + a_2x_2 + a_4x + a_6$$

where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 = 0$ with

$$d_2 = a_1^2 + 4a_2, \quad d_4 = 2a_4 + a_1a_3, \quad d_6 = a_3^2 + 4a_6,$$
$$d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

If L is any extension field of K, then the set of points on E is

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\}$$

where ∞ is the point at infinity.

Remark 2 (Comments on Definition 7)

1. *The equation of an Elliptic Curve is called a Weierstrass equation.*
2. *E is defined over K because the coefficients a_1, a_2, a_3, a_4, a_6 of its defining equation are elements of K. Note that if E is defined over K, then E is also defined over any extension field of K, by Definition 6.*
3. *The condition that $\Delta = 0$ ensures there are no points at which the curve has more tangent lines.*
4. *The point ∞ is the point at infinity and it lies on any vertical (parallel with the line of equation: $x = 0$).*
5. *The points on E are the points (x, y) that satisfy the equation of the curve and whose coordinates are in L. ∞ is considered a point on all extension fields L of K.*

Definition 8 (Simplified Weierstrass equations)
Two elliptic curves E_1 and E_2 defined over K are said to be isomorphic over K if $\exists u, r, s, t \in K, u \neq 0$, such that the change of variables in the respective Weierstrass equations:

$$(x, y) \to (u^2x + r, u^3y + u^2sx + t)$$

*transforms equation E_1 into equation E_2. This transformation is called an **admissible change of variables**.*

Due to Definition 8 we can drastically simplify the elliptic curve equation. Depending on the characteristic of the field K there are three simplifications that can be made. If the characteristic of K is not equal to 2 or 3, then the admissible change of variables:

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 4a_1 a_2 - 12a_3}{24} \right)$$

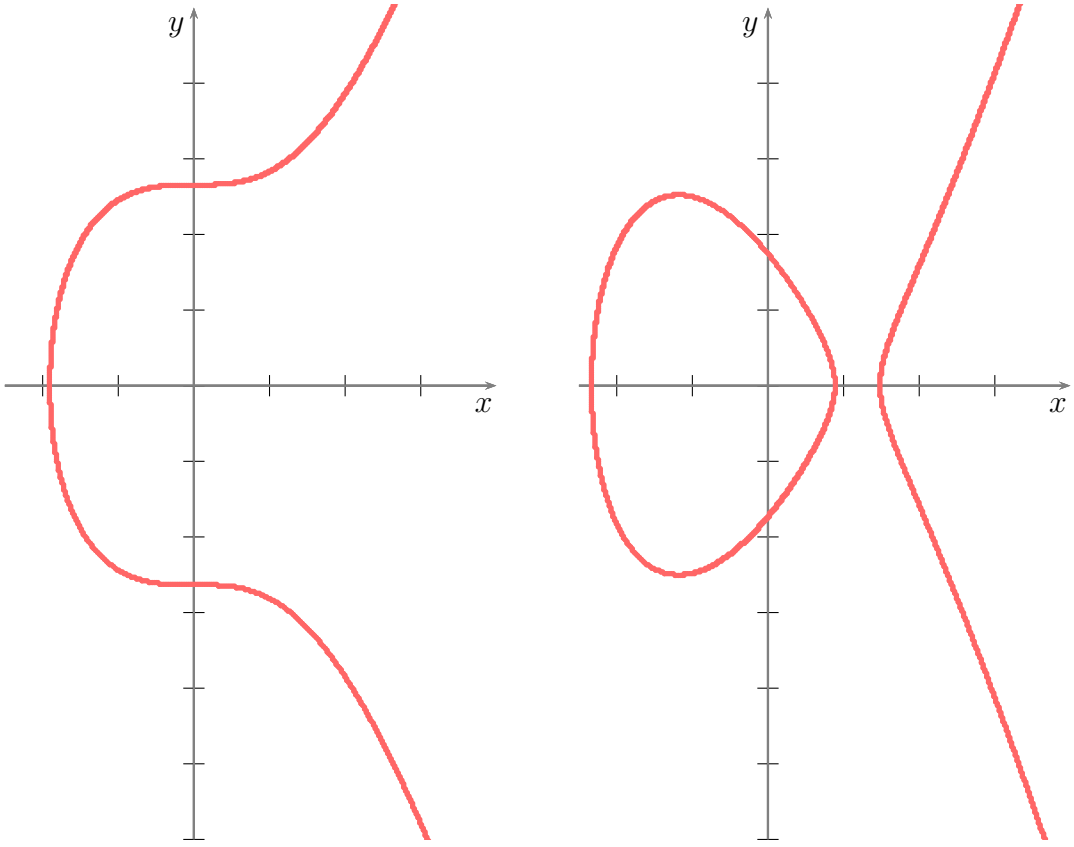
transforms E to the curve:

$$\mathbf{y^2 = x^3 + ax + b}$$

where $a, b \in K$ and the discriminant becomes $\Delta = -16(4a^3 + 27b^2)$.

In cryptography, the characteristic of K has to be large to ensure security, hence this simplification can certainly be made and is the usual form elliptic curves appear in domain literature.

Below, in Figure 2.1, are two elliptic curves over the field $(\mathbf{R}, +, \cdot)$ where addition and multiplication have the usual meaning. On the left we show the elliptic curve used in Bitcoin.



(a) E : $y^2 = x^3 + 7$ (b) E : $y^2 = x^3 - 4.2x + 3$

Figure 2.1: Two elliptic curves over the infinite field $(\mathbf{R}, +, \cdot)$, i.e. the real numbers.

Elliptic Curve Point Addition

Geometrically, elliptic curves exhibit some interesting properties. Until now we have described the set of points $E(K)$ of an elliptic curve E defined over a field K. Further, we define an addition rule over $E(K)$ to form an abelian group with ∞ serving as its identity. This group structure is used in the design of elliptic curve cryptographic systems as the elliptic curve digital logarithm problem is defined over it.

Before we give the formal group law of elliptic curves, we would like to show its graphical properties. Hence, addition of elliptic curve points is achieved through the following rule. For a graphical representation see Figure 2.2 below.

Definition 9 (Geometric Representation of Elliptic Curve Group Law) *The sum $Q = P + R$, is defined geometrically through the **chord-and-tangent rule** as follows:*

Draw a line through P and R. This line intersects the elliptic curve E at a third point $(-Q)$. Then Q is the reflection of this point about the X-axis.

Remark 3 (Special Cases)

1. *When $P = R$ the line through P and R is a tangent of E.*
2. *For a point $P = (x, y)$, the point $P' = (x, -y)$ is called the negative of P, i.e. $P = -P'$ and their addition is, as expected, the identity element ∞.*

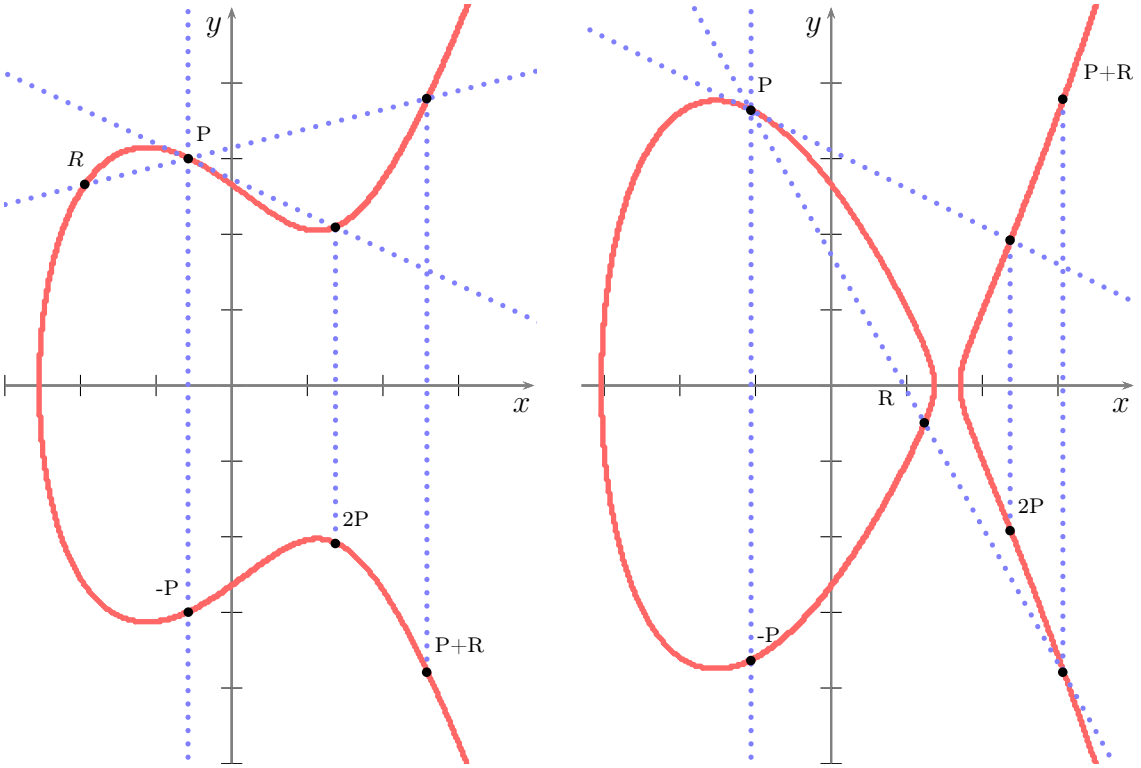


Figure 2.2: Geoemetric representation of elliptic curve addition on two different curves. In both cases we show how to compute $P + R, 2P$ and $P - P$.

To give a formal description for the operation above we define the elliptic curve group law:

Definition 10 (Elliptic Curve Group Law)
The group operation on points of E is represented by '+' and is defined as follows.
If $P \in E$ then $P + \infty = \infty + P = P$ as ∞ is the identity element.
If $P = (x_1, y_1), R = (x_2, y_2) \in E$ then:

$$P + R = \begin{cases} \infty & \texttt{when } (x_2, y_2) = (x_1, -y_1), \\ (x_3, y_3) & \texttt{otherwise,} \end{cases}$$

where $x_3 = \lambda^2 - (x_1 + x_2), y_3 = \lambda(x_1 - x_3) - y_1$

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \texttt{if } P \neq R, \\ \dfrac{(3x_1^2 + a)}{2y_1} & \texttt{if } P = R, \end{cases}$$

Note that all operations are performed over the field K.

Elliptic Curve Discrete Logarithm Problem (ECDLP)

Using all the above theory, we are finally in a position to introduce the elliptic curve discrete logarithm problem, whose hardness is crucial for the security of any elliptic curve cryptographic scheme. The ECDLP belongs to a class of problems called the Hidden Subgroup, hence we will first show what this subgroup refers to by noticing the following:

Remark 4 (Elliptic Curve subgroup of P)
Given an elliptic curve E defined over a finite field F_q and the elliptic curve group law denoted as $+$, we can define the abelian group $(E(F_q), +)$. In respect to this group, we have from Definition 3 any point $P \in E(F_q)$ is a generator of a cyclic subgroup $\langle P \rangle$ of some order n.

Definition 11 (ECDLP)
For some $Q \in \langle P \rangle$, find the integer $l \in [0, n-1]$ such that $Q = P^l = \underbrace{P + \ldots + P}_{l}$.
The integer l is called the discrete logarithm of Q to the base P, denoted $l = \log_P Q$.

Remark 5 (Exponentiation or Multiplication)
Because exponentiation refers to repeated additions here, we will use multiplication from now on to represent it.

Remark 6 (Hardness of the ECDLP)

1. *In order to find l one could compute the series: $P, 2P, 3P, \ldots$ until finding Q, but this has time complexity $O(n)$ and choosing $n \approx 2^{80}$ is intractable. Other optimizations exist that speed up this search, but the time complexity is still fully exponential in the number of bits of n.*

2. *We should note that there is no proof that the ECDLP is intractable, but there is no publicly known efficient algorithm that can solve it.*

Remark 7 (Complexity of Multiplication)
Note however, that computing kP for some random integer k can be achieved much easier using the following algorithm:

Algorithm 1: Elliptic Curve Point Multiplication

Input : k, P
Output: kP
$t \leftarrow$ *number of bits in* k
$Q \leftarrow \infty$
for $i \leftarrow 0$ **to** $t - 1$ **do**
    if $k_i = 1$ **then**
        $Q \leftarrow Q + P$
    $P \leftarrow 2P$
return Q

The time complexity of this algorithm is $O(t) = O(\log_2 n)$ which is definitely tractable. Furthermore, if the point P is known beforehand, there are algorithms which use this knowledge to pre-compute a table of powers of p which can then be used to speed up the algorithm even more.

2.1.2 Elliptic Curve Public-Private Keys

Considering remarks 6 and 7, we can create a public-key cryptographic system relying on ECDLP. Firstly we need to decide on some domain parameters which will be made publicly available to anyone wishing to use the cryptosystem. For the purposes of this paper, we will assume that a field K, a curve E, and a point $P \in E(K)$ with known order n were chosen such that the ECDLP is indeed intractable. Now to generate a pair of private and public keys:

Definition 12 (Private and Public Key Generation)
1. *As **private key**, choose a random integer $k \in [0, n-1]$.*
2. *As **public key**, compute $Q = kP$.*

Remark 8 (Trapdoor Function Explained)
While it is easy to compute $Q = kP$ using algorithm 1 or an optimised version, it is intractable to compute k knowing just Q, as established by the ECDLP. Thus, this construction of the private-public key pair acts a trapdoor function.

2.1.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a scheme based on elliptic curve cryptography that implements the Digital Signature Standard (DSS) [13]. The purpose of signature schemes is to replace handwritten signatures, or, in the context of digital signatures: to provide authentication, integrity, and non-repudiation. Digital signatures are closely related to the concept of private and public keys. A signature created with some private key will be correctly verified only by the corresponding public key.

Definition 13 (Signature Scheme)
A digital signature scheme must describe 4 algorithms:

1. *A domain parameter generation algorithm that generates a set D of domain parameters.*
2. *A key generation algorithm that takes as input a set D of domain parameters and generates key pairs (Q, d).*
3. *A signature generation algorithm that takes as input a set of domain parameters D, a private key d, and a message m, and produces a signature Σ.*
4. *A signature verification algorithm that takes as input the domain parameters D, a public key Q, a message m, and a purported signature Σ', and accepts or rejects the signature.*

The relevant algorithms for constructing and validating ECDSA signatures are given below. Note that in the following two algorithms, H is a cryptographic hash function whose outputs have bitlength no more than that of n.

Algorithm 2: ECDSA Signature Generation

Input : private key d, message m, P, n
Output: signature (r, s)

Select $k \in [1, n-1]$.
Compute $kP = (x_1, y_1)$ and convert x_1 to an integer x_1'.
Compute $r = x_1' \mod n$.
if $r = 0$ **then**
    go to step 1
Compute $e = H(m)$.
Compute $s = k^{-1}(e + dr) \mod n$.
if $s = 0$ **then**
    go to step 1
return (r, s)

Algorithm 3: ECDSA Siganture Verification

Input : signature (r, s), public key Q, message m, P, n
Output: True or False

if $r \notin [1, n-1]$ *or* $s \notin [1, n-1]$ **then**
    return *False*
Compute $e = H(m)$.
Compute $w = s^{-1} \mod n$.
Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$.
Compute $X = u_1 P + u_2 Q$.
if $X = \infty$ **then**
    return *False*
Convert the x-coordinate x_1 of X to an integer x_1'; compute $v = x_1' \mod n$.
return $v == r$

Proof 1 (Proof of Signature Verification)
If signature (r, s) on message m was indeed generated by the legitimate signer, then $s \equiv k^{-1}(e + dr) \mod n$. Which rearranged gives:

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d \mod n$$

On the other hand, $X = u_1P + u_2Q = u_1P + u_2(Pd) = (u_1 + u_2d)P = kP$, and so $v = r$ as required.

With this, we conclude our introduction to Elliptic Curve Cryptography and move on to the following sections in which we will see how Bitcoin uses ECDSA and how quantum computers can solve the ECDLP.

2.2 Some Cryptographic Primitives Used In Bitcoin

This section introduces some cryptographic primitives used in Bitcoin that are essential to understanding the rest of the paper as we make use of them regularly.

2.2.1 Cryptographic Hash functions

Definition 14
An ideal cryptographic hash function H takes an input of any size and returns an output of predefined size. Furthermore, it must exhibit the following properties:

1. ***Deterministic*** *- It produces the same result for the same input, regardless of any other factors.*
2. ***Efficiency*** *- It is effortless to compute the hash of some arbitrary message.*
3. ***Butterfly Effect*** *- A small change to the input changes the output completely.*
4. ***Pre-image Resistance*** *- Given h it should be infeasible to find m such that $H(m) = h$.*
5. ***Collision Resistant*** *- It is infeasible to find two different messages $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$.*
6. ***Second Pre-image Resistance*** *- Given m_1 it should be infeasible to find $m_2 \neq m_1$ such that $H(m_2) = H(m_1)$.*

Remark 9
The last two properties of definition 14 are both required because they protect against different types of possible vulnerabilities. Collision resistance protects against special types of messages which can be modified in a way that will yield the same hash. On the other hand, second pre-image resistance refers to the more specific case in which a message is already known and a second message with the same hash is wanted.

In Bitcoin there are a few important hash functions that are used to offer immutability guarantees. Hashing a piece of data is a common practice for creating a fingerprint of the data. If even one bit of the data would change the hash would be completely different.

SHA256 is a secure hash algorithm (SHA) part of the SHA-2 family of cryptographic hash functions designed by the National Security Agency (NSA) and is believed to be an ideal cryptographic hash function, satisfying all the required properties. It returns outputs with exactly 256 bits.

Double SHA256 (D-SHA256) is the double application of SHA256. Essentially, $\texttt{D-SHA256}(m) = \texttt{SHA256}(\texttt{SHA256}(m))$. It is used in Bitcoin, instead of the simple SHA256 in case a length extension attack on SHA256 is ever found.

RIPEMD160 is a cryptographic hash function based on the MerkleDamgard construction, being part of the RIPEMD family of cryptographic functions. It returns outputs with exactly 160 bits and is also believed to be an ideal

2.2.2 Merkle Tree

Merkle Trees were patented in 1979 by Ralph Merkle [45]. They are trees in which each leaf is some block of data and each non-leaf node is the hash of the concatenation of its children. An example of this is in Figure 2.3 below.

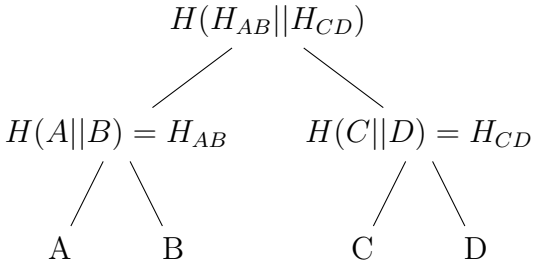


Figure 2.3: Merkle tree with four data blocks.

Assuming H is an ideal cryptographic hash functions, we observe that changing one bit in any of the leaves, will completely change the root node as ensured by property 3 of definition 14. This is especially useful for easily checking the integrity of large data structures.

Proof of Existence

An interesting property of Merkle trees is the fact that we can construct proofs that some data is present in the Merkle tree without actually transmitting the whole structure. The proof creators need all the data in the leaves in order to create

proofs, but the verifiers only need to have a trusted value for the root node in order to be completely convinced that some data exists in the tree. Although the proof creator can simply deny answering a proof request, he cannot trick the requester by providing a misleading proof as this would involve breaking the pre-image resistance of H. Such proofs are also called Merkle branches, because they essentially provide all the information needed to recompute one branch of the tree from leaf to root.

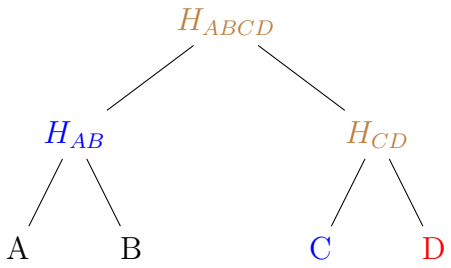


Figure 2.4: The data that needs to be proven to be in the merkle tree is in red. The merkle branch to prove this is depicted in blue. The verification requester can compute the brown values from the blue and red values, thus being able to check that the value of the root matches the trusted value he owns.

Below we give the algorithm that a proof creator would use to generate the proof that some node exists.

Algorithm 4: Merkle Tree Branch Construction

```
1 function constructBranch(node)
2     if node is root then
3         return ([], [])
4     end
6     (hashes, bits) = constructBranch(node.parent)
7     hashes.append(node.sibling)
8     bits.append(node is on the right)
9     return (hashes, bits)
```

The algorithm starts at the leaf we want to prove appears in the merkle tree and builds a list of sibling hashes, marking at each step whether the sibling was on the left or on the right.

Next we give an algorithm that verifiers could use to check the correctness of some existence proof. Note that, they must already be in possession of the root value as otherwise they would have nothing to match against.

Algorithm 5: Merkle Tree Branch Verification

```
1  function verifyBranch(hashes, bits, data, root)
2      if bits is empty then
3          return data == root
4      end
5      if bits[-1] = True then
7          data = H(hashes[-1]||data)
8      end
9      else
11         data = H(data||hashes[-1])
12     end
13     return verifyBranch(hashes[0:-1], bits[0:-1], data, root)
```

The algorithm starts from the data we required verification of and computes successive hashes using the provided sibling at each step. When it runs out of siblings, the full root should have been constructed, so a comparison with the trusted value of the root is done.

2.3 Bitcoin

This section gives a high-level overview of Bitcoin, while focusing with some more detail on the aspects which are specifically important to our work. One can find a more complete description of Bitcoin in Andreas M.Antonopoulos' book "Mastering Bitcoin" [15], on which this section is based.

Bitcoin can be described as a set of standards, ideologies, protocols and technologies that form the basis of a decentralized currency system. The units of digital money are called bitcoins and their ownership can be handed over to other participants through transactions. Users communicate to each other, broadcasting transactions, via a peer-to-peer network over the Internet. Transactions are aggregated together into blocks which are appended to a distributed public ledger called the blockchain. To avoid a central authority that issues new blocks, any consensus participant can append a new block after completing a cryptographically hard puzzle. Each block also contains the hash of its predecessor, thus providing strong guarantees for the immutability of the transaction history.

2.3.1 Blockchain Technology

For the purpose of this section, it can be assumed that transactions are just chunks of data of arbitrary lengths. The blockchain structure is an ordered list of blocks of transactions where each block links to its predecessor by referring to its hash in the block header. We shall first define some basic terms in Bitcoin.

Definition 15 (Block Reward)
The block reward is a diminishing amount of newly created currency that is awarded to miners that successfully append a block to the blockchain. The block reward halves

every 210,000 blocks. It started at 50 Bitcoins and is now at 12.5. Apart from this reward miners also collect the fees associated to transactions in a block.

Definition 16 (Block Header)
The structure of a block header is:

Field	Description	Size
Version	A version number to track protocol updates.	4 bytes
Previous Block Hash	The hash (D-SHA256) of the header of the previous block.	32 bytes
Merkle Root	A hash of the root of the Merkle Tree of this block's transactions.	32 bytes
Timestamp	The approximate creation time of this block (in seconds from UNIX Epoch).	4 bytes
Difficulty	The proof-of-work algorithm difficulty target for this block.	4 bytes
Nonce	A random number found as solution to the proof-of-work algorithm.	4 bytes

Table 2.1: Structure of the block header.

Block

Each block in the blockchain contains all the transactions included in that block, a block header and some meta-data (see table below).

Field	Description	Size
Block Size	The size of the block, in bytes, excluding this field.	4 bytes
Block Header	The header of the block.	80 bytes
Tx Count	Number of transactions in this block.	1-9 bytes
Transactions	Vector with all the transactions in this block.	Variable

Table 2.2: Structure of a Bitcoin block.

Within the blockchain, each block is identified by a hash (blockhash), generated using the D-SHA256 cryptographic hash algorithm on the header of the block. Furthermore, each block specifies the blockhash of the previous block (parent block) in its own header. Thus, a sequence of hashes linking each block to its parent is created. Although, any block has exactly one parent, it is possible for a block to have multiple children. This could be due to two miners generating a new block at approximately the same time. However, the network cannot operate on two chains in parallel and all consensus participants need to decide to follow the same chain. The rule that Bitcoin imposes is to **follow the longest chain**. The purpose of this long cryptographic hash chain is to ensure that historical blocks cannot be altered in any way. If an attacker would try to modify the hash of a deeply buried block, he would break the link with the blocks on top of this block. This would immediately

yield a shorter chain than the original one so other consensus participants would immediately disregard it. As the blockhash is sensible to changes in the merkle tree root of all transactions in a block, this implies that any change to any transaction in a block would require recomputing all the blocks on top of the altered one. This hash chain goes all the way back to the first block ever created, also known as the genesis block.

The Genesis Block

The first block that was ever created in Bitcoin is called the genesis block. It was created in 2009 and was statically encoded within the Bitcoin client software. Its purpose is to serve as a trusted root for all participants to build the blockchain upon. This block only contains one transaction that created the first units of Bitcoin ever.

Mining and Proof of Work (PoW)

Mining serves to protect users against fraudulent transactions (eg.: double spending), and to create a consensus on the current main chain. Miners validate new transactions and record them in blocks. A new block gets added to the blockchain approximately every 10 minutes. For donating their computational power to validate transactions, miners are rewarded in two ways: transaction fees and block rewards. However, in order to publish a block to the blockchain, miners compete to solve a cryptographically hard puzzle called Proof of Work.

Definition 17 (Proof of Work)
Given a block header, find a nonce such that D-SHA256(block header) < difficulty, where the difficulty is specified in the block header and the nonce has to be inserted in the block header.

Remark 10 (Hard to find, easy to prove)
Although it is extremely hard to solve PoW as the only guaranteed way is to test random numbers for the nonce, it is trivial to check that a solution to PoW is correct or not. All that is needed is one hash computation and one comparison with the difficulty.

The reason PoW needs a difficulty threshold, is to regulate the rate of new block additions. The only reason consensus participants can establish which is the longest chain is that miners prefer to mine blocks on top of the longest chain, so eventually one of the competing chains will become the longest as more miners are competing on it. However, if miners would be able to generate new blocks much faster than 10 minutes, then competing chains would exist for much longer as blocks are relayed across the globe and a split of very distant miners would appear. This is why the difficulty threshold adjusts automatically such that the average block time remains at around 10 minutes.

Blockchain Forks

As the blockchain is a decentralized data store, multiple variants of it could exist that are not consistent. Blocks are routed through the network from peer to peer, so they might arrive at different nodes at different times. This might give some miners an advantage as they can start mining the next block faster. Although all miners, only mine on top of the longest chain, it can sometimes happen that miners extend the same block with different chains at an approximately equal rate. This would create two competing chains. In time, the one on which more miners are working will exceed the other one, thus convincing all miners to move towards it. An example of this can be seen in the diagram below.

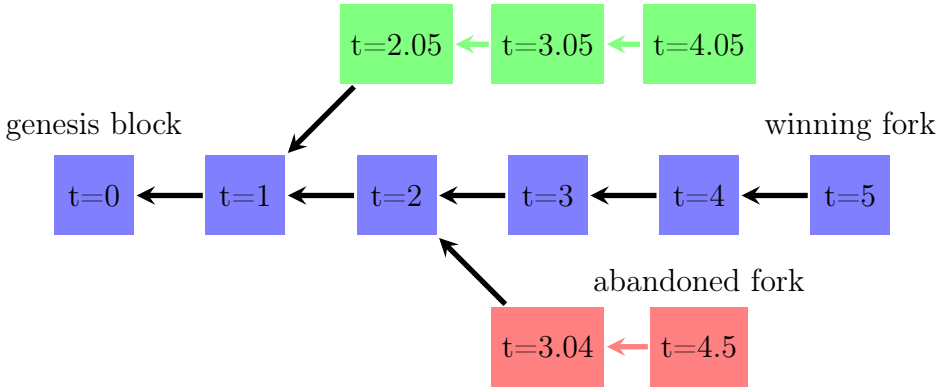


Figure 2.5: Three competing forks; blocks are marked with the creation time. The green miners mine at the same rate as the blue ones until they receive block t=5 which clearly proves to them that the chain they are working on is not the longest. At the same time, the red miners cannot mine as fast as the blue miners and they become aware of this when they receive block t=4. Unfortunately, they have already mined block t=4.5, but they abandon it as they see the longest chain is going to be on top of t=4.

As such, forks occur as temporary inconsistencies between competing versions of the blockchain, which are resolved by eventual reconvergence as more blocks are added to one of the forks at a higher rate.

Consensus forks

Hard Fork - If the current protocol rules are extended and become less strict (i.e. a transaction that was previously invalid is now valid), then old (un-upgraded) clients will not understand the new rules and will not follow the chain of the new miners who will basically be forced to transact only between them. This situation can only be resolved if either all clients upgrade to the new rules or the upgraded clients give up their fork and start running the original code again.

Soft Fork - If the current protocol rules are extended and become more strict (i.e. a transaction that was previously valid is now invalid), then the success of the fork depends on how many miners switch to the new rules. If there is a majority

of miners switching, the fork will be valid as there is more hashing power working towards extending the chain with the new rules. Old miners would see the chain as valid by their rules, so they would not disagree. However, they would not be able to use the new features until they upgrade. On the other hand, if the majority of the hashing power runs the old code, then the fork will fail. Upgraded clients will be able to publish their transactions and they would be valid, but the new rules would not be in effect.

Thus it is usually the case, that any software update in Bitcoin is better deployed as a soft fork. This means that in case the majority of consensus enforcers do not want to upgrade then the fork just fails instead of permanently remaining split in the case of a hard fork.

2.3.2 Bitcoin Network

The Bitcoin network is a peer-to-peer network, meaning that nodes (clients, miners, etc.) connect to other nodes and form a mesh network over which they relay (distribute) transactions and blocks. Nodes can answer to special type of requests from other nodes soliciting data they haven't seen. Even if the network communication layer is not reliable or secure, the security of the Bitcoin system is not undermined as hash functions form digital fingerprints (checksums) of transactions and blocks.

Full Node

A full node is a node which stores all the transactions and all the blocks and indexes them accordingly. A full node has complete knowledge of the state of transactions and can serve information to other nodes which are trying to build their own copy of the blockchain. To run a full node one requires relatively powerful hardware memory and CPU wise.

Miners are full nodes, that also participate in the competition of publishing the next block to the blockchain. They have no incentive to spam, misinform, or trick other nodes as the consensus rules would ban the miner for a period of time and, thus render his hashing power useless. This would imply major financial losses for the miner.

Simple-Payment-verification (SPV) Node

SPV nodes are lightweight devices that only store a list of block headers and relevant transactions (eg: only the ones associated to the user) instead of all the transactions in the whole blockchain. The reason SPV nodes can exist is because of the merkle tree structure in which transactions are stored. As shown in Section 2.2.2, it is possible for an SPV node to check the existence of a transaction in the blockchain by simply asking a full node for a proof of existence. The full node would return a merkle branch, which the SPV node would check. This would completely convince him that his transaction is included in the blockchain.

2.3.3 Bitcoin Transactions

Transactions are the most basic and important structure in Bitcoin; they are data structures that encode the transfer of ownership of funds. All the other parts of the system are designed to ensure that transactions can be created, propagated on the network, validated, and persisted.

Transactions are formed of inputs, outputs and some meta-data. The basic (before segregated witness, a.k.a SegWit, was deployed) format of a transaction is depicted in the figure below.

Field	Description	Size
nVersion	Version number used to specify what consensus rules this transaction should be treated under.	4 bytes
nIn	Number of inputs	1-9 bytes
ins	Vector referencing some unspent outputs that can be consumed as part of this transaction	Variable
nOut	Number of outputs	1-9 bytes
outs	Vector specifying some outputs that will be created by this transaction	Variable
nLocktime	Locktime	4 bytes

Table 2.3: Structure of a Bitcoin transaction before segregated witness activated.

Transactions are identified by their txid or hash, which is obtained by applying SHA256 on the transaction data.

A Coinbase Transaction (or generation transaction) is a special type of transaction meant to create new units of currency out of nothing. Miners include such a transaction as the first transaction in every block to pay themselves the fees associated to all the transactions in the block and the block reward.

Transaction Inputs reference some previously unspent output, consuming the entirety of the funds. Consensus participants can check if an output is spent or unspent because they maintain a set of unspent transaction outputs (UTXOs), which they update continuously as new transactions are formed and new blocks are received. The sum of the funds in all the inputs must be spent by the outputs. Any remainder will be considered as transaction fees. The structure of an input is given in Table 2.4.

Field	Description	Size
Transaction Hash	Hash of the transaction where the UTXO is.	32 bytes
Output Index	Index of the output in its transaction.	4 bytes
ScriptSig Size	Size of the scriptSig.	1-9 bytes
ScriptSig	The unlocking script.	Variable
Sequence Number	Not used anymore	4 bytes

Table 2.4: Structure of a Bitcoin transaction input.

Thus, to consume some unspent output, a user needs to know the transaction in which it resides and the index of the output. However, to successfully consume an output, the user also needs to provide a scriptSig that can successfully unlock the scriptPubKey found in the referenced output.

Transaction Outputs are the building blocks of transactions. They are both produced as outputs out of a transaction, but also referenced as inputs. The outputs must consume all the Bitcoins produced by the inputs, minus a remainder which will be automatically considered as fee. The structure of an output is quite simple (see Table 2.5).

Field	Description	Size
Amount	Value of this output.	4 bytes
scriptPubKey Size	Size of the scriptPubKey.	1-9 bytes
scriptPubKey	The locking script.	Variable

Table 2.5: Structure of a Bitcoin transaction output.

Unspect Transaction Outputs (UTXOs) are tracked by validating nodes continuously. When a new transaction references some outputs, each of them is checked for existence in the current UTXO set. If the transaction is validated, the output is removed from the UTXO set, making all other transactions, that reference the same output, invalid. At the same time, the outputs that are created by a valid transaction are added to the UTXO set. Thus a new block contains transactions that have all the inputs referencing already existing outputs.

Transaction Scripts

In Bitcoin, an UTXO is secured by a locking script, historically known as scriptPubKey. The challenge a UTXO poses to the world is to provide a scriptSig that together with the scriptPubKey will evaluate to True. The Bitcoin scripting language, Script, is not Turing complete to avoid infinite loops and complex scenarios that can lead to resource exhaustion and denial of service attacks when executed. Script is quite flexible though and allows for a variety of scripts to be composed. All scripts can be verified without any state and all the operations have op_codes that represent stack manipulations such as push, pop, and compute data.

Very complex scripts can be created, but most users just construct the following basic scripts.

1. **Pay-To-Public-Key (P2PK)** is an old style script that is no longer used in Bitcoin, but we will present it here for completeness. A P2PK locking script looks like this: `<Public Key A> OP_CHECKSIG`, and the matching unlocking script that would appear in an input wanting to unlock this output should be: `<Signature from Private Key A>`. Hence, the concatenated script would be: `<Signature from Private Key A> <Public Key A> OP_CHECKSIG`, which evaluates to True.

2. **Pay-To-Public-Key-Hash (P2PKH)** is the improved version of the above. To minimize the size of scriptPubKey, the public key is moved to the scriptSig and only a hash of it is required in the output. The scriptSig and scriptPub key become: `<Signature from Private Key A> <Public Key A>` and `OP_DUP OP_HASH160 <Public Key Hash A> OP_EQUAL OP_CHECKSIG`, which will return True if the public key hashes to the correct value and the signature matches.

3. **OP_RETURN (data)** is a type of output that is provably un-spendable as it pushes data onto the stack. Whatever scriptSig provides, this type of script will not unlock. In fact, nodes do not even keep this sort of outputs in the UTXO set as they are provably un-spendable. This sort of output is used mainly to persist data on the blockchain or to tag transactions. For denial of service reasons, each transaction is allowed to have maximum one such input and the length of the scriptPubKey has to be maximum 80 bytes. Such a script would look like this: `OP_RETURN <data>`.

4. **Pay-To-Script-Hash (P2SH)** outputs are an improvement over P2PKH. Instead of specifying whatever script we had in the scriptPubKey, it is much better to simply store a hash of it and to require the input to present a redeemScript that matches the hash. The extra consensus rules will require for the redeemScript to be ran after the remaining scriptSig. This type of output guarantees the fixed size of the scriptPubKey to 32 bytes. For example, if the original scriptPubKey was: redeemScript = `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG`, then the new scriptPubKey will be: `OP_HASH160 <20-byte hash of redeemScript> OP_EQUAL` and the scriptSig will be: `Sig1 Sig2 redeemScript`.

Note that with the exception of OP_RETURN, all other script types require a signature. This is a crucial part of a Bitcoin transaction as this ensures, the owner of the private key is the only one that can modify the transactions he created.

A Bitcoin address is a base 58 encoded scripPubKey with an extra fingerprint. When a user wants to receive some money, he usually uses a public key to create a scriptPubKey of one of the aforementioned types, encodes it in base 58, adds the fingerprint over the data, and sends it over to the payer. The payer removes the fingerprint, decodes the scriptPubKey in base 58, and, if the fingerprint matches,

includes it in the output. This abstraction allows users to protect against transmission errors, i.e. if one of the bits in the address is changes, the fingerprint will not match any more and the payer will know he is about to send funds to an invalid scriptPubKey.

Transaction Signatures

In Bitcoin digital signatures use the ECDSA signature scheme and sign the data of the transaction in part or fully, depending on the signature type specified:

1. **SIGHASH_ALL** is a flag specifying that the data to be signed is all the inputs and all the outputs. This does not let anybody modify the transaction you are creating.
2. **SIGHASH_NONE** is a flag specifying that all the inputs should be signed, but none of the outputs. This basically allows anybody to add, remove, or modify outputs.
3. **SIGHASH_SINGLE** is a flag specifying that all the inputs and only the output corresponding to this input should be signed. This is used when users want to ensure their output is not modified, but other outputs which do not belong to the user can be modified.
4. **SIGHASH_ANYONECANPAY** is a flag that can be used in combination with all the other flags to indicate that we want to sign only the current input and not the others. This is useful if we want to allow other users to contribute money to our transaction.

Elliptic Curve Secp256k1

ECDSA in Bitcoin is implemented over the elliptic curve Secp256k1. This curve was rarely used before Bitcoin, but due to its several nice properties it is gaining popularity. The structure was constructed to optimize computations so it is often more than 30% faster than other curves when sufficiently optimized.

Definition 18 (Secp256k1)
Let $E : y^2 = x^3 + 7$ be defined over F_p where $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ is a prime number. The generator of the cyclic subgroup of $E(F_p)$ is

$$G = 0479BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28\ldots$$

and its order is

$$n = \underbrace{FF\ldots F}_{31}EBAAEDCE6AF48A03BBFD25E8CD0364141$$

Considering the large characteristic of F_p and the large order of the cyclic subgroup, it is certain that classical computers would not be able to break ECDSA in Bitcoin.

2.3.4 Transaction Lifecycle

To conclude the Background section on Bitcoin we will present the lifecycle of a transaction. We assume user Bob wants to send n Bitcoins to user Alice.

1. Transaction Creation
   (a) Find UTXOs that Bob can unlock until the sum of their values is larger than n.
   (b) Crate the input vector by referencing each UTXO we found at the previous step.
   (c) Create an output to Alice, by specifying the amount n and the scriptPubKey that she supplied to us.
   (d) Create an output to Bob with the change. Leave some fees though, to give incentive miners to include our transaction in a block.
2. Transaction Signing - We will just use the general SIGHASH_ALL type, for simplicity. Go through each input and provide a corresponding scriptSig as explained in section 2.3.3. Include this scriptSig in the corresponding input.
3. Transaction Broadcasting - Broadcast the signed transaction to other nodes. They will validate it, by checking that the inputs are not double-spending and correctly unlock the outputs they reference. If validation passes, the transaction is included in their memory pools. Then, nodes will keep propagating the transaction until a miner picks it up from its memory pool, includes it in a block and publishes the block.
4. Transaction Confirmed - From time to time, Bob will ask full nodes if the transaction he created is part of the blockchain. When the transaction is indeed included, any full node can notify Bob that his transaction exists in the blockchain and offer him a Merkle branch proof that will convince him that the transaction is indeed accepted.
5. More confirmations - For extra security, Alice should wait a bit longer so that more Proof Of Work is accumulated on top of the block with Bob's transaction and she can gain more confidence in the immutability of the transaction.

2.4 Quantum Computing

In this section we offer some insight into Quantum Computing, its mechanisms, phenomena, the current state of physical realizations, and the direction it is heading towards.

Even though the idea that atoms or photons could be manipulated to perform highly efficient parallel computations was first formulated in 1959 by Richard Feynman [26], the idea of quantum computers became of (somewhat) wide interest only recently (1994), when Peter Shor developed a polynomial time algorithm for factoring large integers [60]. This is one of the major threats to current cryptographic systems (RSA and Bitcoin's ECDSA included) which rely on the hidden subgroup problem on finite abelian groups.

The power of quantum computing stems from several phenomena and laws of quantum mechanics that are fundamentally different from those encountered in classical computing. In order to understand complex probability amplitudes, quantum interference, quantum parallelism, quantum entanglement, and the unitarity of quantum evolution, one has to understand several basic principles on which quantum mechanics is based. To use these features in the design of quantum algorithms, networks, and processors, we need to study the basics of Hilbert space formalism, which represents the mathematical framework used in quantum mechanics.

In general, quantum algorithms work by preparing the state on which the computations are performed as a superposition of more (or all) possible classical states, thus computing all the possible solutions in only one step. The major problem quantum algorithms have to overcome is wave function collapse, i.e. when the result of a computation is measured, the superposition of states is collapsed and only one of the results is observed, the others being lost forever. Hence, quantum computations try to use the underlying structure of the problem and manipulate the superimposed state in order to increase the likelihood of a certain outcome which can be interpreted deterministically and yield the result wanted.

Motivation for Quantum Computing (QC)

Before diving into the mathematics of quantum mechanics, we note the motivation behind quantum computing and we argue that it is only a matter of "when" and not "if" this technology will become functional. The challenges QC poses to current cryptographic systems must be addressed and resolved before it reaches a stage of maturity.

Current, classical, computers are getting smaller and faster, approximately satisfying Moore's Law. Unfortunately, this trend will stop very soon as physical laws impose limits on how much smaller we can build transistors. Intel currently produces transistors only 14 nanometres wide [41] while an atom is about 0.5 nanometres. We are getting close to a scale where the classical laws of physics simply do not apply any more and quantum mechanics takes over. Phenomena such as quantum tunnelling will render transistors useless as electrons will pass right trough them. To overcome this barrier, we need to drastically change the paradigm of building a com-

puter and make use of the laws of physics at this extreme scale. In fact, according to our current knowledge, the physical world is fundamentally driven by quantum mechanics. All computers are physical devices and computations are physical processes. However, all classical computers and models of computers rely solely on classical mechanics and while their performance is impressive, they do not tap into the full potential of information processing power that our world offers. As such, it is a fundamental duty to deepen our knowledge and understanding of the laws of quantum mechanics and how they can be used to overcome the computational limitations of classical mechanics.

2.4.1 Mathematical Framework

This section aims to present the mathematical tools needed to model, operate, and measure quantum systems. For a deeper understanding of the theory presented here, please refer to Jozef Gruska's book "Quantum Computing" [29].

Hilbert Spaces

The set of all possible (pure) states of a quantum system constitutes a so called Hilbert space. This formalism is the basic framework for rigorous, precise definitions and for the study of quantum mechanical concepts, phenomena, algorithms, and processes.

Definition 19 (Vector (linear) Space)
A vector (linear) space S, with a carrier H, over a field K is an algebra $S = \langle H, +, ^{-1}, 0, K, +_f, \times_f, 0, 1, \cdot\rangle$ such that $\langle H, +, ^{-1}, 0\rangle$ is a commutative group, $K = \langle K, +_f, \times_f, 0, 1\rangle$ is a field, and $\cdot : K \times H \to H$ is a scalar multiplication satisfying the following axioms for any $a, b \in K$ and $\phi, \psi \in H$:

1. $a \cdot (\phi + \psi) = a \cdot \phi + a \cdot \psi, (a +_f b) \cdot \phi = a \cdot \phi + b \cdot$ *(distributive laws)*
2. $(a \cdot (b \cdot \phi)) = (a \times_f b) \cdot \phi$
3. $1 \cdot \phi = \phi$.

Definition 20 (Inner-product Space)
An inner-product space H is a complex vector space, equipped with an inner product operation $\langle \cdot | \cdot \rangle : H \times H \to C$ satisfying the following axioms for any vectors ϕ, $\psi, \phi_1, \phi_2 \in H$, and any complex numbers $c_1, c_2 \in C$.

1. $\langle \phi | \psi \rangle = \langle \psi | \phi \rangle^\star$, *where $\star$ denotes the conjugate of a complex number, i.e.* $x^\star = (a + bi)^\star = a - bi$.
2. $\langle \psi | \psi \rangle \geq 0$ *and* $\langle \psi | \psi \rangle = 0$ *if and only if* $\psi = 0$,
3. $\langle \psi | c_1\phi_1 + c_2\phi_2 \rangle = c_1 \langle \psi | \phi_1 \rangle + c_2 \langle \psi | \phi_2 \rangle$.

The inner product introduces on H the norm

$$\|\phi\|_H = \sqrt{\langle\phi|\phi\rangle}$$

and the metric

$$dist_H(\phi, \psi) = \|\phi - \psi\|$$

$\|\phi\|$ is sometimes called the length of vector ϕ and $\langle\phi|\phi\rangle$ is called the squared length.

Definition 21 (n-dimensional Complex Inner-product Space)
If $H = C^n$ for a fixed n and the inner product is defined by

$$\langle(x_1, \ldots, x_n)|(y_1, \ldots, y_n)\rangle = \sum_{i=1}^{n} x_i^\star y_i$$

*then we speak about the **n-dimensional complex inner-product space**.*

For our purposes, we will assume the above definition for the $\langle\cdot|\cdot\rangle$ operator.

Dirac Notation introduces $\langle\psi|$ and $|\psi\rangle$ called bra and ket vectors respectively. For n-dimensional complex Hilbert spaces a ket $|\psi\rangle$ can be considered as an n-dimensional column vector and a bra $\langle\phi|$ as an n-dimensional row vector. As such, the **scalar product** $\langle\phi|\psi\rangle$ is the result of a usual row vector $\times$ column vector product, i.e. a complex number and the **tensor product** $|\psi\rangle\langle\phi|$ is the result of a usual "column vector $\times$ row vector" product, i.e. a matrix. For an indexed set of vectors $\{x_i\}$ we sometimes write $|i\rangle$ or $\langle i|$ to represent the bra or ket vector of the i-th element in the set.

Definition 22 (Hilbert Space)
*An inner-product space H is called **complete** if for any sequence $\{\phi_i\}_{i=1}^{\infty}$ with $\phi_i \in H$, and with the property $\lim_{i,j\to\infty} \|\phi_i - \phi_j\| = 0$ there is a unique element $\phi \in H$ such that $\lim_{i\to\infty} \|\phi - \phi_i\| = 0$. A complete inner-product space is called a **Hilbert space**. The elements of H are usually called vectors and those elements with norm 1 denote (pure) states.*

Remark 11 (Interpretation of Definition 22)
An inner-product space or pre-Hilbert space is complete if any Cauchy sequence converges with respect to the norm to some element in the space. A Cauchy sequence is a sequence whose elements become arbitrarily close to each other as the sequence progresses.

Orthonormal Basis of Hilbert Spaces

The concept of orthogonality is one of the key constructs in the theory of Hilbert Spaces.

Definition 23 (Orthogonal and Orthonormal)
*Two vectors ϕ and ψ of a Hilbert space H are called orthogonal if $\langle\phi|\psi\rangle = 0$. A set $S \subseteq H$ is **orthogonal** if any two of its elements are orthogonal. S is **orthonormal** if it is orthogonal and all its elements have norm 1.*

Remark 12 (Interpretation of Orthogonal Vectors)
Orthogonal vectors are linearly independent vectors, i.e.

$$\sum_i \lambda_i x_i = 0 \ \textit{implies that} \ \lambda_i = 0 \ \forall i$$

As such, orthogonal vectors can represent events that are independent of each other; e.g. all positions a particle can be located in, all the possible polarizations of a photon.

Orthogonality plays an important role in quantum computing as whenever a measurement is performed on a quantum system, the quantum states that lead to distinguishable outcomes have to be mutually orthogonal.

Definition 24 (Orthonormal Basis of Hilbert Spaces)
An orthonormal set $B \subseteq H$ is an orthonormal basis for H if none of its proper supersets is orthonormal.

Remark 13 (Implications of Definition 24)

1. *A base must contain a maximal number of linearly independent vectors as otherwise a superset of it would also be orthogonal.*
2. *All bases of a Hilbert space H have the same cardinality, also called the dimension of H. Hence, all Hilbert spaces of the same dimension are isomorphic.*
3. *If basis $B = \{\phi_i\}_{i=1}^n$ is a base of an n-dimensional Hilbert space, then any vector ψ can be written as a linear combination of columns of B i.e. $\psi = \sum_{i=1}^n \alpha_i \phi_i$. The vector $(\alpha_1, \ldots, \alpha_n)$ is called the representation of ψ in base B.*

Definition 25 (Linear Functional)
A linear functional on a vector space H over a field K is a map $f : H \to K$ such that:

1. $f(x + y) = f(x) + f(y)$
2. $f(\alpha x) = \alpha f(x) \ \forall x, y \in H, \alpha \in K$

The space of all linear functionals on H, is itself a vector space called the dual space of H and is denoted $H^\star$.

More specifically, for a Hilbert space H the following theorem holds.

Theorem 3 (Riesz Representation)
For each continuous functional $f \in H^\star, f : H \to C$ there exists a unique vector $\phi_f \in H$ such that $f(\psi) = \langle \phi_f | \psi \rangle$ for any $\psi \in H$.

This theorem establishes a bijection between H and $H^\star$, which implies that $H^\star$ is isomorphic to H; in particular $(C^n)^\star = C^n$.

Definition 26 (Linear Operator)
A map $T : V \to W$ between two vector spaces V and W is called a linear map if:

1. $T(x + y) = T(x) + T(y)$
2. $T(\alpha x) = \alpha T(x)$

for all $x, y \in V$ and all $\alpha \in K$.
For $V = W$ we talk about a (linear) operator on V.

Similar to linear functionals, which can be represented as vectors which map other vectors to complex numbers, linear operators can be represented as matrices that map vectors to other vectors. For ease of notation we will usually denote the matrix of an operator A with A also. As such the application $A(x)$ becomes $Ax = (A_{ij})(x_i) = \sum_i A_{ij}x_i$. Furthermore, the composition of two operators $A \circ B$ is just matrix multiplication AB.

Definition 27 (Adjoint Matrix/Operator)
*For a matrix $T = (T_{ij})$, the **adjoint** matrix is $T^\dagger = (T^T)^\star$. Furthermore, if $T = T^\dagger$, then T is called **self-adjoint** and is a Hermitian matrix.*

It follows from the above definition that $\langle T^\dagger\psi|\phi\rangle = \langle\psi|T\phi\rangle$.

Definition 28 (Unitary Matrix/Operator)
A matrix U is called unitary if and only if $U^\dagger = U^{-1}$

Definition 29 (Tensor Product)
The tensor product "$\otimes$" is defined for vectors as:

$$x \otimes y = (x_1, \ldots, x_m) \otimes (y_1, \ldots y_n) = (x_1y_1, \ldots, x_1y_n, x_my_1, ..., x_my_n)$$

and for matrices as:

$$A \otimes B = \begin{pmatrix} a_{11} & a_{12} & \ldots \\ a_{21} & a_{22} & \\ \vdots & & \ddots \end{pmatrix} \otimes \begin{pmatrix} b_{11} & b_{12} & \ldots \\ b_{21} & b_{22} & \\ \vdots & & \ddots \end{pmatrix} = \begin{pmatrix} a_{11}B & a_{12}B & \ldots \\ a_{21}B & a_{22}B & \\ \vdots & & \ddots \end{pmatrix}$$

Remark 14 (Properties of the Tensor Product)
The tensor product exhibits a number of nice properties:

1. *Bi-linearity, which applies to both vectors and matrices: $(\alpha v + \alpha' v') \otimes (\beta w + \beta' w') = \alpha\beta(v \otimes w) + \alpha'\beta(v' \otimes w) + \alpha\beta'(v \otimes w') + \alpha'\beta'(v' \otimes w')$.*
2. *$(M \otimes N)(v \otimes w) = (Mv) \otimes (Nw)$ and $(M \otimes N)(M' \otimes N') = (MM') \otimes (NN')$.*
3. *If M and N are unitary (or invertible) so is $M \otimes N$.*
4. *$(M \otimes N)^T = M^T \otimes N^T$.*

2.4.2 Basics of Quantum Theory

We are finally in a position to present the basic constructs of a quantum system.

Definition 30 (Quantum Postulates)

1. *The **state** of an (isolated) quantum system is represented by a (normalised) vector in a complex Hilbert space H.*
2. *An **observable** is represented by a self-adjoint matrix (operator) acting on a Hilbert space.*
3. *The expected result (average) when **measuring** observable A of a system in state $|x\rangle \in H$ is given by: $\langle A\rangle_x = \langle x| A |x\rangle = \langle x|Ax\rangle$.*
4. *The only possible results are eigen-values λ_i of A.*
5. *The **probability of measuring** λ_n in state $|x\rangle$ is given by: $Pr(A = \lambda_n|x) = \langle x|P_n x\rangle$ where $P_n = |\lambda_n\rangle \langle\lambda_n|$ is the orthogonal projection onto the space generated by eigen-vector $|\lambda_n\rangle$ of A.*

Quantum State

The state of a quantum system offers a complete description of the internal "memory" of the quantum computer at any point during the computation. These states are modelled as vectors of a Hilbert space.

One can say that to each isolated quantum system corresponds a Hilbert space. Other interpretations go further by claiming that reality on the quantum level does not exist and only emerges when measurement is done. Everything we know about the quantum level are computational procedures (expressed in terms of Hilbert space mathematics) that compute the evolution of quantum systems and probabilities of the measurement outcomes.

Using the Hilbert space formalism, any state X of the system can be represented as a ket-vector or a bra-vector. Any state X can be written in terms of any basis B:

$$|X\rangle = \sum_{i\in B} |i\rangle \langle i|X\rangle = \sum_{i\in B} \alpha_i |i\rangle \quad \texttt{and} \quad \langle X| = \sum_{i\in B} \alpha_i^\star \langle i|$$

The complex numbers $\langle i|X\rangle$ are called probability amplitudes and represent the probability that when measuring the state with respect to basis B, the qubit will collapse to $|i\rangle$.

Qubits

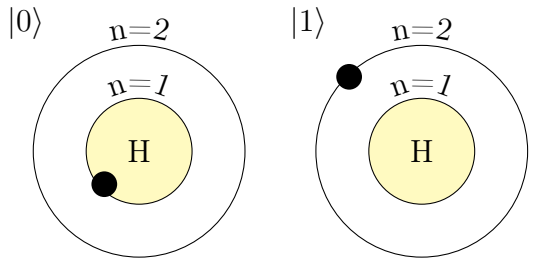
A **quantum state** is represented by a normalised vector in C^n. A **qubit** is a two-dimensional quantum state in C^2. We can represent the coordinates of a qubit with respect to the orthonormal basis

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \{|0\rangle, |1\rangle\}, \quad \texttt{where} \quad |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

also called a standard basis, as follows:

$$|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha\begin{pmatrix}1\\ 0\end{pmatrix} + \beta\begin{pmatrix}0\\ 1\end{pmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers with $\|\alpha\|^2 + \|\beta\|^2 = 1$. Note that all quantum states are normalised, so $\|\psi\| = \langle\psi|\psi\rangle = 1$.



Any two state quantum effect can be used to create physical implementations of qubits. For example, an electron that can be on either of two energy levels in a hidrogen atom (see figure on the left), or the vertical and horizontal polarizations of a photon.

Measurements and Observables

Before quantum mechanics was introduced, it was taken for granted that measuring something represents gaining knowledge of a pre-existing state. In other words, it was thought that measurement does not affect the state of the system in any way. The quantum interpretation is that measurement cannot be done without affecting the system.

Example 1 (Measurement affects the system it measures)
This concept can be understood even in a classical world. When one measures the speed of a an object (e.g. car), what usually happens is that a wave (eg. radio, light) is emitted towards the object and the reflection of the wave is caught with a receiver. As the object is moving, the wavelength will be different after reflection and the speed of the object can be determined. It would appear that the object was not affected at all during this measurement, but, in fact, when particles reflect off the surface of the object they exert a small pressure on the object which slightly changes its position. In a classical world this effect is negligible, but when we talk about microscopic entities (e.g. atoms, photons, electrons) this intervention to the system completely changes the output of the measurement.

In order to extract information from a quantum system we have to observe the system i.e. to perform a measurement of the system. Thus, a quantum test consists of two phases. During the preparation phase, a deterministic physical system is set up; both the observable represented by the testing instrument and the state to-be-measured are fixed. The second phase is the measuring itself, which is a probabilistic process. Only one of the potential outcomes is produced and its probability can be computed using deterministic rules.

Observables are properties of the physical system that can be measured. While in classical physics these can be position, speed, or size, in quantum theory an observable is a self-adjoint operator. The outcome of the measurement of a quantum

state $|\psi\rangle$ with respect to an observable A is one of the eigenvalues of A and the effect of such a measurement is a collapse of $|\psi\rangle$ into a state which is represented by the corresponding eigenvector.

Evolution of a Quantum System

During the evolution of a quantum system, some transformations of the initial state are performed such that the state can be manipulated to amplify certain probability amplitudes. Some physical quantum gate A is represented mathematically by a linear operator A which transforms each complex vector to another complex vector that represents the new state. Since all quantum states should be normalised vectors, the condition for an operator to represent a quantum transformation is to be unitary. This implies that any transformation is also reversible using the adjoint operator $A^\star$.

Compound Quantum Systems

When designing quantum circuits, it is often desirable to treat the system as a composition of two other systems. For example, we have to be able to put together multiple qubits to build a much more complex system. Often, we also need to decompose the system back into individual qubits in order to measure only parts of the system.

Definition 31 (Compound Quantum Systems)
Let S_1 and S_2 be two quantum systems and let H_1 and H_2 be the corresponding Hilbert spaces. Let the compound system of S_1 and S_2 be S. It holds:

1. *The tensor product $H = H_1 \otimes H_2$ is the Hilbert space associated to S.*
2. *Observables of S are self-adjoint operators in H.*
3. *Evolutions in S are determined by unitary operators of H.*

A compound state $|\phi\rangle \in C^{2^n}$ is said to be **separable** if and only if it can be written as the tensor product of some other states. In case this cannot be done, the state is said to be **entangled** and it exhibits some extremely counter-intuitive properties. For example, we know that the state collapses when we measure it, but what would happen if we only measure one of the particles from a fully entangled state. Quantum laws dictate that the whole state would collapse to another superposition where the qubit observer is fixed and all the other qubits are fully entangled. This means that observing a certain particle somehow affects the behaviour of other particles too. In fact, the particles could have been moved to very far locations before observation and the collapse of the un-measured particles would still happen simultaneously with the measurement. Although very un-intuitive, this is the nature of the world.

Quantum Gates

Likewise classical computers, the basic elements of a quantum computer are qubits, quantum registers, quantum gates and quantum networks. However, the properties of these elements are very different from their classical counterparts. A qubit can be in an infinite number of states and a quantum register composed of n qubits can be in any superposition of the 2^n basis states, at the same time. This enables massive parallelism, which can be exploited by performing transformations using quantum gates (implementations of self-adjoint operators). In Figure 2.6 are some examples of common 1 qubit quantum gates and their matrix representation.

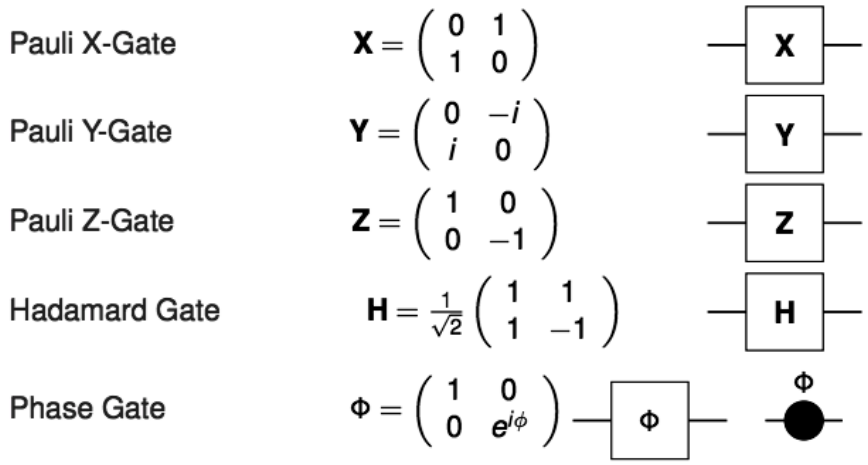
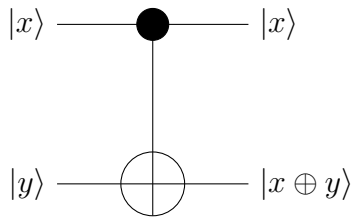


Figure 2.6: Some examples of 1 qubit quantum gates. In fact, from these gates we can construct any other unitary matrix.

Furthermore, we give the representation of the much utilized controlled not gate. The operation $\oplus$ is defined as addition mod 2.



$$\texttt{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.7: The CNOT gate. Whenever $|x\rangle = |1\rangle$, the lower qubit is negated.

Example 2 (Swapping two qubits)
Mathematically the swapping of two qubits (see Figure 2.8 below) can be achieved as follows. We start from the initial state:

$$|x\rangle\,|y\rangle = (\alpha\,|0\rangle + \beta\,|1\rangle)(\alpha'\,|0\rangle + \beta'\,|1\rangle) = \alpha\alpha'\,|00\rangle + \beta\alpha'\,|10\rangle + \alpha\beta'\,|01\rangle + \beta\beta'\,|11\rangle$$

$$\begin{aligned} \textit{we apply the first C-NOT gate} &\rightarrow \alpha\alpha' |00\rangle + \beta\alpha' |11\rangle + \alpha\beta' |01\rangle + \beta\beta' |10\rangle \\ \textit{we apply the second C-NOT gate} &\rightarrow \alpha\alpha' |00\rangle + \beta\alpha' |01\rangle + \alpha\beta' |11\rangle + \beta\beta' |10\rangle \\ \textit{we apply the third C-NOT gate} &\rightarrow \alpha\alpha' |00\rangle + \beta\alpha' |01\rangle + \alpha\beta' |10\rangle + \beta\beta' |11\rangle \end{aligned}$$

Finally the result can be factored as such:

$$\equiv \alpha' |0\rangle (\alpha |0\rangle + \beta |1\rangle) + \beta' |1\rangle (\alpha |0\rangle + \beta |1\rangle) \equiv (\alpha' |0\rangle + \beta' |1\rangle)(\alpha |0\rangle + \beta |1\rangle) \equiv |y\rangle |x\rangle$$

Another way to see this would be to use matrix multiplication using the matrix representation of the C-NOT gate.

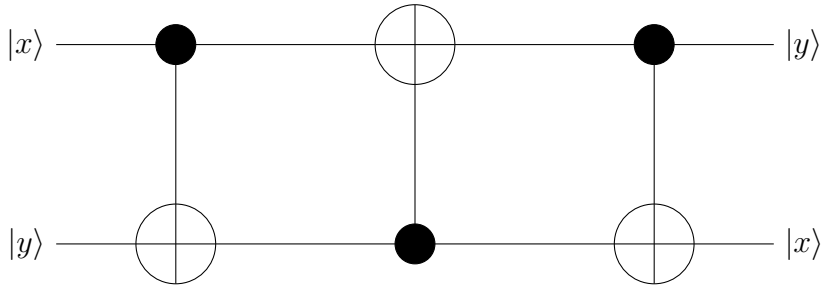


Figure 2.8: Graphical representation of the circuit for swapping two qubits.

2.4.3 Quantum Algorithms

Quantum algorithms outperform their classical counterparts for some classes of problems by using quantum phenomena. In general, they prepare a state in quantum superposition and move through entangled states, thus being able to leverage the power of quantum parallelism. By a single application of a unitary operator, quantum algorithms can perform 2^n (where n is the number of qubits) classical computations on basis states. Note that in quantum registers the amount of parallelism increases exponentially with the size of the system, only requiring a linear increase of physical space.

For our paper we will only present the algorithms which affect Bitcoin and the building blocks needed to create them.

Quantum Fourier Transform (QFT)

Classical Fourier transforms are regarded as powerful mathematical tools that map functions of period r to functions with non-zero values only at the multiples of the frequency $1/r$. Thus, a quantum analogue was developed that can perform the same task in polynomial time.

Definition 32 (QFT)
QFT with the base q (or in the group Z_q) is the unitary transformation:

$$QFT_q : |a\rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} e^{2\pi iab/q} |b\rangle$$

and has the following unitary matrix:

$$F_q = \frac{1}{\sqrt{q}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{q-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(q-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{q-1} & \omega^{2(q-1)} & \dots & \omega^{(q-1)^2} \end{pmatrix}$$

where $\omega = e^{2\pi i/q}$ is the q-th root of unity.

The physical realization of this algorithm is obtained by composing multiple Hadamard and phase shift gates. In the following representation $X_j = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^j} \end{pmatrix}$.

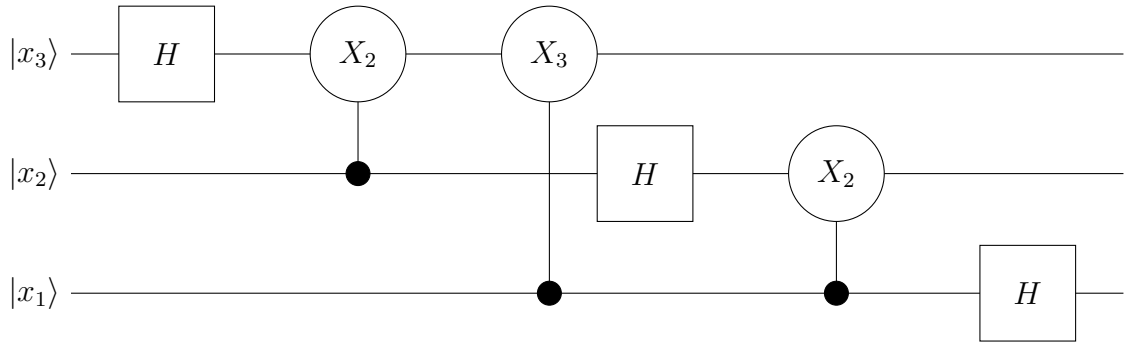


Figure 2.9: Quantum network implementation for the Quantum Fourier Transform for $n = 3$.

Shor's Algorithm

Shors algorithm can solve in polynomial time the hard problems of integer factorization and discrete logarithms. The first part of the solution can be run on classical computers and reduces the problem of factoring large integers to finding the period of a function. The second part makes use of the QFT to extract the exact period. However, for our purposes we are more interested in how we can solve the ECDLP not integer factorization. For simplicity, we will only show how Shor's algorithm is implemented for a general discrete logarithm problem as the elliptic curve specialization requires implementing the elliptic curve group law (addition of points). This can be implemented easily as it reduces to simple operations [60], but it would exceed the space limitations of this paper.

Definition 33 (Discrete Logarithm Problem)
Determine an r such that $g^r \equiv x \mod p$ given a prime p, a generator g of the multiplicative group $Z_p^\star$ and some $x \in (0, p)$.

We first start by preparing the quantum register in the following form:

$$|\phi_0\rangle = |x\rangle\,|x\rangle\,|0\rangle\,|0\rangle\,|0\rangle = |x, g, 0, 0, 0\rangle$$

. We apply the QFT_{p-1} two times to modify the third and fourth registers and obtain

$$|\phi_1\rangle = \frac{1}{p-1}\sum_{a=0}^{p-2}\sum_{b=0}^{p-2}|x, g, a, b, 0\rangle$$

a uniform distribution of all pairs (a, b) with $0 \leq a, b \leq p-2$. Next we apply the following uniform transformation mapping (unitary operator): $(x, g, a, b, 0) \rightarrow (x, g, a, b, g^a x^{-b} \mod p)$ and have

$$|\phi_2\rangle = \frac{1}{p-1}\sum_{a=0}^{p-2}\sum_{b=0}^{p-2}|x, g, a, b, g^a x^{-b} \mod p\rangle\,.$$

From now on, the computation leaves the registers on which x and g reside untouched, so we will not include them anymore until the end. We apply the QFT_{p-1} twice on the registers of a and b again. First we map $a \rightarrow c$ with amplitude $\frac{1}{p-1}e^{\frac{2\pi i}{p-1}ac}$, and then we map $b \rightarrow d$ with amplitude $\frac{1}{p-1}e^{\frac{2\pi i}{p-1}bd}$. The result is

$$|\phi_3\rangle = \frac{1}{(p-1)^2}\sum_{a,b,c,d=0}^{p-2} e^{\frac{2\pi i}{p-1}(ac+bd)}\,|c, d, g^a x^{-b} \mod p\rangle\,.$$

Now let us compute the probability amplitude of the states that are of interest to us, i.e. the states for which $x = g^r \mod p$. The probability is the square of the sum of all the corresponding probability amplitudes. Substituting x in the last register gives $g^a(g^r)^{-b} \mod p = g^{a-rb} \mod p$. Further, we can use the fact that for a prime p the following holds: $g^{p-1} \equiv 1 \mod p$. As such, the following implication holds $a \equiv b \mod (p1) \rightarrow g^a \equiv g^b \mod p$. Now if we define $a - rb \equiv k \mod (p-1)$, the final register becomes $g^k \mod p$.

Therefore, we measure the state $y \equiv g^k \mod p$ with probability

$$\left\|\frac{1}{(p-1)^2}\sum_{a,b,c,d=0}^{p-2} e^{\frac{2\pi i}{p-1}(ac+bd)}\right\|^2 \quad \texttt{where}\ a - rb = k$$

$$\equiv \left\|\frac{1}{(p-1)^2}\sum_{b,c,d=0}^{p-2} e^{\frac{2\pi i}{p-1}(kc+b(d+rc))}\right\|^2$$

When $d + rc \not\equiv 0 \mod (p-1)$, the above expression is over a set of $(p-1)$st (even) roots of unity that cancel each other, so the probability of this event is 0.

As such, it must be that $d + rc \equiv 0 \mod (p-1))$ and the above expression becomes $(p-1)^{-1}e^{\frac{2\pi ikc}{p-1}}$. Because the expression does not depend on b, the probability is $(p-1)^{-2}$.

Hence, we can measure the registers of c and d to retrieve some $c, d < p-1$ that satisfy: $d \equiv -rc \mod (p-1)$. From this we can extract $r = -\frac{d}{c} \mod (p-1)$, provided that $gcd(c, p-1) = 1$.

It can be proven that the probability that $gcd(c, p-1) = 1$ is $\Omega(\frac{1}{\lg p})$. Therefore, the number of number of tries we have to do to obtain this event and be able to retrieve r is polynomial in $\lg p$.

This demonstration proves that we can break ECDLP in time polynomial with n, the size of the key. To achieve this, it has been calculated that approximately $6n$ qubits are needed [53]. These calculations are made specifically for the elliptic curve discrete logarithm, with point addition implemented and all details accounted. Thus, quantum computers just need a small leap in order to be able to break ECDSA and RSA.

Grover's Algorithm

Definition 34 (Unstructured Search Problem)
Given a set $\{x_i\}_{1\leq i\leq N}$, and a function $f : \{0,1\}^N \to \{0,1\}$ with the property that $\frac{\sum_{i=1}^{N} f(x_i)}{N}$ is relatively small. Return one item x for which $f(x) = 1$. In other words: Find an item with some un-indexed, rare property in an unordered structure.

Remark 15
In fact, this class of problems covers all the hash functions and many more problems.

Classically this problem can only be perfectly solved in $O(N)$ time which means exponential in terms of the number of bits in N. Grover's quantum search algorithm can find a desired value x in approximately $O(\sqrt{N})$, which means a quadratic speedup. To design such an algorithm we first need to create a unitary operator U_f that can compute f and select the elements we want. In fact, it can be proven that we can construct U_f which maps $|x\rangle \to (-1)^{f(x)} |x\rangle$. This essentially means that we will negate (invert) only those basis states that correspond to wanted solutions.

Let $N = 2^n$ and assume there is only one state we are looking for, i.e. $f(x) = 1$ holds for exactly one state. The **Grover quantum search algorithm** is:

1. Initialize the system with an n-dimensional vector $|0\rangle_n$.
2. Using the n-dimensional Hadamard gate H_n, prepare the register in a super position of all possible inputs.
$$|0\rangle_n \to |\phi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$
3. Apply U_f to $|\phi\rangle$.
$$|\phi_1\rangle \to |\phi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle$$
4. Apply the inversion about average operator $D_n = -H_n R_n^1 H_n$, to ϕ_2.
5. Iterate steps 3 and 4 (the Grover iterate), $\lceil \frac{\pi}{4}\sqrt{2^n} \rceil$ times.

6. Measure the register, and retrieve one of the wanted values.

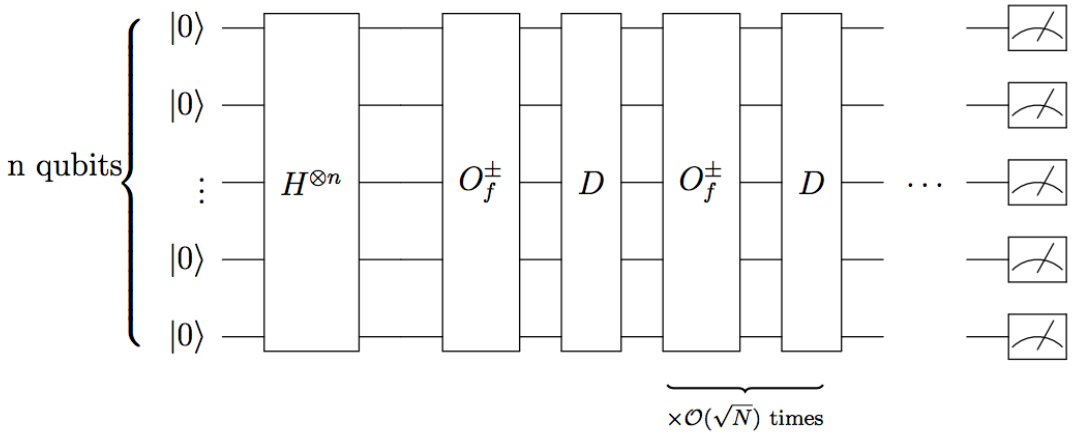


Figure 2.10: A network implementing Grover's quantum search algorithm.

The Grover Iterate is successively amplifying the probability amplitudes of the wanted states. When the probability is maximal the measurement is done and the wanted solution is represented by the collapsed state.

For the general case, where we have k possible wanted states instead of 1, the runtime complexity is $O(\sqrt{\frac{N}{k}})$.

2.5 Post-Quantum Cryptography

Post-quantum cryptography is a new branch of cryptography interested in a suite of algorithms which are believed to be secure even against attackers equipped with quantum computers [16]. There have been multiple proposals of cryptographic systems which are believed to withstand attacks by quantum computers. As each of the following cryptographic systems is a very complex and vast domain in itself, we are not aiming to cover the theoretical basis underlying these concepts.

This section's purpose is to demonstrate that cryptographic schemes that can withstand quantum computers exist and can be implemented.

Code-based cryptography relies on the intractability of decoding unknown linear error-correcting codes [59]. McEliece used the algebraic properties of Goppa codes and proposed the first such system [42], which took his name. In general all code-based cryptographic systems rely on the concept of permutation equivalence.

Definition 35 (Permutation Equivalence)
Two codes $C, C' \in F_q^n$, with their respective generator matrices G, G', are called ***permutation equivalent*** *provided there is a coordinate permutation f, which sends C to C', i.e.*

$$C' = \{f(x) : x \in C\}$$

It follows then, that there must be a non-singular matrix S and a permutation matrix P such that: $G = SG'P$

Thus, in such cryptosystems, we can construct private and public keys as follows:

1. The private key is composed of:
   (a) A random non-singular matrix S.
   (b) An efficient decoding algorithm for some random irreducible binary Goppa code C with its generator matrix G.
   (c) A random permutation matrix P.
2. The public key is $G' = SGP$.

The trapdoor function of this cryptosystem is the knowledge of an efficient error correcting algorithm for the chosen code (which is available for any Goppa code) and of the permutation P [50].

Cryptosystems relying on linear codes perform very efficiently when generating the keys, encrypting, and decrypting, but their drawback is the very large size of the public keys.

Hash-based cryptography relies solely on the security of cryptographic hash functions, which, as mentioned, are not drastically weakened by QC. Merkle [46] was the first to propose hash-based digital signatures by building on the concept of one-time signature schemes such as Lamport's signature scheme [38]. As the name suggests, one-time signature schemes have a major drawback, i.e. the private key is gradually revealed to the public as more and more messages are signed. In particular, approximately half of the private key is revealed every time a message is signed. To overcome this problem, Merkle used the ideas of hash trees to build a tree of public keys, while the private keys are kept in the same way as before. This extends the number of messages that can be signed without revealing enough information to break the scheme to the number of leaves in the Merkle tree of public keys.

Although the public key size is now quite small, the key generation algorithm requires creating as many key-pairs as many messages we want to sign, which is why this system is not used in practice.

Lattice-based cryptography is based on the hardness of lattice problems such as approximating the closest vector problem in a lattice [49]. A lattice is a set of points in n-dimensional space with a periodic structure [50], or more formally:

Definition 36
Given n linearly independent vectors $b_1, \ldots, b_n \in \mathcal{R}^n$, the lattice generated by this basis is the set of vectors:

$$\mathcal{L}(b_1, \ldots, b_n) = \left\{ \sum_{i=1}^{n} x_i b_i : x_i \in \mathcal{Z} \right\}$$

Furthermore, it is intuitive that for any lattice there are more than one basis that correctly describe it. However, some of these basis will allow the computations of vectors in the lattice to be relatively cheap computationally, while others will require very expensive calculations.

Therefore, a public-key cryptosystem can be designed where the private key is a "good" basis, which consists of short vectors, almost orthogonal vectors and the public key is a "bad" basis for the same lattice.

Although lattice problems exhibit a periodic underlying structure, which is what quantum algorithms usually look for, attempts to create such algorithms date since Shors factoring technique and none have been successful in applying the same strategies.

For the purposes of our paper, it is important that the Bitcoin community agrees on and implements an appropriate alternative (or perhaps more than one) to replace Elliptic Curve Cryptography as the basis for digital signatures of transactions.

Section 3: Post-Quantum Bitcoin

In this section, we analyse the impact a malicious adversary in possession of a quantum computer can pose to Bitcoin.

Nowadays, multiple companies have shown major interest in quantum computing and extensive research on error-correction, physical implementations, theoretical algorithms and much more is going on [69, 22, 68]. Considering the number of new players that enter this growing domain, it seems increasingly probable that powerful quantum computing will emerge in the near future. A sudden improvement in the approach of physical implementations might lead to a fast quantum computer appearing virtually overnight.

We will present a few attack strategies that become possible with quantum computing and the extent to which they can be used. Note that we do not discuss the possibility of using Grover's quantum search to retrieve the public key from a P2PKH or P2SH challengeScript in our work, as the achieved speed-up is merely quadratic and can be mitigated by increasing the key size [14].

3.1 Attacks on Proof Of Work (PoW)

Attacks on Bitcoin's Proof Of Work (PoW) have the purpose of gaining an unfair advantage when mining such that the attacker can produce valid blocks considerably faster than the rest of the network. As such, an adversary can successfully rewrite the blockchain history by publishing a chain of blocks that links to an older block instead of the tip of the blockchain. In order for such an attack to be successful, the attacker needs to extend his chain until it exceeds the length of the main chain, thus convincing the rest of the nodes to abandon the original main chain and start mining on the malicious chain. This is especially difficult since while an attacker is mining blocks, the rest of the network keeps mining on the main chain, adding a block every 10 minutes.

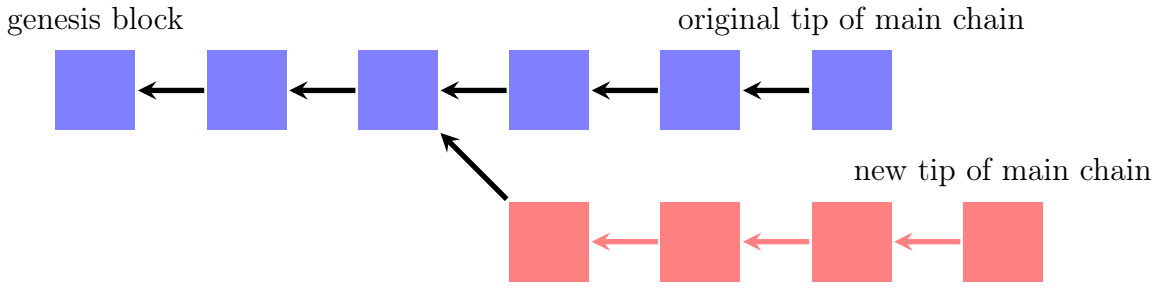


Figure 3.1: The red attacker mines blocks that do not link to the tip of the blockchain and outperforms the honest miners (in blue) so his chain will become the new main chain.

As explained in the Section 2.3 of this paper, Proof-of-Work (PoW) is a hard cryptographic puzzle based on hashing. For the purpose of this attack we will ignore the Bitcoin specific details of POW and, without loss of generality, we can simplify the challenge a miner needs to solve to the following problem:

Definition 37 (Simplified Proof-Of-Work)
Given:

1. *A predefined cryptographic hash function H that takes an arbitrary long bitstring as input and outputs an n-bit bitstring of deterministically generated data.*
2. *A bitstring m of some length.*
3. *A target difficulty expressed as a n-bit bitstring.*

Find a nonce n such that:

$$H(m||n) < t$$

where "$||$" represents the concatenation operation and "$<$" represents the "`lower than`" comparison operation for n-bit bitstrings.

Remark 16
As H is a predefined function, both a classical miner and a quantum capable miner can pre-compute information about this function to speed-up the computation. In fact, classical miners make use of application-specific integrated circuits (ASIC) which they design specifically to implement function H, thus drastically improving their hash-rates.

Given the current state of quantum computing research, the most suitable tool for attacking Proof of Work is actually Grover's quantum search algorithm that we described in Section 2.4. Grover's algorithm was designed exactly for this type of unstructured search problems. This provides a quadratic speed-up compared to a classical computer, which should give a decent advantage to the adversary. In fact, the multiple acceptable solutions version of Grover's algorithm should be employed here as any $H(m||n) < t$ is acceptable. Thus, the worst case runtime complexity is actually $O(\sqrt{\frac{2^n}{t}})$. Compared to the classical worst case time complexity of $O(2^n - t)$, Grover's algorithm offers a considerable speed-up. However, current miners use parallel computations on optimised hardware (ASICs) and achieve extremely high hashrates which could definitely not be matched by a quantum computer, given the clock cycles predicted [14]. Moreover, the quantum circuits will need to load the message m before starting the computation, and interfacing with classical computers is currently one of the bottlenecks of QCs. It is hence difficult to predict if and when they will be reliable and fast enough to outperform classical highly parallel computations.

To this end, we assume early generations of QCs will not be capable of outperforming classical miners, so no advantage can be gained. Furthermore, once quantum computations become scalable enough to make mining profitable, the technology would probably also be widespread, so a quick adoption among miners can be

expected. Hence, an equilibrium will be achieved as the network difficulty adjusts. For these reasons, in this paper, we do not aim at addressing potential vulnerabilities rooted in Bitcoin's Proof of Work but rather the weaknesses of the embedded transaction verification mechanism.

3.2 Attacks on ECDSA

Once efficient quantum computers with internal states comprised of many qubits are implemented, the underlying cryptographic guarantees of Bitcoin's ECDSA can be challenged. As mentioned in Section 2.4, an attacker with a quantum computer of about $6 * n \approx 1500$ (given the ECDSA public key is a 256 bit string) qubits [54, 63] can use Shor's algorithm to solve the ECDLP and compute an ECDSA private key given the public key. Once an adversary obtains a private key, he is indistinguishable from the original owner so he is in full control of the funds.

3.2.1 Public key unveiling

In the following paragraphs we highlight why the community should be concerned about exposing their public keys. Although a discussion has to be made regarding the speed a quantum computer can achieve when computing private keys from public keys, it is somewhat clear that attacks where the public key is publicly available for more than a few hours can succeed as there is enough time for a quantum computer to run Shor's algorithm.

As such, we can identify the following scenarios where Bitcoin users reveal their public key.

1. Bitcoin UTXOs secured by P2PK challengeScripts display the public key in plaintext in the output of the transaction. As soon as a transaction with such an output is broadcast to the network, a quantum attacker can compute the corresponding private key and thereby consume the P2PK output just created.

2. Any Bitcoin UTXO secured by any other type of challengeScript will require the public key in the scriptSig that unlocks it. Thus, attackers can scan the blockchain for any other UTXOs that are locked by scripts with the same public key, and unlock them. In fact, a fast quantum attacker could even attack the very transaction that reveals the public key in its input, but this case is considered separately in Section 3.2.2. However, if we operate under the assumption that the quantum attacker needs at least a few hours to compute a private key from a revealed public key, then a solution to this problem is not reusing a public key. One can prevent against this scenario by using public keys only once. In fact, reusing public keys is not recommended, neither by Bitcoin developers nor the community, as numerous studies identifying privacy risks have been conducted [43, 58, 70, 17, 27].

3. A group of parties using an m-of-n multisig type UTXO requires, each party to give his public key in order to construct the challengeScript. Thus, all the

members of the group have access to all the public keys. In fact, this type of UTXO usually appears when different set of parties in the group are not trusted, so it would make sense for one of the parties to disclose the public keys of the other members to a quantum attacker.

4. Bitcoin users that own currencies on other Bitcoin forks (e.g. Bitcoin Cash [1] or Bitcoin Gold [2]) can use the same public key on all forks where they operate. As such, users could reveal a public key in the challengeScript of one of the outputs or in the scriptSig of any of the inputs of transactions on Bitcoin forks. As Bitcoin forks share the same transaction history prior to the fork point, such behavior may allow adversaries to gain control over the funds on all forks.

5. A Bitcoin user can reveal his public key as part of a signed message to ensure integrity, in forums, or in payment channels (e.g. Lightning Network [52]). This type of unveiling is not immediately threatening as attackers do not know if any public key they find online is also used to secure some bitcoins. However, it is not far-fetched to imagine adversaries that regularly scan the network for standard type challengeScripts built using the revealed public key. As they have precomputed the private key needed to unlock the output, they can simply consume the funds. Furthermore, attackers could monitor all broadcast transactions and wait until one of the public keys they found online appears in an input as part of the scriptSig. When this happens they can use the computed private key to overwrite the transaction by specifying a higher fee on their own transaction. Miners would not be able to tell which of the transactions is the original one so they would just include the one with the higher fee.

3.2.2 Live Transaction Hijacking

In this section we consider the special case in which a quantum capable attacker can perform live transaction hijacking. Thereby an attacker attempts to compute the private key corresponding to a public key revealed in the input of a transaction broadcast to the network but not yet included in a block. Consequently, just like in a double-spending attack [34, 56, 35, 61], the attacker creates a conflicting transaction[1] spending the same UTXOs, thus stealing the victim's funds. Note, however, that this form of transaction hijacking differs from the more conventional notion of double-spending as the attacker is the beneficiary rather than the original transaction initiator.

It is important to note that the attacker, must not only create, sign and broadcast the conflicting transaction, but also first run Shor's algorithm to derive the private key. As timing is essential for such attacks, the performance of quantum computers plays a central role for the success probability of this type of attack.

An extension to this attack could be to combine it with selfish mining strategies [25, 57, 48, 27]. Assuming the adversary is also a miner, he could employ his

[1] Possibly with a higher fee to incentivise inclusion in the blockchain over the victim's transaction

computational power to attempt to build up a secret chain and, when in the lead, selectively publish blocks to cause main chain reorganisations. In contrast to traditional selfish mining attacks, the feasibility of this combined attack is expected to improve significantly, since the adversary can now also perform transaction hijacking, thus drastically increasing his profits. As such, the prospectively-gained revenue consists of more than just block rewards and transaction fees, as all funds contained in (non-quantum-resistant) UTXOs spent in the overwritten transactions are also at the mercy of the attacker.

3.3 Estimated Losses

As described, it is clear that in a post-quantum world the current implementation of Bitcoin is rendered useless as transactions can be immediately hijacked. Hence, the community will most likely deploy a quantum resistant signature scheme at some point in the future. If this is done in time users will be able to move their funds from unsafe outputs secured by ECDSA to quantum resistant ones. However, if a quantum computer appears without notice and the community has not fully transitioned to the quantum resistant signatures, all funds secured by revealed public keys can be stolen using the attacks described above. In order to illustrate the full extent of the issue posed to the community, we estimate the potential losses in Bitcoin (BTC). This is of relevance as the attacker can subsequently invest the profits in further improving his quantum capabilities.

Currently[2], approximately 33% [3] of the total amount of BTC reside in unspent outputs secured by a revealed public keys[4]. At the time of writing, this amounts to approximately 50 billion USD[5]. If a quantum capable attacker would appear at the time of this analysis, it would be impossible to guarantee the safe retrieval of these funds even if our scheme is deployed because a quantum enabled attacker would be in possession of the same information as the original owner. The community can remedy the situation by moving their funds to outputs that do not have the associated public key revealed.

Furthermore, approximately 30% [6] of these 33% are actually in P2PK outputs, which cannot be recovered in any way once quantum attackers appear. The only way to recover these funds is for the original owners to transition them to P2PKH or P2SH outputs. But the fact is that most of these funds originate in very old blocks so it is very probable that the original owners lost the private keys that could have unlocked these funds. Thus, this is a guaranteed bounty for anyone who manages to scale quantum computers enough to break ECDSA in Bitcoin.

[2] at Bitcoin blockchain height: 514877

[3] 5,687,262 ₿ out of 16,937,813 ₿

[4] All the data was gathered using the blocksci [3] tool and the Amazon Machine Image they provide.

[5] Calculated using a BTC/USD exchange rate of $9.369 [4].

[6] 1,761,130 ₿

3.4 Hindering Transition to Quantum Resistance

Given these possible attacks, the Bitcoin community is considering different quantum resistant signature schemes that could be suitable for Bitcoin. Once one of them is deployed, users will be able to create challengeScripts using the new quantum resistant public key, hence securing their funds against quantum capable attackers. However, **if adversaries have the ability to perform live transaction hijacking, users will not be able to transition their funds to quantum resistant UTXOs as their transactions would be immediately hijacked**. This is the exact problem we are trying to solve in this paper.

Section 4: Transition to Quantum Resistance

In this chapter we offer a solution to the problem of transitioning to quantum resistance securely. More specifically, we describe a new scheme, called QRWit, that can be used to consume outputs secured by the elliptic curve digital signature algorithm by leveraging on the security of a quantum resistant cryptographic system[1]. Thus, our scheme can be used to transition to quantum resistant signature schemes by simply consuming the funds and sending them to a quantum resistant output. As the scheme we are about to describe relies on a quantum resistant cryptographic construct that can produce signatures for a given message, we are assuming that such a system will be deployed before or at the same time with our scheme. A detailed discussion on deployment of quantum resistant signatures and their timing is done in Section 5. For the remainder of this section we assume quantum resistant signatures are deployed in Bitcoin. This means users can generate pairs of private-public keys that cannot be broken by quantum computers. The protocol update we propose will make use of such a pair of keys.

After the transition protocol is given, we offer a deeper analysis of each step of the protocol and demonstrate the security guarantees we claim it provides. Bitcoin-specific implementation details, as well as discussions on parametrization and necessary data structures are considered separately in Section 5.

4.1 Protocol Overview

In order to consume funds protected by some ECDSA public key pk, we propose the creation of a new quantum resistant public key (pk_{QR}) which will act as surrogate for pk. Thus, whenever pk is used to verify the validity of a signature, the protocol update we introduce would also require a second signature verification with pk_{QR}. Even though the classical signature can be forged by quantum attackers, we still require it for backwards compatibility reasons. Un-upgraded users will trust the transaction is valid because they check the classical signature. On the other hand, upgraded clients will also check the quantum resistant signature which cannot be forged. Furthermore, users must first prove to the network that pk_{QR} is indeed their own creation and is meant as a surrogate for pk. To achieve this we can employ the same strategies used in hash-commitment schemes. Hence, before wanting to spend funds associated with pk, a user must first publish a hash commitment $H(pk||pk_{QR})$, i.e., the hash of his concatenated public keys, thus linking the two keys together. After including this commitment in the blockchain, the owner of pk can prove to the network he is the creator of pk_{QR} by revealing both pk and pk_{QR} and pointing

[1]Bitcoin community shall decide the specific one

to the commitment transaction. As pk is not known by anyone other than the owner, only he could have produced such a commitment so the network will be convinced of his claim and validate the transfer. Furthermore, to avoid the risk of an attacker rewriting the blockchain history to insert his own commitment to his own quantum resistant public key, users should wait a sufficiently long time t_{sec} between committing and revealing pk. Any time which is long enough to guarantee chain rewrites are impossible is fine, but we argue this delay should be of the order of months. We discuss this subject further in Section 4.1.2.

We give an overview of our protocol in the form of a diagram (Figure 4.1) and describe each of the three steps in more detail in the following paragraphs.

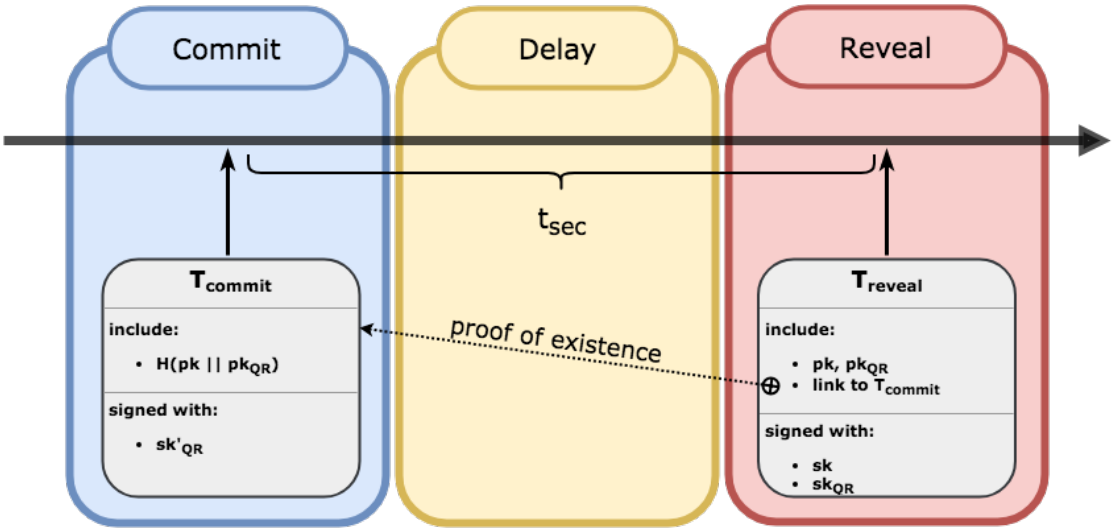


Figure 4.1: Overview of the commit-delay-reveal protocol for secure spending in the presence of a quantum attacker.

4.1.1 Commit

The first phase of the transition is to create a quantum resistant surrogate for pk. Such a public key can be created by simply using the quantum resistant cryptographic system that is now active in Bitcoin to generate a pair (sk_{QR}, pk_{QR}). To mark the commitment of the funds secured by pk, the user must include the hash of both public keys pk and pk_{QR} concatenated, i.e. $H(pk||pk_{QR})$, in some block of the blockchain in a transaction denoted T_{commit}. There are multiple variants for how to achieve this and the community or developers can decide upon the most suitable format and method of this commitment. For example, we could create a transaction T_{commit} which includes the hash commitment in an OP_RETURN type challengeScript in one of the outputs.

After T_{commit} is broadcasted to the network and included in the blockchain, the user will ask for a merkle branch proof of the existence of this transaction. This can be used at a later stage to prove to the network the existence of the commitment.

An important discussion should be made on what type of transaction is T_{commit}. As T_{commit} would immediately reveal the public key that verifies the signature on

itself, this poses an intrinsic problem in publishing it. Basically, an adversary capable of performing live transaction hijacking could attempt to prevent the creation of T_{commit} by double spending the transaction thus performing a denial of service. To overcome this problem there are multiple defence mechanisms we could employ depending on the timing of the transaction. For example, a guaranteed method of ensuring T_{commit} cannot be hijacked is to sign it with some quantum resistant key-pair (sk'_{QR}, pk'_{QR}). In fact, this can even be the same key-pair as (sk_{QR}, pk_{QR}). To achieve this, the user needs to consume some output that is already secured by a quantum resistant signature scheme. He could acquire these through trade or by successfully mining a new block.

Note that T_{commit} does not even have to be created by the owner of pk. In fact, one can envision a service that could offer commit transactions. Users would send the bytes representing the hash of their concatenated keys and the service providers would send back the proofs of existence which can be checked by the users immediately. After a delay of t_{sec}, the user can transition his funds as we will describe in the following paragraphs.

An important aspect of this phase is that it does not require any code changes in the Bitcoin code. The earlier a user publishes a commitment, the earlier he can reveal the keys and transition the funds. Users can start publishing such commitments even now, provided that they can guess the format on which the developers will decide upon for the commitment data.

4.1.2 Delay

This phase requires no user interaction and its only purpose is to allow the network to build Proof of Work on top of the block where T_{commit} was included. The user just needs to wait for a predefined security period t_{sec}.

Any funds associated with pk must be left untouched for this period of time as otherwise, pk would be revealed and quantum capable attackers would compromise the entire scheme. In fact, if one would try to consume funds associated with pk after our scheme is deployed, the transaction would not even be valid under the new protocol rules. For instance, clients that did not upgrade their code to our protocol update would still be constructing transactions without the quantum surrogates which would never be accepted. At the same time, they would be revealing the classical public key which would allow attackers to break the private key and subsequently use our scheme to steal the funds.

We argue that a long delay is necessary to ensure no blockchain reorganization could have occurred accidentally or have been caused intentionally by an adversary. While the specific choice of delay may be subject to follow-up scientific work and discussion in the community, we propose an initial period of 6 months.

The reasoning behind our choice is explained in the next paragraphs taken from the paper [62], which we have co-authored.

Necessity for a Long Delay Phase

The correct choice of the security period t_{sec}, used as protection against accidental and adversarial chain reorganisations, has a significant impact on the security properties of the proposed transition protocol. In contrast to previous proposals and discussions [6, 65, 66, 67], we emphasise the necessity of a sufficiently long delay phase, substantially longer than the standard confirmation period of ~ 6 blocks in Bitcoin. While the exact duration of t_{sec} may be subject to future discussion, we propose to require hash commitments to be older than *6 months*, i.e., the UTXOs used as input to T_{reveal} must remain unspent during this period.

As explained in Section 3 we assume that the feasibility of block reorganisation attacks, such as 51% attacks or selfish mining attacks requiring a smaller fraction of the overall computational power, is significantly increased for quantum-capable adversaries. In contrast to traditional reorganisation attacks, the prospective gains in this scenario are not only comprised of block rewards and transaction fees but also include any funds whose public keys have been revealed in one of the blocks overridden by the attacker. Hence, relying on a short security period of a few blocks (or no delay at all) provides insufficient protection against chain reorganisations in the presence of a quantum-capable attacker.

We note that in theory an adversary controlling a significant portion of the overall computational power could successfully rewind the chain further than t_{sec}, thereby altering the transaction history, and attempt to steal funds from all non-quantum-resistant outputs which were spent from during this period. However, we argue a fork overriding the block history of such substantial period as 6 months would be classified as a catastrophic failure of the system, forcing out-of-band measures to be undertaken by the majority of honest consensus participants. Specifically, we assume clients and miners will have incentive to manually reject the conflicting branch of the attacker[2].

However, by intuitive continuity arguments there must exist a point between short- and long-ranged attacks, where the community is unable to find even out-of-band consensus on how to proceed, i.e., whether to perform a manual invalidation (override) of attacker's fork or accept the conflicting branch of the adversary, as visualized in Figure 4.2. While under different circumstances, similar disputes have been observed in other cryptocurrencies and have led to permanent chain splits, as in the case of Ethereum [21] and Ethereum Classic [5]. Hence, a quantum-capable adversary may have incentive to attempt to exploit this "sweet-spot" to her advantage, as a destabilisation or split of the chain could yield a higher success probability of an attack.

By implementing a long delay phase, sufficient to trigger out-of-band actions in case a longer fork is created by an adversary, the probability of a malicious chain reorganisation interfering with the transition protocol can be minimized.

[2]Note that this does not require any changes to the reference client implementation, as Bitcoin's JSON-RPC API provides an `invalidateblock` call, which permanently marks a specific block as invalid, as if it had violated a consensus rule [19]. (There are of course many other clients available, some of which may require code changes to manually reject a branch.)

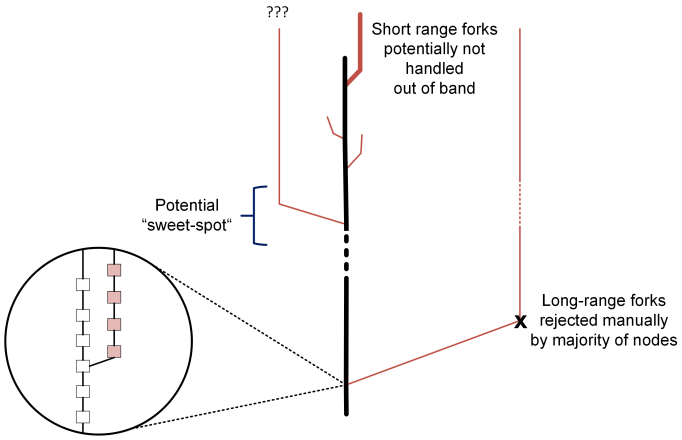


Figure 4.2: While long-range forks are expected to be manually rejected by the majority of nodes, this may not be possible with short-range chain-splits due to the limited time frame. There may exist a "sweet-spot" which causes a dispute whether to accept or reject the conflicting branch, destabilising or even permanently splitting the network to the benefit of the adversary (red).

4.1.3 Reveal

After the delay period has passed, the user can safely spend funds associated with pk by creating a transaction T_{reveal}, in which he does the following:

1. Reveal pk; this happens intrinsically when trying to consume an output locked by pk. To be more exact, this will appear in the scriptSig of the input of T_{reveal} that references the output to be consumed.

2. Give a classical signature that can be checked with pk; this also happens automatically when trying to consume the output. The signature is required as part of the aforementioned scriptSig. Note that this signature can be forged by any quantum attacker, but we require it for backwards compatibility reasons. The challengeScript was created using the old protocol rules, so the un-upgraded clients would consider the transaction invalid if the scriptSig does not provide the classical signature.

3. Give a quantum resistant signature that signs exactly the same data as the classical signature did; this can be included in a new segregated area that we call QRWit and that only upgraded clients can see. The purpose of this signature is to offer ownership guarantees in the presence of quantum attackers.

4. Reveal pk_{QR}; this will be needed in order to verify the quantum resistant signature and can be included in QRWit as well.

5. Give a proof that pk_{QR} is indeed the surrogate of pk as committed to in some transaction T_{commit}. This proof can also be included in the segregated area. The proof consists of a merkle branch proof that T_{commit} was included in some block and that it contains the hash of the concatenated public keys. This, also, can be included in QRWit.

As all the additional data will be transmitted through the segregated area of the transaction QRWit, un-upgraded consensus participants will simply believe T_{reveal} is a normal transaction consuming some UTXO secured by pk. They will be satisfied with just the classical signature and accept the transaction. However, the upgraded users will be able to see the additional data and to check the proof of existence for the creation of the surrogate quantum resistant key. Finally, they can now check that the quantum signature is valid and accept the transaction as valid.

The necessary implementation specifics are provided in Section 5. The protocol updates $P \rightarrow P'$ that we propose restricts the definition of a valid transaction. As such, the set of blocks that are valid under the new rules (P') is a proper subset of the blocks that would have been valid under the old rules (P). This means we can be deploy these changes as a soft fork. Once the majority of the network upgrades and accepts our code changes, the new protocol begins to be enforced.

We would like to accentuate that once the protocol is deployed, classic ECDSA signatures will no longer be accepted in Bitcoin without providing the extra data as well. Furthermore, if one uses an un-upgraded client and spends some funds, then the respective funds will not be transferred and will become prone to theft, as the public key is now revealed.

4.2 An Alternative Interpretation of QRWit

To prove the simplicity of the new protocol we would like to offer the following summary in the form of a definition, which also offers a different point of view on QRWit. What we would like to do is to strengthen the concept of a valid ECDSA signature, to require the extra checks we described.

Definition 38 (QRWit (Quantum Resistant Witness))
Denote an ECDSA signature created with secret key k for message m as $sig_k(m)$. Denote a quantum resistant signature created with secret key k for message m as $qrsig_k(m)$.

For any elliptic curve digital signature $d = sig_{sk}(m)$, where (sk, pk) is a private-public key pair, d is valid only if:

1. *a new valid quantum resistant signature, over m, is generated with sk_{QR} and verifiable with pk_{QR}, i.e. $qrsig_{sk_{QR}}(m)$ can be checked using pk_{QR}. In this scenario, validity refers to the rules of the quantum resistant signature verification algorithm.*
2. *a proof of common ownership of the two public keys is given, i.e. the user must provide a proof that (sk_{QR}, pk_{QR}) was created by the owner of (sk, pk).*

To prove that our protocol really implements this definition we would like to explicitly show how each requirement is satisfied by our protocol update.

1. During the reveal phase of our protocol we require a quantum resistant signature over exactly the same data that the classical signature was over.

2. The proof of existence of the hash commitment and the impossibility of altering this commitment, serves as proof that pk_{QR} was created by the owner of pk, which in turn implies the second requirement.

4.2.1 Flexibility

The advantage of looking at the protocol in this way is that the flexibility of the scheme is immediately clear. In this section we would like to illustrate this and show how users can benefit from it.

The inputs can reference any type of output. Apart from the standard output types that were described in Section 2.3, the scripting language of Bitcoin can be used to create many different challengeScripts. However, our scheme is not concerned with this in any way. We are just requiring that whenever an ECDSA signature is checked against a public key, a second quantum resistant signature over the exact same data is given. This means that more complex challengeScripts that include multiple signatures are automatically accommodated.

The outputs can be created in any way the user wants. There is no limitation imposed by our scheme on how to spend the coins. We only enforce rules on the consumption of UTXOs. A user could even send his funds to an UTXO secured by an old, non-quantum-resistant public key, if he wishes so. Furthermore, in comparison to other schemes we became aware of while working on this project, our protocol does not require the user to predefine the way in which he wants to spend the coins. The user can take this decision at the time of transitioning the funds.

4.3 Real Case Scenarios

In this section, we give a few real case scenarios that we believe are very probable to arise. In terms of when quantum capable attackers appear, we can rate the scenarios from optimistic (appears very late) to pessimistic (appears tomorrow). Hence, to prove our scheme is needed in any scenario, we will present the scenarios and the actions that should be taken from most optimistic scenario to most pessimistic.

Quantum Attackers Never Appear

Although this scenario is highly unlikely, we will go ahead and consider it. Even though quantum computers are not scalable enough to be used against Bitcoin, the community will still upgrade the current signature scheme (ECDSA) to a more evolved one (let us call it MISS: Much Improved Signature Scheme) at some time. Thus, when MISS will be deployed, the community will need to offer a way to transition from ECDSA to MISS under MISS security guarantees. If this would not be the case, then attackers that can break ECDSA would hijack the transition and

MISS would never be used. Our scheme can be used in this scenario by use of a MISS surrogate for the ECDSA public key.

Quantum Attackers Give Plenty Notice

This scenario assumes that the community is aware of the appearance of a quantum capable attacker with months in advance. In such a case, the community has plenty of time to deploy a quantum resistant scheme (let us call it QRS). QRS can be ran alongside ECDSA until quantum capable attackers are believed to start operating. In fact, the exact moment should take into consideration t_{sec} as quantum attackers could try to rewind the blockchain and replace older transactions. Anyway, while both signature schemes are active, users would be able to transition their funds from ECDSA secured UTXOs to QRS secured UTXOs just by normal transactions.

However, at some point, ECDSA must be considered broken and invalidated in Bitcoin. At that time, instead of simply invalidating all ECDSA signatures, the community should instead deploy our scheme. If ECDSA would just be invalidated, then users who did not transition and still have their funds in UTXOs will never be able to recover them. On the other hand, replacing ECDSA with our scheme, would allow users to transition to QRS even at a later time.

We note that ECDSA must not be invalidated before deploying our scheme as this would require a hard fork to recover the funds. We will prove this by contradiction. If ECDSA is invalidated before deployment of our scheme, there will be some new clients which upgrade to this protocol change which only accept QRS. Thus, the set of rules for valid blocks would state that all valid blocks contain only transactions signed with QRS. Now, if we further want to deploy our scheme, we would have to loosen the set of rules to re-allow ECDSA signatures which are required by the challengeScript in the outputs to be consumed. As such, this cannot be deployed as a soft fork as the set of blocks valid under the new rules is not a proper subset of the previous set of valid blocks.

Quantum Attackers Appear Tomorrow

This scenario analyzes the extreme case in which quantum computers appear without notice and our scheme has not been deployed yet. However, we assume quantum resistant signatures exist. This scenario could be identified by the Bitcoin community through reports made by users who are attacked. If this becomes real, all funds associated with revealed public keys are not recoverable and is only a matter of time until attackers consume them.

To recover the system from this extreme scenario, all transactions consuming ECDSA secured UTXOs should stop as live transaction hijacking is possible. The next step the community should take is to deploy a scheme such as ours as soon as possible. Commit transactions would need to be created using quantum resistant UTXOs, so we expect a surge in the price of quantum resistant bitcoins as most of the currency is still not quantum resistant.

4.4 Standard Reveal Transaction

To conclude the description of our new scheme (QRWit), we would like to describe how an average reveal transaction would look like. We shall try to predict the behaviour of an average user. The scenario in which he operates can be any of the above and we assume all his public keys have quantum resistant surrogates. In fact, he could even have the same quantum resistant surrogate for more of his public keys. However, this is not recommended as it would prove to the network that all those public keys belong to the same person, information that would aid adversaries in deanonymizing a user.

Furthermore, let us assume that the average user has multiple UTXOs where his funds reside. After he waits for the delay period to pass for each of the public keys he plans to use, he can construct reveal transactions. Once he uses one of the public keys in the reveal transaction, this key is now revealed and a quantum attacker can immediately use it to commit to his own quantum resistant surrogate. However, in order to consume the rest of the funds associated to it, the attacker would have to wait for the delay t_{sec} to pass. During this time, the legitimate owner of the funds should consume all of them by using his quantum resistant surrogate and creating reveal transactions. If the user consumes all the UTXOs in time, the quantum attacker cannot do anything to steal the funds as the outputs are already consumed.

As such, we would like to emphasize that there is not a one-to-one correspondence between commit transactions and reveal transactions. One commit transaction can be used as proof of existence for many different reveal transactions. Moreover, one reveal transaction can, and probably will, consume multiple UTXOs so multiple commit transactions can be linked in one commit transaction. However, for the whole reveal transaction to be valid, all commit transactions have to be older than the delay period.

Section 5: Implementation

In this section, we describe the implementation of our scheme in Bitcoin. We will show how we simulated the deployment of quantum resistant signatures and how each phase of the scheme impacts the code. Furthermore, we will explain how we adapted the code to allow this protocol update to be deployed through a soft-fork. We have built our code on top of the Bitcoin code base [18] as it was on the 18th of May 2018. More specifically, we have chosen the commit with hash `d792e47421fcb9ce3b381c1e6d8902777ae3f9f3`. Bitcoin does have stable releases every few months, but we decided to extend the code as it was at the aforementioned commit as some pieces of code were refactored in such a way that enabled an easier integration of the changes we were trying to implement.

5.1 Quantum Resistant Signatures

As mentioned in Section 2.5, we are assuming that quantum resistant signatures exist in Bitcoin. However, their implementation is a completely different research study and a convoluted topic in itself, hence we do not aim to offer a viable prototype for quantum resistant signatures in Bitcoin, but rather a hack that would allow us to build the rest of the code. As such, we need to make an assumption about how a new quantum resistant cryptographic system will be deployed in Bitcoin, so that we know how to simulate it. Thus, when the complete implementation will appear, our code can be easily integrated.

Currently, in Bitcoin, there is only one cryptographic system implemented (ECDSA) so the code is not flexible enough to allow an easy integration of a new cryptosystem. The current elliptic curve cryptographic primitives are modelled in the following way:

1. `CKey` is an object representing an ECDSA private key. Private keys can be created, compared, serialized (and un-serialized), validated (for integrity), and invalidated. Furthermore, the object offers functionality to create ECDSA signatures for given messages using the private key represented by this object.
2. `CPubKey` is an object representing an ECDSA public key. The object handles creation, comparison, serialization, verification, and invalidation of public keys. Moreover, it offers a method (`Verify`) through which users can check that a signature over some message was created using the private key associated to the public key represented by this object.

Quantum resistant signatures can thus be deployed in an elegant manner by applying the strategy pattern on the aforementioned objects. When instantiating a

`CKey` or a `CPubKey`, the strategy of choice can be specified in the constructor to obtain a pair of keys that belongs to a certain cryptosystem. However, this would require changes in all the seams where private or public keys are constructed. Furthermore, the serialization implementation would have to be modified to specify the strategy chosen.

In the future, the community will need to deploy quantum resistant signatures and offer a way to check if a certain public key is quantum resistant or not. As our protocol is not interested in how this will be achieved we are just assuming that the `CPubKey` object will offer a method that can be called to obtain the guarantees offered by the underlying public key (e.g. is it quantum resistant?). To avoid modifying the code base in many different places, we have designed a simple hack for providing the required functionality. We will simply consider that half of the ECDSA public keys are quantum resistant. To implement this neatly we provide the following implementation of the method $IsQR$ which classifies public keys as quantum resistant or not.

```
/** An encapsulated public key. */
class CPubKey
{
private:
    unsigned char vch[PUBLIC_KEY_SIZE];
    ...
public:
    ...
    // Check if this public key is quantum resistant.
    bool IsQR() const {
        return vch[4] < 0x80;
    }
}
```

Figure 5.1: Implementation of quantum resistant public keys, by considering approximately half of the ECDSA public keys are quantum resistant.

Note that the 4th byte was chosen randomly, while the hex value `0x80` is the median of all 8-bit numbers. As any byte in a public key is essentially randomly generated, the approach we implemented will yield quantum resistant public keys half the time. Note that with such an implementation any quantum resistant public key is also a valid ECDSA public key, which means that signature verification, generation and any other constructs remain completely unchanged. In fact, the only thing we care about is the ability to classify some keys as quantum resistant or not. This is enough to allow implementing the other bits of the protocol.

5.2 QRWit Implementation

To implement the protocol we have described, there are two stages we need to think about: commit and reveal. Apart, from these aspects, we also have to consider deployment of the code and backwards compatibility. In the following sections we will approach each of these aspects with code samples and explications.

One of the challenges of being a Bitcoin developer is the backwards compatibility thinking. Because the system supports a currency intended to be used by people throughout the world, any protocol update should be backwards compatible so that users who do not wish or are just unable to upgrade can do so without losing their funds. Sometimes users are off-line for a long time, but they should be able to recover their funds even after years of not using the system. Although sometimes this is not possible and a hard fork is required, the core developers try to implement all changes as soft forks.

We would first like to explain why we tried to alter existing code as little as possible and why all code changes introduced have to be inserted in just the right places in this very convoluted system. Bitcoin is an open source project where developers are volunteers with expertise and a very specific Bitcoin mentality. As ensuring security is one of the main goals of the system, all code changes have to go through a very lengthy process of peer review, implementation, peer verification and optimization. Before code is merged in, the core developers have to accept the changes. Considering all these factors, code proposals should be very specific and only implement the functionality required and no other refactoring should be done. This does not mean refactoring doesn't exist in Bitcoin, but that is a separate process from integrating a new feature.

5.2.1 Commit Stage

The first phase of our protocol is the hash commitment that will be used to prove the common ownership of and ECDSA public key and a new quantum resistant public key. As explained in Section 4.1.1, this phase requires no code changes and users can start creating hash commitments whenever they want, once quantum resistant signatures exist. Below we will describe the specific syntax we used and the additional tools we created in order to help users create such commitments much easier.

Structure of the Hash Commitment

The structure of the hash commitment we use in our implementation is simply the direct concatenation of the bytes of the old public key, followed by the bytes of the quantum resistant public key. This can be seen below

```
uint256 hashCommitment = Hash(
        oldPubKey.begin(), oldPubKey.end(),
        qrPubKey.begin(), qrPubKey.end());
```

Although there exist other formats which would provide more flexibility, we decided on this trivial approach for simplicity as we are building just a prototype to prove the concept.

In our implementation this hash commitment is included in a transaction by using an OP_RETURN type output. Transactions containing such outputs can be created using the functionality that already exists in the Bitcoin client.

Tools

To enable users to easily create such hash commitments and to generate the proof of existence for the commitment transaction, a few remote procedural call (RPC) commands can be used. Some of them are implemented by us to add functionality while others already existed in Bitcoin. As such, we will detail the functionality we added and explain how to use the existing RPC commands. Note that the RPC commands prefixed with `m` are implemented by us. The following list also serves as a guide for completing the Commit phase of our protocol.

1. `mkeys` is an RPC command that takes no arguments and lists all the public keys owned by the user. The output is a map from addresses [1] to key ids. For informing the user, we classify the ids of the keys by resistance against quantum computers. For example, after generating 5 new addresses and running the command we obtain:

```
$ bitcoin-client mkeys
{
  "2MtT7n9...": "qr-e8fbc9703cc25...",
  "2Mtx3HB...": "non-qr-8784e25c3c3a9...",
  "2Mu9qcT...": "qr-13816b76f42aa...",
  "2Mucsei...": "non-qr-90e1a91b6161d...",
  "2Mv1wqz...": "qr-57a3ea191b111...",
}
```

   Colours are added to aid in the visualization of the example we will use throughout the section.

2. `mcreatesurrogateforkey` is an RPC command that takes as input the id of an old ECDSA key and the id of a new quantum resistant key, and returns the hash of the concatenated public keys. This hash should be used as value for the hash commitment. For example, let us use the first key from the above output as surrogate for the second one to obtain:

```
$ bitcoin-client mcreatesurrogateforkey 8784e25c3c3a9...
e8fbc9703cc25...
"f32f6d55db7bcfcedf542a52b3..."
```

[1] Addresses are base 58 encoded challengeScripts. They are used in Bitcoin as destinations of funds. Many people consider that outputs must be directed to some address, but the fact is that only a subset of the possible challengeScripts are actually addresses. For example, P2PKH and P2SH are addresses, but OP_RETURN is not an address.

3. `createrawtransaction` is an RPC command used to create transactions. As arguments it takes a list of inputs and a list of outputs and as result it returns a blob of data which represented the serialized transaction. Each input is a reference to some UTXO and each output is a key value pair. If sending money to someone, the key should be the recipient's address and the value, the amount of money to send. If creating an OP_RETURN output, the key is `"data"` and the value is a hex value that represents the data that will be pushed in the challengeScript. As such, in order to create a hash commitment a user would run a command similar to this:

```
$ bitcoin-client createrawtransaction "[
␣␣␣␣{\"txid\":\"6efc668eaf5...\",␣\"vout\":0}
]" "[
␣␣␣␣{\"data\":\"f32f6d55db7bcfcedf542a52b3...\"},
␣␣␣␣{\"2N9kiH7gGV...\":11.999}
]"
0200000001ffd233eedb38974396b4dd26dc9982c0a6...
```

   The above transaction can be signed with the `signrawtransactionwithwallet` RPC command and broadcast with `sendrawtransaction`. The latter, returns a 64 byte string representing the txid of the commit transaction.

4. `gettxoutproof` is an RPC command that creates merkle branch proofs for given txids in a block. Users will use this to generate proofs of the fact that their commit transaction is indeed included in a block.

```
$ bitcoin-client gettxoutproof "[\"6efc668eaf5...\"]"
0000002029a40fbb30c52fb582e24884e1297e605dec...
```

   The output can be checked using the RPC command `verifytxoutproof` which takes as input merkle branch proofs and produces a list of txids and the block in which they all appear. In fact, in our case it would be only one txid.

Equipped with all this data, a user can move on to the next stage of the protocol.

Delay Stage

This doesn't require any actions from the users as they just need to wait for the predefined 6 months delay period to pass. We have decided on 6 months for considerations discussed in Section 4.1.2, but this could be determined to be different after more extensive research on the topic.

To ensure users do not reveal their public keys during this stage, we added code in the Bitcoin client that warns the user every time he is about to broadcast a transaction where a public key, with a surrogate commitment, would be revealed.

5.2.2 Reveal Stage

Having waited for the security period to pass, users can consume funds secured by ECDSA public keys with the guarantee that no attacker can hijack their transaction. In order to complete this section in code we had to think about adding the surrogate data (i.e. quantum resistant signature, quantum resistant public key, proof of existence) to a transaction, verifying the proof and validating the signature.

Structure of the Reveal Data - QRWit

Under the new protocol rules, all transactions that consume an ECDSA secured UTXO, will need to provide quantum resistant surrogate data. To this end, we create the concept of a quantum resistant witness (QRWit), i.e. data witnessing the fact that quantum resistant surrogates exist in this transaction.

For each input of a transaction, there will be an additional area of data at the end of the transaction. We will name this area "qrWit" and it will contain a vector of public key surrogates. We require one qrWit for each input instead of a global one, because each input could be signed with a different SIGHASH type so for ease of implementation and clarity we have taken this choice. All the qrWits in a transaction form the segregated are we call QRWit.

A public key surrogate is represented by the following object:

```
class CPubKeySurrogate
{
public:
    CPubKey pubKey; // old key
    CPubKey qrPubKey; // new key
    CTransactionRef commitTx; // commit transaction in full
    std::string proof; // a serialized merkleproof object
    std::vector<unsigned char> qrSig; // new signature
    ...
}
```

We have modified the code such that when users sign transactions under the new protocol rules, the Bitcoin client will automatically add surrogates for all the ECDSA public keys that appear in the input as part of the scriptSig. The surrogates are grouped by input and added in vectors that represent qrWits, one for each input.

Furthermore, we have created an additional RPC command, `mgetrawrevealtx` that creates a reveal transaction which moves all the funds associated to a public key to some new address. This does not mean users have to move all the funds tied to a public key at once, but as explained in Section 4.4 they will have to do it in a certain time frame. To simplify this process for them, we have created the aforementioned command. The command takes as inputs the id of the public key we are trying to transition to quantum resistance, the new address to which we wish to send the funds, a serialized transaction representing the commit transaction and a serialized merkle branch proof. The returned value is a serialized transaction

with all the surrogate data in place and all the UTXOs associated to the old public key consumed as inputs. The user can now sign and broadcast this transaction to transition his funds.

Verification of QRWit

Having described the creator's part of the protocol, we will now focus on the verifier's part. More specifically, we need to add code that enforces the new protocol rules at a consensus level. To this end, we will first give a short introduction on the stages a transaction has to go through in order to be accepted by a node. Afterwards, we will continue to demonstrate the changes we implemented to accommodate our protocol updates.

Currently, a transaction object is serialized and relayed to the network using the Bitcoin peer-to-peer network. Whenever e node receives a transaction, it will have to decide whether this transaction is valid or not. If it is valid the transaction will be added to the node's memory pool, where it will sit until it is included in a block. However, we do not want to perform unnecessary expensive checks if there is no need. Furthermore, we want to access the global state, which requires thread synchronization, as little as possible. Thus, the Bitcoin pipeline for validating transactions is the following:

1. Unserialize transaction blob of data to the C++ object `CTransaction`. This could fail if the transaction is malformed or the creator used a buggy client implementation.
2. Perform transaction checks that require no context. Some of this checks are not consensus critical, but enforce a good behavior and prevent denial of service attacks through CPU or memory exhaustion.
   (a) Verify the size respects the imposed limits.
   (b) Verify there is at most one OP_RETURN output.
   (c) Verify the transaction is not a coinbase transaction as those are only accepted in blocks not as loose transactions relayed by the network
3. Acquire a lock on the memory pool and UTXO set and cached the outputs referenced as inputs by the transaction. After the caching is done, the lock is released and the data cached is used to check for conflicts with other transactions in the memory pool. If the inputs of two transactions overlap then the transactions conflict and it is guaranteed that one of them will not be included in a block. The miner will choose to keep the one with a higher fee as that will benefit him most.
4. Acquire a lock on the blockchain and retrieve the current consensus rules represented through flags in the latest block. Use these flags to perform specific checks that might require the blockchain and pass them to the signature validating objects to enforce specific protocol updates.

5. Fully validate the transaction by checking the scripts in the inputs and the signatures. This is the most expensive part of the validation.

If all the above checks pass, the transaction is considered valid and is included in the memory pool. Every ten minutes it will have the chance to be picked up and included in a block.

Our code changes are listed below.

1. We modified the transaction serialization format and the respective code to include one qrWit per input. This is serialized in vector format, i.e. the number of elements in the vector first and then the elements one after the other.
2. We did not modify the context independent checks.
3. We did not modify this stage at all, as well.
4. The protocol update is deployed through version bits which will be described in Section 5.2.4. When the flag marking activation of our scheme is present we have to validate the surrogate data. Furthermore, we will instruct the signature validators to check quantum resistant signatures by adding a script check flag.
5. When fully validating the transaction by checking the signature we simply modify the definition of a valid signature as described in Section 4.2.

We will now describe in more detail steps 4 and 5 of the above changes.

Validate Surrogate Data

Once our protocol update is activated, all transactions that have surrogate data have to be validated. As such, we iterate over all qrWits and over all their elements (`CPubKeySurrogate` objects) and call the

```
bool CheckSurrogate(CPubKeySurrogate surrogate)
```

static method. The boolean returned specifies if the given `CPubKeySurrogate` is valid or not. The following checks, essentially, ensure that the two public keys specified in the surrogate are indeed created by their owner.

1. Verify that `qrPubKey` is indeed quantum resistant, by calling `IsQR()`.
2. Verify that `commitTx` contains the hash of the concatenated `pubKey, qrPubKey`.
3. Verify that the merkle branch proof for the existence of `commitTx` corectly matches this transaction. To achieve this, we unserialize `proof` and obtain a `CMerkleBlock` object as used in `gettxoutproof`. This object specifies the block on which the merkle branch proof was constructed and the transactions matched. We check if the txid of `commitTx` is matched and if it isn't we declare the surrogate as invalid and the whole transaction is rejected.

4. Verify that the block in which `commitTx` appears, as returned by the unserialized `CMerkleBlock` object, is indeed part of the main chain and is older than t_{sec}. In order to perform this check we need to obtain the blockchain lock once again.

Verify Quantum Resistant Signature

Once the check described above passes and we reach the final stage of validating a transaction, the scriptSigs and the challengeScripts from each input will be concatenated and ran as a script. Inside this script, various operations will take place. We are not interested in any of these operations and we do not add or change these in any way. However, some of them will be signature verifications, i.e. a public key and a signature will be provided and a `TransactionSignatureChecker` will be used to check the validity of the signature. As we mentioned multiple times, this is what we are interested in modifying. We will change the implementation of the `VerifySignature` method as follows:

```
bool TransactionSignatureChecker::VerifySignature(
    const std::vector<unsigned char>& vchSig,
    const CPubKey& pubkey,
    const uint256& sighash) const
{
  if (!pubkey.Verify(sighash, vchSig)) return false;
  if (enforceQR && !pubkey.IsQR()) {
    for (const auto& s : txTo->vin[nIn].qrWit)
      if (pubkey == s.pubKey)
        return s.qrPubKey.Verify(sighash, s.qrSig);
      return false;
  }
  return true;
}
```

Remark 17 (Valid Surrogate)
Note that the `qrPubKey` is guaranteed to be in control of the same user who controls pubkey as we have already validated all the `CPubKeySurrogate` objects.

As we have added an additional area of data to the transaction, i.e. the qrWits, we need to include a fingerprint of this data in the block that will contain this transaction. Otherwise, adversaries could subsequently change this data when they relay the block and other nodes would consider the transaction invalid. To this end, we build a merkle tree of hashes of all the qrWits in the transactions of a block and include it in a special script of the coinbase transaction. The hash of the coinbase will be part of the merkle tree of all the transactions of the block, thus ensuring that no one can alter any surrogate data of any transaction in a block. We could have achieved the same goal by including the hash of the qrWits of a transaction in the transaction itself, but for backwards compatibility reasons this is not possible as we will explain.

5.2.3 Backwards Compatibility

As this protocol update is intended for soft fork deployment, we need to ensure that old clients can also validate transactions with qrWit. However, if they would receive such a transaction they would not be able to understand the format of the new data. Hence, when a new node communicates with an old one, all the transactions and blocks relayed will be stripped of the qrWit data. Thus, old nodes will consider reveal transactions as normal transactions with just an ECDSA signature which they will accept as they think such signatures are still valid.

For this reason, the merkle tree root in the block header has to be constructed from stripped transactions so that old miners can validate them. At the same time, new nodes know that the transaction has some extra data, the fingerprint of which is included in the special script of the coinbase transaction. They will check that the fingerprint matches the data and consequently validate the transaction.

Furthermore, in case a new node mistakenly sends a complete transaction with the qrWit included, the old nodes should reject the transaction as soon as possible so that they do not waste resources on transactions that they cannot even unserialize. Hence we change the serialization format of a transaction as shown in the digram below.

Original Serialization Format

```
int nVersion
vector<CTxIn> vin
vector<CTxOut> vout




uint nLockTime
```

⇒

Updated Serialization Format

```
int nVersion
unsigned char dummy = 0x00
unsigned char flags (!= 0)
vector<CTxIn> vin
vector<CTxOut> vout
if (flags & 2):
  vector qrWits
uint nLockTime
```

With this change, it is easy to see why old clients will immediately invalidate such transactions. As vectors are serialized with their length first, old clients will treat the `dummy` value as the length of the vector of inputs and any transaction with no inputs is immediately rejected.

The `flags` variable is included there to mark the type of extra data we added. In fact, this approach is already employed in Bitcoin for the implementation of segregated witness (SegWit)[40], which uses the first bit of `flags` to mark extra data serialized by them. Similarly, we reserve the second bit to mark our extra data.

5.2.4 Version Bits

To conclude the implementation section we would like to explain how deployments are achieved in Bitcoin through the use of Version Bits [51]. This concept allows for multiple soft fork deployments at the same time. The idea is to allow users to upgrade their code whenever they want, but to only start enforcing the new rules

when the deployment is supported by a majority of users.

Each block includes in its header a version number. Initially this was used as an unsigned integer that was incremented whenever upgrades were deployed. Miners would read the version of the last block and know to enforce the specific rules specified by the version. The drawback of this approach is that it does not allow for multiple deployments in parallel. Hence, the concept of Version Bits was introduced.

The version of a block is now treated as an area where miners can signal adoption of a feature. Each feature reserves a bit in this area, for example, we chose bit 2 for qrWit. Furthermore, the bit will have a specified start time and timeout after which the feature is considered to have failed deployment because not enough participants upgraded to it. Miners who do upgrade their code, will signal this by setting the specific bit in the block header. When enough blocks in the blockchain have a certain bit set, the corresponding feature is considered active and the rules enforced by it start to manifest in the code. To describe the process in more detail we give the following state automata which describes the state of a feature depending on its version bit.

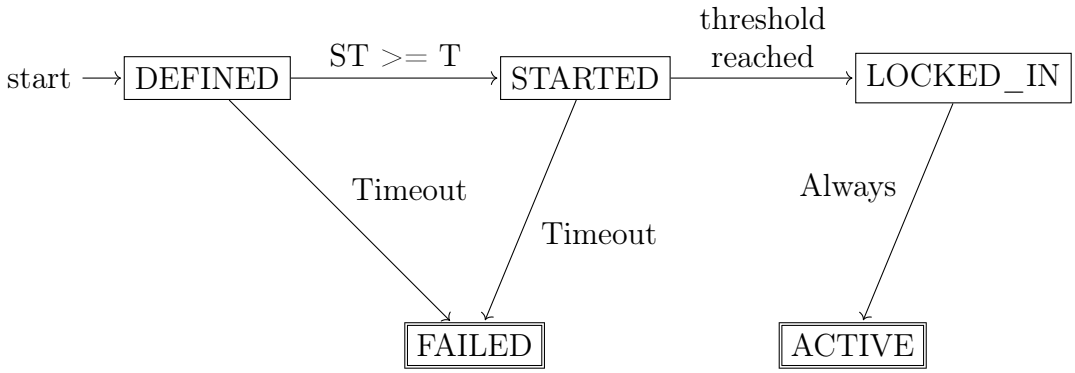


Figure 5.2: Finite state automata showing the different stages of a feature deployment using the Version Bits protocol. T denotes the current time and ST the specified start time of the feature. Threshold reached refers to the fact that a certain percentage of the blocks in the last 2016 blocks signal the version bit of this feature.

Therefore, to allow our protocol to be deployed in a manner that does not impact other features and to allow the miners to upgrade their code at their own pace, we implemented our code using the Version Bits technique. Hence, all consensus critical changes we implemented such as surrogate validation and quantum resistant signature verification only activate when the feature is in active state.

Therefore, to summarize the whole scheme, QRWit is a segregated transaction area that is checked for the quantum resistant signature over the transaction, and for the existence proof of the commit transaction.

Section 6: Related Work

Before we evaluate the overall success probability of QRWit, we would like to construct a more complete picture of the existing proposals and alternative schemes that we could find that aim to solve the same issue as our proposal, QRWit. With the exception of Fawkescoin, which is a proposed cryptocurrency, none of the works we found have formal descriptions. Before we go into these schemes, we would like to present some general research done towards integrating Bitcoin in the post-quantum era, even though it does not directly relate to our specific work.

As the possibility of quantum computers emerging in the near future is increasingly appreciated by members of the Bitcoin community, a number of approaches to make Bitcoin resilient against quantum-capable adversaries have recently appeared online.

As we outline in Sections 3 and 4, a first step towards maintaining Bitcoin's security properties in a post-quantum world is replacing ECDSA with a signature scheme that is at least believed to be quantum resistant. As presented in Section 2.5, such schemes exist and are meant to be implemented on classical computers [28, 23, 64]. On the other hand, other proposals aim to provide quantum resistant signatures by means of quantum hardware [32, 36, 31]. An alternative research direction focuses on identifying alternatives to Proof-of-Work, as a countermeasure to possible unfair advantages in mining through Grover's algorithm [12, 33].

However, we will not focus on these works as they do not consider the issue our paper is most interested in, i.e., transitioning Bitcoin in a post-quantum world in the presence of an already fast quantum-capable attacker. Instead, we would like to briefly analyse the following ideas and works. As very little data was available around these proposals, we have contacted their authors and requested more explanations. In some cases we obtained further details which we include here, while in others, we learnt that the authors conceded their ideas, hence we will not mention them any more. Furthermore, we would like to note that our scheme was developed completely independently from all the ideas we are about to mention.

6.1 Johnson Lau's Two-Stage Commitment

This method is presented on Twitter in a comment from Adam Back [6] and in further explanations he offers [7, 8, 9]. From our understanding, the scheme follows the structure of a two stage hash commitment. The first step is to commit to the ECDSA public key and a quantum resistant public key in the form of a hashed pair. This commitment would be stored in the OP_RETURN field of a transaction, similarly to our scheme. The second step is to reveal the pair in plain-text in another transaction that is signed against the quantum resistant public key. The

commitment ensures that the owner of the ECDSA public key, also controls the quantum resistant public key, while the quantum resistant signature guarantees QCs cannot modify the transaction.

This scheme is similar to out proposal, with the exception of the delay phase. The lack of a sufficiently large delay period means quantum attackers have the opportunity to rewrite the blockchain and insert their own commitment, thus being able to steal the funds. It appears that Adam Back is aware [10] of this, but assumes [11] Grover's algorithm does not offer a considerable advantage in mining, thus making massive chain rewrites impossible.

6.2 Tim Ruffing's Committed Transaction

This scheme is clearly described on the Bitcoin-dev mailing list [65] and in the discussions that led to it [66, 67]. The initial proposal suffered from multiple deficiencies, as mentioned by the author, but the current version of the scheme seems functional, although very rigid (i.e. the user has to create a transaction in advance of actually broadcasting it). The general idea of the scheme is to secure a non-quantum resistant transaction by first committing to it in a different transaction. The commitment is enacted by symmetrically encrypting the unsafe transaction. To guarantee that only someone who can consume the UTXOs being transitioned can create such a commitment, the encryption key is the scriptSig itself.

Apart from this, the scheme also uses a tagging technique on the commitment which eliminates the need for a link between the commit and the reveal transactions. Similar to this, we found an improvement to our scheme that will allow users to self-configure the delay period. We describe this in Section 8.

The question of long delay between commitment and reveal is not approached and we would like to note that a delay actually affects the usability of this scheme a lot. Although it might be feasible to predict spending in case of a delay of a few blocks, it is certainly not feasible if the delay needs to be much larger. However, the advantage of this scheme, as the author points out, is that the reveal transaction can be immediately considered confirmed as its encrypted form was already committed in the blockchain, so the community can trust this transaction.

6.3 Fawkescoin

This is a cryptocurrency relying only on secure hash functions, thus avoiding completely the need for public-key cryptography [20]. In this digital currency system, each user holds a secret value which is used to provide ownership guarantees. Transactions are implemented as a commit–reveal scheme, thus involving two interactions with the blockchain for a single transfer. In this work, we present the basic mechanism behind transactions. For a more detailed explanation, refer to the Fawkescoin paper [20].

Assume that user O_Y, who holds secret Y, wants to receive some money from user O_X, who is in possession of secret X. To this end, the following steps are taken, where H is a cryptographic hash function.

1. O_Y sends $H(Y)$ to O_X.
2. O_X proceeds to include $H(X, H(Y))$ in the blockchain, thus committing to sending funds to whoever can show Y.
3. A few blocks are added to the blockchain on top of the block including the commitment. This is necessary to ensure immutability.
4. O_X reveals $(X, H(Y))$. This act proves that O_X owns any outputs to $H(X)$.
5. O_Y can now spend his funds in a similar manner.

While not aiming at transitioning to a quantum resistant signature scheme, this cryptocurrency shares some conceptual similarities with our proposal. However, it is more restrictive as it requires users to commit to the destination of funds in advance and it does not take into account the attacker's ability to rewind the blockchain.

Section 7: Evaluation

Having presented the theory behind QRWit, its practical implementation, and some alternative schemes which achieve similar effects, we can evaluate our proposal both theoretically and from the implementation perspective.

Furthermore, note that we have already provided a comparison with other schemes that have the same goal as ours, in Section 6. Thus, the advantages of QRWit over the other proposals are clear, i.e. it works even if quantum attackers can rewrite the blockchain for limited periods of time, and is much more flexible in regards to when and what types of UTXOs can be consumed.

Before we go into the actual analysis, we would like to clearly mention one of QRWit's drawbacks: the long security delay. Although, this delay is necessary in the current implementation as we described in Section 4.1.2, there is another scheme that we describe in Section 8 which solves this problem, letting the users to decide for themselves how long to wait before commit and reveal.

7.1 Theoretical Analysis

Before ensuring that our code changes correctly implement the protocol rules enforced by QRWit, we need to show that the new scheme really is secure. To this end, we will informally describe the requirements of a secure scheme in this scenario, and show how the assumed capabilities of a quantum attacker do not offer any means of hijacking a transaction. Through these, we can construct an adversarial model of provable security, against which we can evaluate QRWit and determine if it meets the requirements or not.

Furthermore, note that for our purposes, security only needs to refer to the safe spending of funds under the presence of QC. We do not need to make any additional claims about the type of outputs, as a quantum resistant output is just a special case of a standard one (i.e. it requires a quantum resistant public key in the challengeScript). In fact, for most outputs, it would be impossible to decide if they are secured by ECDSA or a quantum resistant scheme, until they are consumed as part of other transactions.

Without making any claims about the current security Bitcoin offers in the absence of quantum capable attackers, we can define the new notion of security (i.e. in the presence of such attackers) by building on the former.

Definition 39 (Secure Spending of UTXOs)
A scheme for consuming (spending) UTXOs*, in the presence of a quantum capable attacker (QCA), is* ***secure*** *if and only if, the QCA is not more powerful than a classical attacker, i.e. he cannot achieve anything more than what a classical attacker can.*

To formally describe the capabilities of a quantum attacker, we can define them in terms of a classical attacker, as well.

Definition 40 (Quantum Attacker Capabilities)
A quantum capable attacker can perform the following actions:

1. *Any actions that a classical attacker can perform.*
2. *Given an ECDSA public key (*pk*), can immediately compute the secret key that was used to generate* pk*.*

Therefore, we have to prove that Definition 39 holds under the new protocol rules introduced by QRWit, as described in Section 4. First of all, any transactions that consume quantum resistant UTXOs, are impossible to hijack by the QCA, as he cannot forge quantum resistant signatures. Secondly, the commit and delay phases of the protocol do not introduce any vulnerabilities, as there are no public keys revealed during these stages, so a QCA cannot use his extra ability to gain an advantage over a classical attacker. Thirdly, we have to analyse the main attack point, T_{reveal}.

During the reveal phase the ECDSA public key is revealed as part of T_{reveal}. However, for such a transaction to be valid, all the surrogate data must be valid. This implies that any ECDSA public keys revealed must have an associated commitment ($H(pk, pk_{QR})$) older than the security period t_{sec}, and that any data signed with ECDSA is also signed with a quantum resistant scheme. Thus, the QCA can construct a valid transaction, that consumes the same input as the original one, only if he manages to construct a quantum resistant signature over the data. As he cannot break the quantum resistant signature scheme, the only option is to use a different quantum resistant private-public key pair, (sk^a_{QR}, pk^a_{QR}), that is under his control. Furthermore, he also needs to have committed $H(pk, pk^a_{QR})$ in some transaction older than t_{sec}. It is impossible, even for a QCA, to construct such a commitment as he did not have knowledge of pk at the time at which he should have introduced the commitment in the blockchain. Thus, we conclude that a quantum attacker cannot modify T_{reveal} in any way.

Therefore, a quantum capable attacker does not benefit from his extra ability in any way, hence he is equivalent to a classical attacker, as required by Definition 39.

7.2 Implementation Analysis

For testing the implementation of our scheme, we have used both unit testing and manual testing on a simulated blockchain over which we have full control. The former tests target only the validity of the consensus rules and they act as a specification of the protocol, while the manual testing allowed us to test real case scenarios and also to create demonstrations that we can showcase when presenting the protocol.

7.2.1 Unit Testing

To ensure that our core and most relevant functionality runs as expected, we decided to write some unit tests that check the new transaction validation rules. We used these tests as specification for the overall protocol. Naturally, these tests only regard the reveal phase and the segregated data introduced in there. Using the current Bitcoin code architecture, one can write unit tests that create transactions and then send them through the standard verification pipeline, simulating the exact behaviour of a network node who includes the transaction in the memory pool. Note that these tests do not require a network or a blockchain to run as they will directly call the functions which handle validation of transactions and blocks.

To check the validity of individual transactions , we need to test the function `AcceptToMemoryPoolWorker`, to ensure that when our protocol is deployed and active, no invalid transactions are accepted. A test would simply be comprised of a call to the aforementioned function with a custom transaction as parameter and with the new consensus rules activated. Thus the types of transactions that we tested and the errors we expected from the test are presented below, grouped by the mechanism that should trigger the failure.

1. Transactions rejected when checking the surrogate data using the `CheckSurrogate` function. This is the first check enforced by the new rules. Such transactions can fail because one of the surrogates (qrWit) is invalid in the following ways:
   (a) The surrogate public key is not quantum resistant. We tested this by simply providing an ECDSA public key.
   (b) The T_{commit} referenced does not contain the hash of the pair of the provided ECDSA public key and quantum resistant public key. We tested this by including some random data in T_{commit}, instead of the actual hash.
   (c) The proof of existence (i.e. a Merkle branch) given does not prove the existence of T_{commit}. This could happen for multiple reasons:
      i. The Merkle branch does not generate the expected Merkle tree root.
      ii. The Merkle branch references more than one transaction.
      iii. The Merkle branch references one transaction, but it is not the hash of T_{commit}.
   (d) T_{commit} is not on the main chain. We tested this by creating an orphan block that is not included in the main chain.
   (e) T_{commit} is on the main chain but it is not old enough. We tested this by leaving a delay shorter than t_{sec} between commit and reveal.
2. Transactions rejected when checking a signature. These tests are only interested in rejecting those transactions which do not provide valid signatures under the new rules. Transactions that fall under this type of failure are:
   (a) Transactions which present invalid ECDSA signatures. These would immediately fail even by the old rules.

(b) Transactions which present a valid ECDSA signature, but no corresponding quantum resistant signature. These transactions would be considered valid by the first checks which validate the surrogate data as no surrogate data is present for the respective ECDSA public key. However, when checking the signatures in the scriptSig, the ECDSA signatures will require both a valid ECDSA signature and a valid quantum resistant signature, which would not be found as the ECDSA public key in question has no surrogate.

(c) Transactions with no segregated data. In fact, these transactions are just a special case of the previous one, as no segregated data implies no quantum resistant signatures are provided.

(d) Transactions which present a valid ECDSA signature, but an invalid quantum resistant signature. As in the previous two cases, these transactions would pass the `CheckSurrogate` verification, but would be considered invalid when checking the quantum resistant signature. We invalidated a quantum resistant signature by changing one of its bits.

To check the validity of a block of transactions, we need to test the function `AcceptBlock` to ensure that we only consider valid blocks when constructing our local copy of the blockchain. The changes in code that are tested here are those regarding the fingerprint of the segregated data QRWit. Any blocks which do not include the Merkle tree root of all the QRWits in transactions of this block should be invalidated. To test for this scenario we generated a block and then modified it in the following ways:

1. Remove the slice of data that represented the Merkle tree root. We achieved this by computing the root in the test, searching for the binary data in the serialised version of the block, and removing it.

2. Alter the slice of data that represented the Merkle tree root. To test this, we had a similar approach as to the above case, but instead of removing the data we simply flipped a random bit.

3. Alter one of the QRWits in a transaction. As in the previous two cases, to test this block is invalid, we searched for some known QRWit data in the transactions of the block and flipped one of the bits.

7.2.2 Manual Testing

In order to manually perform transactions by sending RPC commands to the modified Bitcoin client, we need a network on which to broadcast transactions and blocks. These blocks would then be validated by the other participants of the network. However, we do not want to operate at real costs, consuming real Bitcoins as this would be both very slow (a block appears on average every 10 minutes) and not very useful for our scenario. Furthermore, the real network would not even take into consideration the new consensus rules our protocol proposes. To solve these issues, the

Bitcoin code base provides two main testing frameworks or modes of operation: the testnet and the regtest.

Testnet

The testnet is an alternative blockchain with the same general rules as the actual Bitcoin network. Coins on the testnet are completely separate, cannot be traded for real bitcoins, and are supposed to have no value. The purpose of the testnet is to allow application developers or bitcoin testers to experiment with their implementations, without actually paying the costs of transactions or having to worry that their version of the code is broken.

For the changes we are trying to test, this framework is not useful because we would need a majority of miners on the testnet to run our version of the code. If we do not have a majority, then our rules would just never take effect. First of all, the deployment through Version Bits would never activate. Secondly, if we would manually activate the deployment, our client would basically drop all transactions as it considers them invalid by the new rules. However, the rest of the network would accept them so the blockchain would extend with blocks that we consider invalid. Transactions created by our client with the segregated data in place, would just look like normal transactions to the other miners as they would not even see the segregated data.

One way to overcome this problem would be to spawn enough miners to exceed the average hash power, so that we can construct a majority of miners. However, this would require too much work and it would just not be efficient from a tester's point of view, as any small change would require each of the miners forming the majority to upgrade.

Regtest

The regtest is one of the modes of operation of the Bitcoin client. It will instruct the client not to connect to nodes on the real Bitcoin network, but to nodes specified by the developer. As we do not require interaction with random nodes, this mode is perfectly suitable to test our deployment. Furthermore, regtest lets developers instantly create brand-new blocks that can be filled with no transactions. We will use this feature to simulate the passing of time which is required for the delay phase.

Therefore, the tests we are interested in performing using regtest are those that require node communication. From unit tests, we know that an individual node will correctly validate transactions and blocks, but using manual tests through the regtest mode, we can also test the communication between two nodes. As such, we can check if blocks and transactions are correctly relayed by the upgraded nodes.

All upgraded nodes should see all transactions and blocks with the QRWit present, hence being able to check the validity of that data as enforced by the new rules. However, old nodes which did not upgrade should only see stripped transactions which contain no QRWit data. The block will still contain the Merkle

tree root of all QRWits in the special location of the coinbase transaction, but they will not try to validate it as they do not even understand what that data represents. Thus, we have devised two manual tests that rely on regtest.

1. A new node communicating with an old node. In this case, we just aim to test that any standard transactions are accepted by the old node. Furthermore, to ensure that there is no mistake and the old node does not even see the QRWit data, we send manually form blocks with invalid surrogate data and check that the old node does not invalidate the block. In fact, he does not even receive the invalid data.

2. A new node communicating with another new node. On the other hand, in this scenario, the new node should be able to fully validate the data.

To conclude this section, we would like to note that the theoretical aspects of the work described here have already been subject to peer review, as our paper [62] was accepted for publication in a Blockchain Technology issue of the Royal Society Open Science journal.

Section 8: Future Work

In this section, we present further improvements to QRWit that would enhance usability thus increasing the chances of this scheme to be accepted by the Bitcoin community. For each of the changes we mention in this section, we will also clearly specify the issue they are addressing and the goals they achieve. We aim to describe the enhancements both from a theoretical point of view and implementation wise.

As our current implementation of QRWit is only a proof of concept prototype, we did not implement any of these changes, but we note that most of them could be implemented with little code changes as the basic transaction validation mechanism remains unchanged.

8.1 Commit Multiple Keys

The improvements we address in this section aim to decrease the overall cost of transitioning to quantum resistance. With the current scheme, users need to create one commitment for each of their ECDSA public keys. However, we estimate that T_{commit} will be quite an expensive transaction, as it needs to be funded by UTXOs secured with the new quantum resistant signature scheme and, under some scenarios, such outputs might be very scarce, so their value would rise in comparison to the non-quantum-resistant UTXOs. To solve this issue, we suggest to aggregate multiple commitments of different ECDSA public keys under the same T_{commit}.

However, one has to be very careful about how the aggregation is achieved in a way that does not deanonymise the user, i.e. the scheme must not reveal any new information about the user. In particular, it should not be possible to link multiple public keys to the same owner, as this would allow attackers to gain information that could lead to finding the owner's real identity. In fact, even with the current implementation, users should not commit different ECDSA keys to the same quantum resistant key, as this would prove to the world that the ECDSA keys have the same owner.

A naïve implementation, could try to optimise space by committing a list of multiple (or all the) ECDSA public keys of a user to the same quantum resistant public key. Although this would reduce space in both T_{commit} and T_{reveal}, it would also act as evidence for everyone else that the keys belong to the same person.

8.1.1 Merkle Pair-Tree

To achieve the aforementioned specification, we propose to change the data that is committed in T_{commit}. Instead of the current pair of two public keys (pk, pk_{QR}), the

new data will be a Merkle Tree root constructed from multiple pair commitments. More exactly, each leaf of the Merkle tree will contain the data that was previously contained in T_{commit}, as in Figure 8.1. Consequently, the root of the Merkle tree will be included in the OP_RETURN output of T_{commit}.

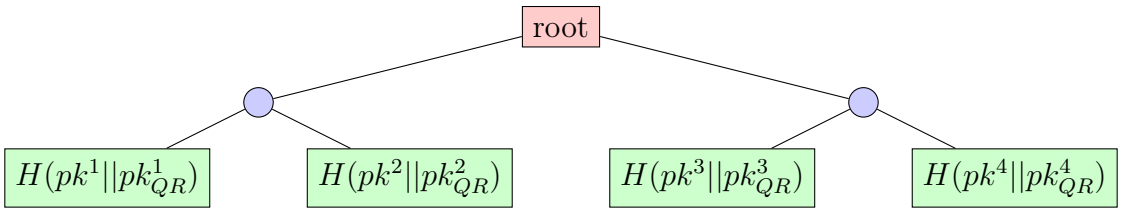


Figure 8.1: Merkle tree of hash commitments. The green nodes are the leaves and they each contain a hashed pair of keys. The blue nodes are intermediary nodes composed by hashing the two children. The red node is the data that will be included in T_{commit}. pk^i and pk^i_{QR} refer to keys of the same user i.

From the implementation perspective, the consensus rules validating the existence proof need to change to account for the fact that the hashed pair of keys is now deeply buried in the root of the Merkle tree. To this end, the new segregated area will also need to provide a Merkle branch that proves that the hash of the concatenated revealed keys (pk, pk_{QR}), is present inside the data committed in T_{commit}. The new Merkle branch required can be generated by the creator of the Merkle tree, whoever he is. For space issues in T_{reveal}, it might make sense to limit the size of the Merkle tree (and thus the size of the branch) to prevent memory exhaustion attacks.

Note that, this change does not weaken the security guarantees of the scheme in any way, because faking a commitment in the Merkle tree requires breaking the pre-image resistance of the cryptographic hash function.

Having described the improvement, we would like to point out that simply linking to the same T_{commit} from multiple ECDSA public keys, is not enough reason for an attacker to believe that the keys have the same owner, as there could be, and probably will be, service providers that post commitments on behalf of users. Actually, this is a very probable scenario as users who do not own quantum resistant UTXOs still need a way to commit.

Furthermore, using a service provider is economically better for all parties. The providers will be able to combine different hash-pairs (from many users) into the same T_{commit}, which means that their cost is fixed regardless of how many key pairs are committed. To prove to the client that the data given is indeed persisted in the blockchain, they will send him a Merkle branch of the data in the tree, T_{commit}, and the Merkle branch of T_{commit} in the block. Thus, users can commit all of their ECDSA public keys in one T_{commit}, which they will reference every time they reveal any of the keys.

8.2 Flexible Commitment Structure

As we mentioned in Section 4.1.1, users can start committing their ECDSA public keys to quantum resistant ones even now. However, in order to do this they would need to know what quantum resistant signature scheme will be used, and the specific format imposed by the validation format. It could be that the validation rules will consider the commit data is included in the first OP_RETURN output (like our current implementation), or maybe the rules will look for the commit data in some other area of T_{commit} that the community believes is better suited.

8.2.1 User-Configurable Commitment Location

For the problem of where to place the commitment data (whatever that is: hash-pair or Merkle root), we suggest some validation rules which do not decrease the security of the scheme at all, while allowing users to place the data wherever they want in the transaction. Instead of providing T_{commit} in the qrWit segregated area, we could provide three pieces of information:

1. **prefix data** – This is a slice of T_{commit} from byte 0 to the first byte that is part of the commitment data.
2. **commitment data** – This is the data that is actually committed in T_{commit}. This could even be a simple hash-pair or a merkle root as described above.
3. **suffix data** – This is another slice of T_{commit} from the last byte of the commitment data to the last byte in the transaction.

Concatenating these three pieces of data will create T_{commit}, which can be checked for existence in the blockchain as previously. The implementation changes that should be done are obvious.

8.3 User-Configurable Delay

One of the most common objections to our scheme is the long delay phase. The choice of how long the delay phase should be is sure to cause a massive debate in the community. Some users will want very large periods out of precaution as they believe QC to be extremely powerful, while other users will want to risk and choose a short delay period as freezing their funds for long period of times is not acceptable. As such, we will describe a variant of our scheme which offers configurable delays. To understand why the scheme is structured as it is, we will first present an unsuccessful attempt at creating a scheme with user-configurable delay.

No Delay – A possible approach to user configurable delays is to remove the idea of a delay altogether. Users will simply reveal with the same data as before, whenever

they feel confident that their commitment is immutable, even under the presence of a quantum-capable attacker. However, this strategy has a major vulnerability as attackers can listen for T_{reveal} transactions (i.e. transactions with QRWit data), retrieve the ECDSA public keys included, compute the private keys, create their own commitments, and publish their own reveal transaction linking to their commitment, all this while the owner's T_{reveal} is still in the memory pool. Thus, the attacker would be able to steal any funds he wants given that he can complete the above actions really fast.

First-Seen – To mitigate the issue described above, we could employ a "first-seen" rule for the commitments to a certain public key. However, in order to find the first commitment belonging to a key, T_{commit} will need to be tagged in some form with something linking it to the key. We cannot simply use the public key because this would reveal it, but we can use a hash of it. Hence, the commitment for data $H(pk, pk_{QR})$ would be tagged with $H'(pk)$. Miners and validators would index all the commitments by tags, only storing the first transaction for each tag. Therefore, the reveal transaction would not even need to link to the commitment transaction any more, as it would be indexed by the tag. Indeed, such a change would solve the aforementioned problem as attackers would need to revert the chain until before the creation of T_{commit} in order for their commitment to be valid. Such a chain rewrite is not possible if the user waits long enough between commit and reveal. However, this approach allows griefing to happen, i.e. adversaries could listen for commit transactions, retrieve the tag associated to it, and publish some random commitment data with the same tag, essentially invalidating the original commitment. The funds would be irrecoverable now, as they only commitment that would be accepted by the consensus rules is one with some random data in it. Although this attack is not profitable for the adversary, griefers who want to disrupt the network might employ such techniques.

First Valid Commitment

To overcome the issue of griefing we have to change the strategy from first-seen to first valid commitment. Where valid means that it must be possible to determine if the commitment was created by the owner of the public key or by an adversary. Thus, we will change the structure of the commitment data completely. The 80 byte OP_RETURN output must now contain:

1. 1 byte **flag** that can indicate to miners that this is a commitment transaction. This is needed so that miners know which UTXOs to index.
2. 32 bytes **tag,** $H'(pk)$, that can link this transaction to the public key pk being committed in it. This will be used by miners to index transactions. Note that miners will need to index all transactions with a specific tag, rather than just the first, as they do not know yet which ones are valid.
3. 32 bytes **commitment data,** $H(pk, pk_{QR})$, that represents surrogate links.

4. 15 bytes **validation data,** $\texttt{ES}_{pk}(d||\texttt{CHKSUM}(d))$, where $\texttt{ES}_k$ is a function for symmetric encryption with key k, d is some 12 bytes of random data, and $\texttt{CHKSUM}(d)$ is a 3 byte checksum of that data.

With the above data in T_{commit}, T_{reveal} does not need the proof of existence any more, as the commitment is indexed anyway. The new consensus rules would verify a surrogate as follows.

From the revealed ECDSA public key, compute $H'(pk)$ and use it to retrieve the list of commitments for this tag. Only one of these commitments is created by the real owner, and that is the first valid one. Given pk, we can check if a commit is valid by doing the following:

1. Decrypt the validation data vd (last 15 bytes) using pk as key, i.e. $\texttt{DS}_{pk}(vd)$.
2. Check if the checksum of the first 12 bytes of vd matches the last 3 bytes of vd.

Thus, this method solves the sole issue of the previous proposal as the only way for somebody to construct a valid commitment for some pk, is to have access to pk.

Furthermore, note that this method can be combined with the strategy for aggregating commitments, with a small change, i.e. The tag would be another Merkle tree root, built from hashes of public keys that are committed inside this transaction. As such, this would increase usability to the maximum, as users could commit with low costs and reveal whenever they want. The delay is configurable in the sense that it doesn't exist any more. However, we still recommend a long delay between commit and reveal as otherwise attackers could rewind the blockchain and include a malicious commitment before the actual user.

Section 9: Conclusion

To conclude this paper, we would like to reiterate our contributions and accomplishments. We believe our most important achievement is the publication of the theoretical part of this paper in a special issue (on Blockchain technology) of the Royal Society Open Science journal [62].

We presented the basics of Elliptic Curve Cryptography and the workings of the elliptic curve digital signature algorithm (ECDSA), which is the main vulnerability in Bitcoin once quantum attackers appear. To place in context this breach of security, we gave a short introduction to Bitcoin and explained how ECDSA is used and what it guarantees. Furthermore, we illustrated the exact method through which Quantum Computing enables attackers to break ECDSA and how scalable the QCs would need to be for these attacks to be efficient. As alternatives to ECDSA, we summarized some proposed quantum resistant cryptosystems, which will have to be considered by the community.

In light of the emerging threat of quantum-capable adversaries in Bitcoin, we have outlined how Bitcoin could become subject to theft of funds due to the exposure of public keys. In particular, we showed how live transaction hijacking can be achieved, and how it allows attackers to compromise any transactions published to the network.

Thus, we have proposed QRWit, a commit–delay–reveal scheme, to allow for the secure transition to a quantum-resistant signature scheme in Bitcoin. We have explained the protocol, considering each phase individually, but we also offered an alternative, more intuitive view, by regarding our changes as introducing quantum resistant surrogate keys.

Given the theoretical specification of QRWit, we have shown how the underlying protocol modifications can be implemented as a soft fork and how the initial commit phase can be done even before QRWit is actually deployed.

For the security of the transition scheme, we emphasise the need for a sufficiently long delay period and propose an initial period of 6 months in order to prevent possible blockchain reorganisation. At the same time, we also provide a future extension of QRWit which allows users to transition at their own pace if they believe a shorter or longer delay is more suitable.

Furthermore, we mentioned all the other proposals which aim to solve the same goal as QRWit and compared their usability and cost efficiency with ours. We also individually evaluated the security model of QRWit, and the specific implementation we provided.

Finally, we described a series of improvements which would further increase the usability by decreasing the cost of the overall transition and by allowing users to choose their own delay period.

Bibliography

[1] Bitcoin Cash. https://www.bitcoincash.org/. Accessed: 2018-02-18.

[2] Bitcoin Gold. https://bitcoingold.org/. Accessed: 2018-02-18.

[3] BlockSci. https://github.com/citp/BlockSci. Accessed: 2018-02-18.

[4] Coinmarketcap. http://coinmarketcap.com/. Accessed 2016-09-10.

[5] Ethereum Classic. https://ethereumclassic.github.io/. Accessed: 2018-02-18.

[6] Adam Back. https://twitter.com/adam3us/status/947900422697742337. Accessed: 2018-02-18.

[7] Adam Back. https://twitter.com/adam3us/status/948105646062391297. Accessed: 2018-05-04.

[8] Adam Back. https://twitter.com/adam3us/status/948213904668352512. Accessed: 2018-05-04.

[9] Adam Back. https://twitter.com/adam3us/status/992123846063882240. Accessed: 2018-05-04.

[10] Adam Back. https://twitter.com/adam3us/status/992378285748375552. Accessed: 2018-05-04.

[11] Adam Back. https://twitter.com/adam3us/status/992380561414152192. Accessed: 2018-05-04.

[12] D. Aggarwal, G. K. Brennen, T. Lee, M. Santha, and M. Tomamichel. Quantum attacks on bitcoin, and how to protect against them. *arXiv preprint arXiv:1710.10377*, 2017.

[13] American National Standards Institute. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). ANSI X9.62, 2005.

[14] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck. Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3. In *International Conference on Selected Areas in Cryptography*, pages 317–337. Springer, 2016.

[15] A. M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition, 2014.

[16] D. J. Bernstein. Introduction to post-quantum cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-quantum cryptography*, chapter 1, pages 1–14. Springer, 2009.

[17] A. Biryukov, D. Khovratovich, and I. Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29. ACM, 2014.

[18] Bitcoin community. Bitcoin-core source code. `https://github.com/bitcoin/bitcoin`. Accessed: 2015-06-30.

[19] Bitcoin community. Original Bitcoin client/API calls list. `https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list`. Accessed: 2018-02-18.

[20] J. Bonneau and A. Miller. Fawkescoin: A cryptocurrency without public-key cryptography. In *Cambridge International Workshop on Security Protocols*, pages 350–358. Springer, 2014.

[21] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2014. Accessed: 2016-08-22.

[22] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

[23] D. Derler, S. Ramacher, and D. Slamanig. Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In *International Conference on Post-Quantum Cryptography*, pages 419–440. Springer, 2018.

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

[25] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[26] R. Feynman. Theres plenty of room at the bottom. In *Feynman and computation*, pages 63–76. CRC Press, 2018.

[27] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[28] L. Groot Bruinderink. Towards Post-Quantum Bitcoin, 2016. Masters thesis. Eindhoven University of Technology.

[29] J. Gruska. *Quantum computing*, volume 2005. McGraw-Hill London, 1999.

[30] D. Hankerson, A. Menezes, and S. V. Springer. Guide to Elliptic Curve Cryptography.

[31] K. Ikeda. qbitcoin: A peer-to-peer quantum cash system. *arXiv preprint arXiv:1708.04955*, 2017.

[32] J. Jogenfors. Quantum bitcoin: An anonymous and distributed currency secured by the no-cloning theorem of quantum mechanics. *arXiv preprint arXiv:1604.01383*, 2016.

[33] K. P. Kalinin and N. G. Berloff. Blockchain platform with proof-of-work based on analog hamiltonian optimisers. *arXiv preprint arXiv:1802.10091*, 2018.

[34] G. O. Karame, E. Androulaki, and S. Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012:248.

[35] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):2, 2015.

[36] E. O. Kiktenko, N. O. Pozhar, M. N. Anufriev, A. S. Trushechkin, R. R. Yunusov, Y. V. Kurochkin, A. I. Lvovsky, and A. K. Fedorov. Quantum-secured blockchain. arXiv:1705.09258, 2017. Accessed: 2017-06-29.

[37] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[38] L. Lamport. Constructing digital signatures from a one-way function, 1979. Technical Report. CSL-98, SRI International Palo Alto.

[39] R. S. Lehman. Factoring large integers. *Mathematics of Computation*, 28(126):637–646, 1974.

[40] E. Lombrozo, J. Lau, and P. Wuille. BIP141: Segregated Witness (consensus layer). `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`, 2012. Accessed: 2018-02-18.

[41] M. Lundstrom. Moore's law forever? *Science*, 299(5604):210–211, 2003.

[42] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *The Deep Space Network Progress Report*, 42-44:114–116, 1978.

[43] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proc. 2013 Internet Measurement Conference*, pages 127–140. ACM, 2013.

[44] A. Menezes. Evaluation of security level of cryptography: the elliptic curve discrete logarithm problem (ecdlp). 2001. Technical Report. University of Waterloo.

[45] R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

[46] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[47] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, Dec 2008. Accessed: 2015-07-01.

[48] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *1st IEEE European Symposium on Security and Privacy, 2016*. IEEE, 2016.

[49] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. *Journal of Cryptology*, 22(2):139–160, 2009.

[50] R. Overbeck and N. Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.

[51] G. M. g. R. R. r. Pieter Wuille <pieter.wuille@gmail.com>, Peter Todd <pete@petertodd.org>. BIP009: Version Bits with timeout and delay. `https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki`, 2015. Accessed: 2018-02-18.

[52] J. Poon and T. Dryja. The bitcoin lightning network. `https://lightning.network/lightning-network-paper.pdf`, 2016. Accessed: 2016-07-07.

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

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

[55] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[56] M. Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[57] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. `http://arxiv.org/pdf/1507.06183.pdf`, 2015. Accessed: 2016-08-22.

[58] N. Schneider. Recovering bitcoin private keys using weak signatures from the blockchain. `http://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html`, 2013. Accessed: 2018-02-18.

[59] N. Sendrier. Code-Based Cryptography. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 215–216. Springer US, Boston, MA, 2011.

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

[61] Y. Sompolinsky and A. Zohar. Bitcoin's security model revisited. *arXiv preprint arXiv:1605.09193*, 2016.

[62] I. Stewart, D. Ilie, A. Zamyatin, S. Werner, M. F. Torshizi, and W. J. Knottenbelt. Committing to Quantum Resistance: A Slow Defence for Bitcoin against a Fast Quantum Computing Attack.

[63] L. Tessler and T. Byrnes. Bitcoin and quantum computing. *arXiv preprint arXiv:1711.04235*, 2017.

[64] K. K. A. Thissen. Klepto for post-quantum signatures, 2016. Bachelors Thesis. Eindhoven University of Technology.

[65] Tim Ruffing. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-February/015758.html`. Accessed: 2018-02-18.

[66] Tim Ruffing. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-January/015659.html`. Accessed: 2018-02-18.

[67] Tim Ruffing. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-January/015619.html`. Accessed: 2018-02-18.

[68] M. Veldhorst, C. Yang, J. Hwang, W. Huang, J. Dehollain, J. Muhonen, S. Simmons, A. Laucht, F. Hudson, K. Itoh, et al. A two-qubit logic gate in silicon. *Nature*, 526(7573):410, 2015.

[69] T. Watson, S. Philips, E. Kawakami, D. Ward, P. Scarlino, M. Veldhorst, D. Savage, M. Lagally, M. Friesen, S. Coppersmith, et al. A programmable two-qubit quantum processor in silicon. *Nature*, 555:633637, 2018.

[70] Y. Yarom and N. Benger. Recovering OpenSSL ECDSA nonces using the FLUSH+ RELOAD cache side-channel attack. *IACR Cryptology ePrint Archive*, 2014:140, 2014.