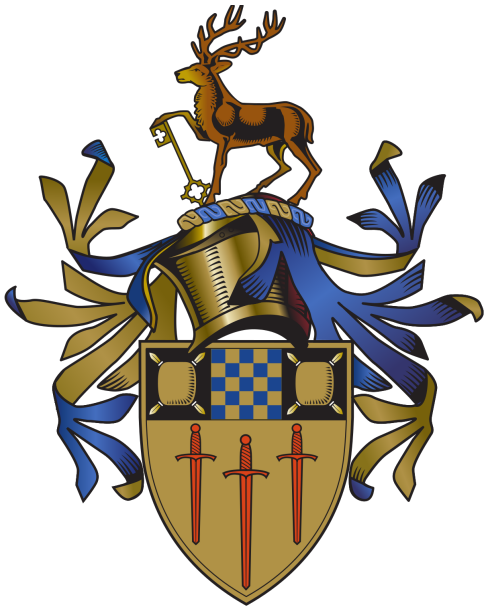
University of Surrey

Faculty of Engineering & Physical Sciences



NetNotes: adding untraceability to Mimblewimble

by

Marco Endrizzi

URN: 6573978

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

Supervised By

Prof. Liqun Chen

Guildford, May 2023

Declaration of Originality

I confirm that the submitted work is my own work and that I have clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin® UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

Acknowledgements

I would like to thank my supervisor, Prof. Liqun Chen, for her valuable guidance and support throughout this dissertation project. Her vast knowledge, insightful feedback, and encouragement have immensely helped me to improve the quality of this work.
I would also like to express my gratitude to my parents and girlfriend for their love and unwavering support during my academic journey, as well as to my brother for introducing me to the world of computers. Their encouragement and belief will always be the main driving force behind my personal growth.

Abstract

In 2016, pseudonym Tom Jedusor logged onto a Bitcoin IRC channel and uploaded a text file hosted on a TOR server, before disappearing forever. This document described the Mimblewimble scheme, today known as both a private and extremely lightweight cryptocurrency protocol thanks to its novel way of structuring and storing transactions on the blockchain. Nonetheless, despite initially being thought to be a privacy solution on par with Monero [1] and Zerocash [2], the protocol was revealed to be flawed in 2019 [3].

This work proposes a new protocol, NetNotes, which is intended to build upon Mimblewimble and fix its privacy shortcomings. The novel protocol has been implemented in the Rust programming language and benchmarking has been performed to compare its performance to other similar schemes. Overall, the NetNotes protocol is competitive in terms of performance, but sacrifices Mimblewimble's scalability properties to achieve its additional privacy features. Furthermore, a second contribution is also presented in the form of having optimised the only Rust implementation of One-out-of-Many proofs [4].

Contents

List of Figures

List of Tables

Glossary

Atomic swap	A method of exchanging one cryptocurrency for another without the need for a trusted third party.
Double-spending	The act of using the same digital token for more than one transaction.
Fiat money	A government-issued currency not backed by a physical commodity.
Fork	A change in the protocol of a blockchain that creates a split in the blockchain network.
Know your customers	A process that financial institutions use to verify the identity of their clients.
Ledger	A book or other collection of financial accounts.
Node	Device participating in a blockchain network.
Smart contract	A program stored on the blockchain that runs when predetermined conditions are met.
Wallet	A software program that stores private and public keys and interacts with the blockchain nodes to enable users to send and receive cryptocurrency.

Abbreviations

ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
KYC	Know Your Customer
PLONK	Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge
PoS	Proof-of-Stake
PoW	Proof-of-Work
UTXO	Unspent Transaction Output
zkSNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

CHAPTER 1

Introduction

1.1 A primer on electronic cash

Ideas for a digital currency first emerged in 1983, when American cryptographer David Chaum published a paper [5] outlining an early form of anonymous cryptographic electronic money. Chaum later built upon his early ideas by founding Digicash and developing the first cryptographic electronic money: eCash [6]. Although DigiCash went bankrupt in 1998, the ideas the company put forward played an important role in developing later digital currencies. It also served as an important lesson: digital cash needed to be decentralised, without a single point of failure.

In 1997, Adam Back tried to solve email spam by introducing a Proof-of-Work algorithm called Hashcash [7]. The algorithm required users to use their computer power to solve a cryptographic puzzle in order to send an email. Since this process required waiting a length of time proportional to the user's computing power, it aimed at discouraging spam emails by making them costly and time consuming.

Despite its name, HashCash did not involve transacting digital cash; many attempts were however made to create a decentralised digital cash system based on its Proof-of-Work protocol. For instance, in 1998, Wei Dai proposed an anonymous, distributed cash system nicknamed B-Money [8] and later the same year, Nick Szabo designed a mechanism for a digital currency called Bit Gold [9]. Although these proposals were never implemented, they laid the foundation for the ground-breaking 2008 Bitcoin paper [10] by pseudonym Satoshi Nakamoto — as well as the subsequent launch of the first blockchain network in 2009.

The Bitcoin paper introduced the now well-known Blockchain data structure, a distributed and immutable database (also known as a *ledger*), which used in conjunction with Proof-of-Work, became the first decentralised solution to solve the dreaded double-spending problem: a fundamental flaw through which the same digital token could be spent more than once.
Today, more than a decade later, both Bitcoin and the blockchain industry that was spawned by it have grown massively, with over $1 trillion dollars being held across over 20000 different cryptocurrencies [11]. It could be argued that Satoshi's original vision – a financial network that transcends borders, states and private ownership – is more relevant than ever today.

1.2 Problem Statement: Bitcoin is not private

There is a common misconception that since cryptocurrencies use public-key cryptography to identify different participants, they offer privacy. In fact, even Nakamoto's 2008 paper [10] had a "Privacy" section, where it is stated that by using a new key pair for every transaction, the information would be comparable to stock exchanges, where "the 'tape' is made public, but without telling who the parties were" [10]. While it holds true that in a cryptocurrency address provides an alias that cannot be linked to a real life identity (as it does not reveal any identifiable information), over time, techniques have been developed to break this anonymity.

Forensic analysis of the public blockchain has actually spawned a whole industry: *blockchain analysis*. Blockchain analysis is usually conducted by private companies with the goal of uncovering valuable insights regarding all the different actors transacting in a cryptocurrency. Such forensic analysis has in fact been successfully used by the FBI and other government agencies to track and convict various criminals involved with unlawful cryptocurrency payments [12] [13].

The problem lies in the public nature of the blockchain, which due to containing all historical records of transactions made for a cryptocurrency, allows for analysing the flow of funds of every single transaction. As such, the majority of cryptocurrencies are now considered to be only *pseudo-anonymous*, and furthermore, it has become clear that *complete privacy* cannot be achieved by merely concealing one's identity; it also requires keeping one's actions hidden.

Complete Privacy

Since privacy is a complex and multi-faceted concept, this dissertation will define the notion of *complete privacy*, based on [14]'s properties of a fully anonymous cryptocurrency. To satisfy the definition of *complete privacy*, a cryptocurrency protocol should offer the following properties:

Unlinkability. For any two transaction, it should be unfeasible to determine whether they were sent to the same receiver. As such, no two transactions should be sent to the same address.

Untraceability. For any transaction, every possible sender should be equiprobable. This implies an observer will not be able to follow the money trace for a coin.

Confidential transactions. The value of every transaction should only be known to the transacting parties. On the blockchain, these amounts should be obfuscated to be indistinguishable from random data.

1.3 Aims & Objectives

1.3.1 Aim

The single aim of this dissertation is to contribute to the field of privacy-preserving cryptocurrencies by designing a novel cryptographic scheme enabling *completely private* digital transactions. The protocol's performance should also be strong enough to justify its potential implementation in a cryptocurrency that is suitable for everyday payments.

1.3.2 Objectives

The specific objectives for this dissertation project are as follows:

Objective	Description
1	Research and learn about the cryptography used in cryptocurrencies
2	Learn about the current state of the art in privacy-preserving cryptocurrency solutions
3	Design a novel cryptographic scheme that achieves "complete privacy" and is competitive in terms of performance
4	Implement the novel protocol in a chosen programming language
5	Perform benchmarking of the novel protocol to provide a performance comparison with other privacy-preserving schemes
6	Analyse the security properties of the novel protocol and provide a general evaluation of the scheme's strengths and weakness compared to other privacy-preserving protocols

Table 1.1: Project objectives.

These objective will be used as a reference in the Conclusion chapter to determine if the project was successful.

1.3.3 Limitations

No limitations for this project have been identified in terms of resources, data access or ethical consideration.

The author has identified *time constraints* and limited exposure to cryptography as the primary limitations for this project. The first project objective has especially been estimated to require a substantial amount of time. As objectives 3-5 are dependent on the successful completion of objective 1, any delays in the completion of the latter could cause a delay in the overall project timeline. In such an event, the project scope can be scaled down by eliminating objectives starting with objective 5 and progressing backward.

1.4 Dissertation Structure

Table 1.2 gives a brief overview of the structure of this dissertation.

Chapter	Name	Brief Description
1	Introduction	Sets the scope of the project by providing its aims and objectives
2	Cryptographic Background	Provides a brief overview of the cryptography used in privacy-focused cryptocurrencies
3	Literature Review	Presents research findings regarding privacy in the blockchain field
5	Mimblewimble	Describes the Mimblewimble scheme, upon which the new protocol is based
6	NetNotes	Thoroughly describes the novel protocol
6	Protocol implementation	Covers the implementation and testing of the protocol in a chosen programming language
7	Performance & Evaluation	Provides an evaluation of the protocol's performance and security in comparison to other privacy-preserving protocols
9	Conclusion	Draws conclusions on the project's success and discusses future work

Table 1.2: Overview of the dissertation's structure.

CHAPTER 2

Cryptographic Background

The following chapter will provide an introduction to the cryptographic primitives used in this dissertation and in general for private cryptocurrencies. Note that the following sections have been written with the assumption that the reader does not have any prior knowledge of cryptography.

2.1 Public-key Cryptography

In general, cryptography involves keeping information secret by transforming it into a form that only the intended recipient can understand. For instance, take three friends Alice, Bob and Charlie. Alice wants to send an *encrypted* message to Bob, so that even if Charlie intercepts it, he won't be able to *decrypt* it. Alice could employ a simple encryption scheme by shifting letters around by a predefined amount of characters; this is known as a Caesar cipher, one of the oldest and simplest encryption schemes.

There are however two glaring problems with this approach:

1. **Easy to break.**: With today's computing power, trying all possible 26 keys (i.e. alphabet) is trivial. This means that if the message is intercepted, Charlie will easily be able decrypt it and read it.
2. **Requires communication.**: Alice needs to communicate the key to Bob for him to be able to decrypt the message. If Bob lives in a different country, Alice has no choice but to either trust a courier, or perhaps, use the internet. Both of these methods are not only unsafe, but if they were intercepted, Alice would have no way of knowing.

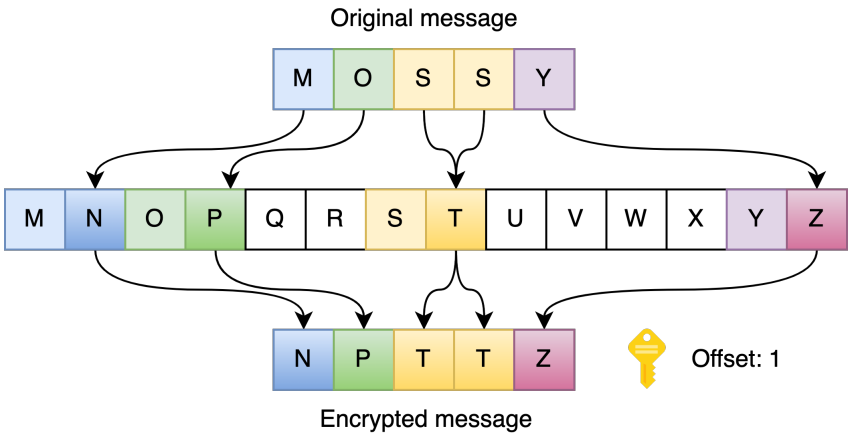


Figure 1: A caesar cipher example using the word "mossy" and a key of 1.

2.1.1 Diffie-Hellman Key Exchange

In 1976, Whitfield Diffie and Martin Hellman solved the aforementioned problems by creating the first practical method to establishing a shared secret key over an insecure communications channel [15]. Their novel *asymmetric* encryption system employs two related – yet separate – keys, namely a *public key* for encryption, and a *private key* for decryption. This is in contrast to traditional "symmetrical" cryptography, which uses a single key to encrypt a message between a sender and a recipient.

Public-key cryptography works thanks to the fact that despite the two asymmetrical keys being mathematically related, it is not possible to derive one from the other. In fact, a lot of modern cryptography relies on the existence of so-called *trapdoor functions*: functions that are easy to compute in one way, but not the other.

2.1.2 RSA

The first public-key cryptosystem widely used for secure data transmission, RSA, was created in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [16] and named after their surnames.

RSA is based on the mathematical properties of large prime numbers, and to illustrate its trapdoor function, take the following example: given a large prime number such as 27292996856087, can you easily find two prime numbers that multiply to give such result? In essence, RSA's security is based on the difficulty of factorising the product of two large prime numbers, where the product is the *public key*, and the two prime numbers are the *private key*.

The RSA cryptosystem can be summarised in three steps:

Key Generation. To generate the RSA keys, two large prime numbers, p and q, are chosen and multiplied together to produce $n = pq$. A third number, e, is also chosen and must be coprime to the value $\Phi(n) = (p-1)(q-1)$. A fourth number, d, is calculated using the extended Euclidean algorithm. The *public key* is then defined as (n, e), while the *private key* as the secret value d.

Encryption. To encrypt a message, the sender uses the public key of the recipient. To calculate the encrypted message m, the message is raised to the power of e modulo n. The result is the ciphertext $c \equiv m^e \pmod{n}$.

Decryption. To decrypt the message, the recipient uses their private key. The ciphertext is raised to the power of d modulo n. The result is the original message $m \equiv c^d \pmod{n}$.

To break RSA, an attacker would essentially need to figure out the private key d from the public key (n, e). This is equivalent to factoring n into p and q to get $\Phi(n)$ and d, which is computationally unfeasible.

Despite this, factorisation has become a popular problem and many algorithms have been developed to brute-force factorising large numbers; this fact, coupled with the ever-increasing computational power of computers, has made RSA less secure over time. Currently, RSA is no longer considered secure for modern applications for key sizes smaller than 2048 bits, whose size makes it unsuitable for low-end devices. As a result, a more robust trapdoor function had to be developed.

2.2 Elliptic-curve Cryptography

To provide a more secure alternative to RSA, Elliptic-curve cryptography (ECC) was introduced in 1985 by Neal Koblitz and Victor Miller who both independently developed the idea [17][18].

The main advantage that ECC has over RSA is providing equivalent security with much smaller key sizes, which renders it more efficient and therefore suitable for devices with limited computational power. The enhanced security is due to ECC's trapdoor function, the discrete logarithm problem [19], which is considered to be more difficult to solve – as well as quantum resistant – compared to factorisation of prime number.

2.2.1 Operations on elliptic curves

An elliptic curve is a curve symmetric about the x-axis and defined by the equation $y^2 = x^3 + ax + b$. In short, from such curves a public-private keypair can be generated, which can then be used to encrypt and decrypt message by performing mathematical operations on the curve. Two of the most important such operations on elliptic curves are described in the following sections: *point addition* and *point multiplication*.

Point addition

Addition on elliptic curves is not much different to normal addition, and it retains its associative and commutative properties. In essence, point addition involves adding (or subtracting) two points on the curve, which will in turn result in a new point on the curve.

Definition. Given two points P and Q on the elliptic curve, their sum is defined as the negation of the point resulting from the intersection of the curve and the line connecting P and Q.

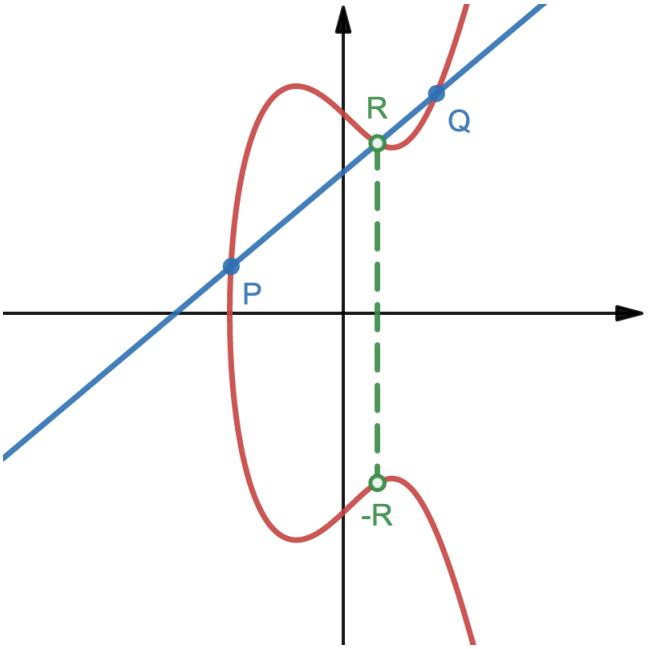


Figure 2: Visualisation of point addition in elliptic curves.

Looking at Figure 2, R can be observed as the point where the line between P and Q intersects the curve. The point addition however results in point $-R$, the negation of point R.
This is due to the inherent property of elliptic curves, where the sum of three aligned points will result in zero. Keeping in mind that when drawing the line passing through P and Q, point R will be aligned, this can be summarised as:

$$P + Q + R = 0$$

which can be rewritten as:

$$P + Q = -R$$

where $-R$ is the negation (symmetric to the x-axis) of the point R.

To a third-party, addition of points on elliptic curve is like hopping around on the curve to a different seemingly random point; that is unless you know the exact points used to get there.

Point multiplication

Another fundamental operation in ECC is point multiplication, which is not only used to generate the public key from the private key, but is also used for encryption and decryption of messages. Despite its name, point multiplication does not involve multiplying two points on a curve.

Definition. Given a point P on the curve, multiplication by an integer d result in a new point R on the curve, such that R is the result of adding P to itself d times.

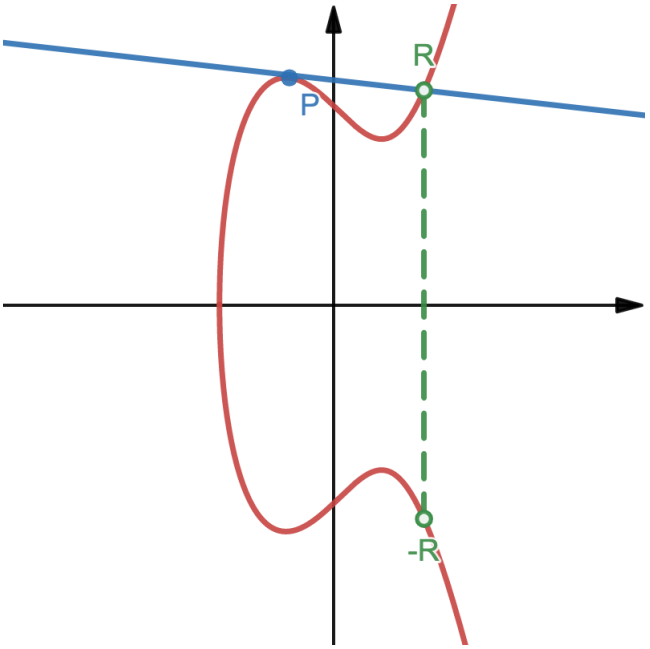


Figure 3: Visualisation of point multiplication in elliptic curves.

This operation can be easily demonstrated by using $d = 2$, such that $P + P = 2.P$ (where the dot notation "." represents the point multiplication operation). In figure 3, the tangent to P can be seen as passing through three points: 2 of them being P and one R. Similarly to point addition, this can be written as $P + P + R = 0$ and rewritten into the point multiplication $2.P = -R$.

Generating Key Pairs

An ECC system defines an elliptic curve E and a publicly known "generator" point G. In this system:

- A **private key** (d) is a randomly chosen 256-bit integer
- A **public key** (Q) is an Elliptic curve point represented as the coordinates $\{X, Y\}$ and derived by point multiplication between generator point G and the private key d, such that $Q = d.G$

Despite Q and G being public, an attacker will not be able to derive private key d due to the difficulty of the discrete logarithm problem (i.e. EC multiplication is easy, EC division is hard).

2.3 Hash Functions

A hash function is a mathematical function that accepts an input (or "message") and returns a fixed-length string of characters (a "digest"), that is unique to the input's unique values. More formally, a hash function's properties are as follows:

Deterministic. Given the same input, the function will always produce the same digest.

Fixed-length. A hash digest will always be of fixed length, regardless of the size of the input.

One-way trapdoor. Given a digest, it is computationally unfeasible to find its original input.

Uniform distribution. Changing an input by one character does not change the digest in a deterministic way. Rather, the whole output changes uniformly, which makes it impossible to deduce inputs based on marginal changes. This is also known as the *avalanche effect*.

Collision Resistant. It is computationally unfeasible to find two inputs producing the same digest. Although it's extremely unlikely that two different inputs will produce the same output, it is impossible to fully eradicate collisions.

Figure 4: Hashing example using the SHA-256 algorithm.

These properties make hash functions a very versatile cryptographic primitive used in a variety of applications today:

1. **Integrity Checks**: Thanks to their *fixed-length*, *uniform distribution* and *deterministic* properties, hash functions are ideal for ensuring data integrity. For example, by including a hash digest along a message, the receiver can check if the message was altered in transit by comparing the received hash with the hash digest of the received message.
2. **Indexing**: Hash functions can be used to create an efficient key-value *hash table* data structure. By hashing the keys, the resulting indexes used to locate values are ensured to be unique and evenly distributed (thanks to the *collision resistant* and *uniform distribution* properties).
3. **Proof-of-Work**: Hashcash [7] introduced the concept of using hash functions as a denial of service counter-measure, by requiring users to provide a digest with a specific pattern (e.g. beginning with four 0s) before being able to perform certain actions. This essentially forces users to use their time and computational power to create the demanded hash digest, as due to the hash function's *one-way property*, they can only rely on brute-forcing random inputs to find a valid hash digest.

2.4 Digital Signatures

A digital signature is a cryptographic scheme for verifying the authenticity of digital messages or documents. As digital signatures use asymmetric cryptography, its most popular implementations are RSA and ECDSA, a variant using elliptic-curve cryptography.

Creating a digital signature is simple and involves a combination of public-key cryptography and hash functions:

1. **Hash**: The prover uses a hash function on the message or document to be signed.
2. **Encrypt**: The prover then uses his private-key to encrypt the digest created in step 1 in order to form the digital signature.
3. **Verify**: At this point, the prover sends his digital signature along with a document or message to a verifier. To verify the signature, the verifier decrypts it with the prover's public-key, before hashing the received message (or document) with the same hash function used by the prover. If the values resulting from these two separate operations are identical, the digital signature is successfully validated, and the message can be therefore considered authentic and unmodified.

The process of creating and verifying a digital signature is illustrated in Figure 5. To understand how such protocol provides *integrity* and *authenticity* for a message, take the case of a malicious actor being successful in altering a message or document in transit. Once the receiver runs the hash function mentioned in step 1, the resulting value would be different from the decrypted digital signature (as the sender encrypted a hash digest of the original data) and the verification would therefore fail, thus preserving *integrity*.

To understand how *authenticity* is ensured, consider a possible attack where a malicious actor pretends to be the sender. In this case, as the receiver would use the public key of the true sender, it would be impossible for him to decrypt the digital signature created with the malicious' actor private key.

There are however many other properties that a digital scheme must satisfy to be considered secure:

1. **Non-repudiation**: A digital signature should be considered evidence that the signer signed a message, so that the signer cannot later deny having signed it.
2. **Uniqueness**: A signature for a message should not be able to be used for another message.
3. **Integrity**: A digital signature should ensure that the original contents of the message have not been modified in transit.
4. **Authenticity**: A valid signature should imply that the signer deliberately signed a message; i.e. it should be able to verify the signer's identity
5. **Unforgeability**: For any *public-key*, signatures should not be forgeable by anyone but the owner of the corresponding *private-key*.

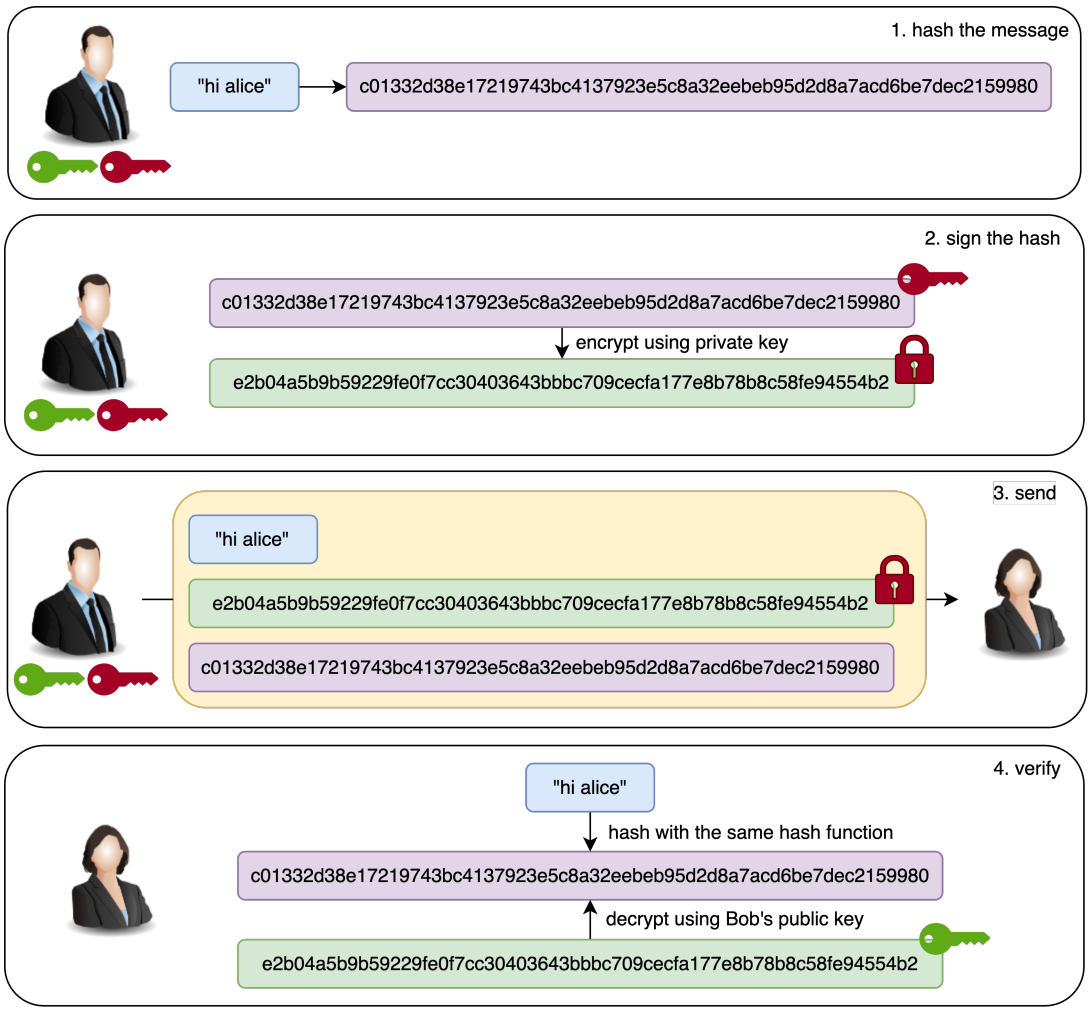


Figure 5: An example of a digital signature on message "hi alice".

2.4.1 Schnorr signatures

Schnorr signatures are a special type of digital signature first proposed by Claus-Peter Schnorr in 1991 [20]. This type of signatures represented a significant improvement over the traditional digital signatures based on ECDSA, thanks to their use of a single nonce for both the generation and verification of signatures, which simplified the scheme and eliminated the potential for nonce reuse; a known security vulnerability in ECDSA.

Given a message m, an ECC generator point G, private-key d and public-key $P = d.G$, the Schnorr signature scheme is defined as follows:

Signing. Choose a random nonce k and compute its corresponding *public* point $R = k.G$. Calculate the challenge $e = \mathscr{H}(R||P||m)$, where $\mathscr{H}$ is a cryptographic hash function and $||$ is the concatenation operator. Finally, compute the signature value $s = k + e * d$.
The complete Schnorr signature will be the pair (s, R).

Verifying. A verifier provided with message m, public-key P, and signature (s, R) will be able to calculate the challenge $e = \mathscr{H}(R||P||m)$. To verify the signature, the following equation: $s.G \stackrel{?}{=} R + e.P$ has to hold true. This works as $s.G \equiv k.G + e.(d.G) \equiv R + e.P$

Schnorr signatures offer several scalability advantages over traditional digital signatures. For instance, they are faster, more efficient, and allow batch verification as well as merging of multiple signatures into a single valid signature. This makes them ideal for use in cryptocurrencies, where several transactions need to be verified as fast as possible.
Schnorr signatures are also useful for saving space in a blockchain. For example, in Figure 6, Alice wants to prove that she owns three different inputs in a transaction. With a single Schnorr signature, she proves ownership of all three inputs without having to create and store three separate digital signatures.

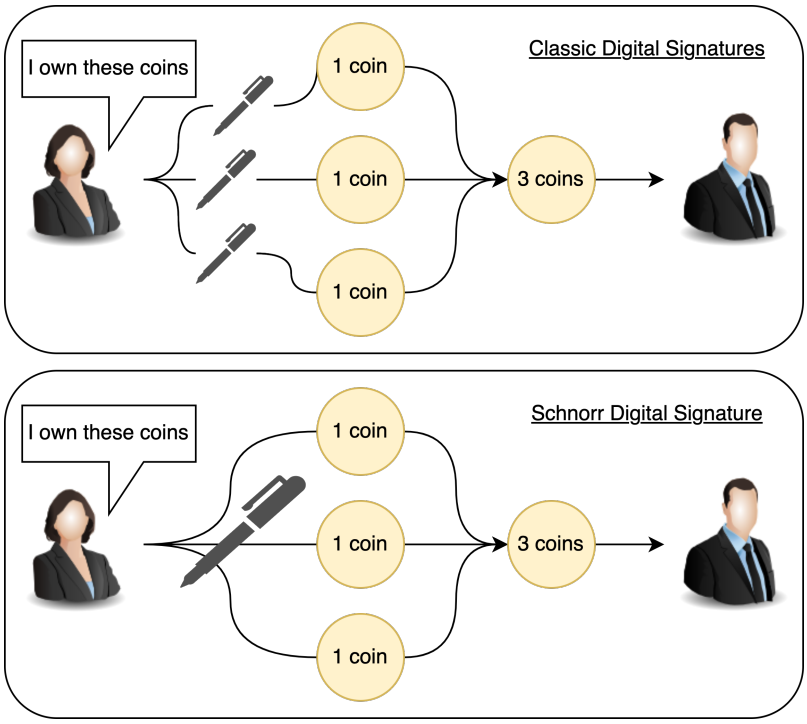


Figure 6: Example showcasing Schnorr signature aggregation.

2.5 Commitment schemes

A commitment scheme is a type of encryption that allows one party to commit to a value without revealing it to another party, such that the committed value can later be revealed but cannot be changed.

An example of such scheme is shown in Figure 7, where a chest represents the committed (i.e. encrypted) value. In the example, Alice commits to a value and sends the locked, encrypted chest to Bob. At this point Bob cannot do anything with the box, but wait for Alice to reveal the secret key to reveal the hidden value. Once she does, Bob can "unlock" the chest and verify the value is the same as Alice's claim. As the chest was in Bob's possession at all times, it is impossible for Alice to have cheated and revealed a different value. Alice therefore *committed* to the secret value when sending the chest.

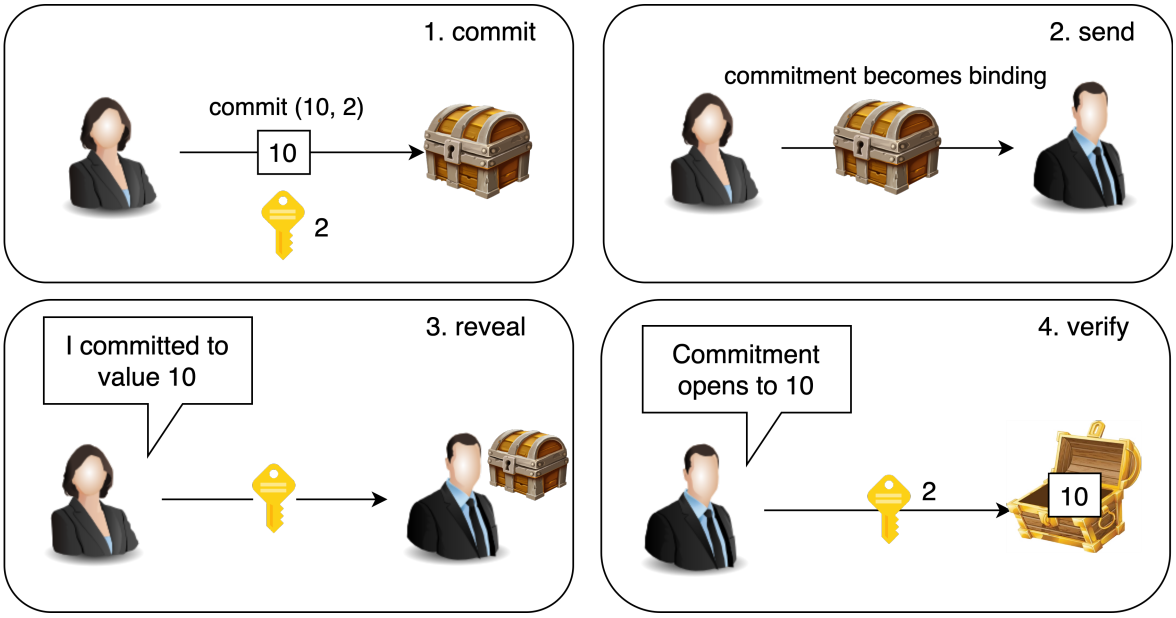


Figure 7: Commitment scheme example using Pedersen commitment.

2.5.1 Hash-based commitment

It is possible to create a simple commitment scheme using a secure hash function. For instance, take a commitment $C = \mathscr{H}(message||blinding\_factor)$, where $||$ is concatenation.

Bob can send the commitment C to Alice, who due to the *one-way* property of hash functions, is unable to "open" it (reveal the original message). Once Bob wants to reveal the commitment, he sends the message, as well as the blinding factor, to Alice. To verify the commitment, Alice simply runs the same hash function with provided parameters and verify that the output digest equals the commitment.

Notice that it is necessary to append a *blinding factor* to the message to prevent Alice from being able to compare the received commitment to a pre-computed list of different message hashes. By introducing randomness, the commitment becomes unique, and therefore *hiding*.

2.5.2 Commitment schemes properties

Commitment schemes are generally defined by two properties:

Hiding: Receiving a commitment to a value should provide the receiver with no information about the value.

Binding: The sender should not be able to compute a second commitment that encrypts to the same value as his original commitment; otherwise it could be possible to use a different key to open an already-sent commitment to a different value.

These two properties can be split into two security levels:

Computationally binding/hiding properties are secured by the underlying mathematical problem that the commitment scheme is based on (e.g. the discrete logarithm problem [19]). This means that the property is secure as long as the underlying problem is too hard to be solved in reasonable time for an attacker.

Perfectly binding/hiding ensures it is impossible to break the property even if the attacker has unlimited computational power.

It is important to note that a commitment scheme is always a compromise between one of the properties not being perfectly secure.

For instance, assume an attacker with infinite processing power and a *perfectly binding* commitment scheme (i.e. a commitment can only have a single value associated with it). The attacker can simply compute the commitment for every possible value until he finds a pair that outputs the commitment. This would identify the value of the commitment, and therefore break the *perfectly hiding* property.

To prevent this, a commitment could be made *perfectly hiding* by allowing multiple values to output to the same commitment. This prevents the attacker from knowing which pair corresponds to which commitment, but allows him to compute different value/blinding factor pairs that can open the commitment to a different value compared to the original (breaking the *perfectly binding* property).

The hash-based commitment scheme described above is a good example of a *perfectly hiding* and *computationally binding* scheme, as hash collisions are computationally unfeasible to create on demand.

2.5.3 Homomorphic commitments

More advanced commitment schemes can possess additional useful properties. For instance, a *homomorphic* commitment allows to perform mathematical operations on the encrypted values, as if they were not encrypted. This property makes them ideal for implementing *confidential transactions* in cryptocurrencies, as outputs and inputs can be added and subtracted together without revealing the underlying values of coins.
The two most common types of homomorphic commitments are Pedersen commitments [21] and ElGamal commitments [22], explained in the following sections.

Pedersen commitments

Pedersen commitments are a *perfectly hiding* (can never leak values) and *computationally binding* commitment scheme based on the hardness of the discrete logarithm problem. A Pedersen commitment scheme over elliptic curves enables to commit to $m \in Z_p$ by picking a random blinding factor $r \in Z_p$ and computing $C = m.G + r.H$, where G and H are nothing-up-my-sleeve-points.

In essence, Alice can send a commitment $C = m.G + r.H$ to Bob, and later reveal the value m and blinding factor r. Bob can then verify that the commitment is correct by computing $C' = m.G + r.H$ and verifying that $C' \stackrel{?}{=} C$. It is impossible for Alice to change the value of

m without Bob noticing, and it is impossible for Bob to compute the value of m or r without Alice revealing them.

Due to Pedersen commitments using elliptic curves – and therefore being *homomorphic* – it is also possible to perform standard addition and subtraction on the encrypted values. For instance, given two commitments $C_1 = m_1.G + r_1.H$ and $C_2 = m_2.G + r_2.H$, their sum commitment $C_{sum} = C_1 + C_2$ can be simply computed as:

$$\begin{aligned} C_{sum} &= m_1.G + m_2.G + r_1.H + r_2.H \\ &= (m_1 + m_2).G + (r_1 + r_2).H \end{aligned} \tag{2.1}$$

where the + sign indicates point addition on the elliptic curve. The resulting commitment C_{sum} will now indeed be hiding the value of $m_1 + m_2$, and the blinding factor $r_1 + r_2$.

ElGamal commitments

ElGamal commitments are a *computationally hiding* and *perfectly binding* (values can never be changed) commitment scheme also based on the hardness of the discrete logarithm problem.

An ElGamal commitment scheme over elliptic curves enables to commit to $m \in Z_p$ by picking a random blinding factor $r \in Z_p$ and computing $C = m.G + r.H, r.J$, where G, H and J are nothing-up-my-sleeve-points. The addition of the third point J is what makes the commitment unique and therefore perfectly binding. It posses the same homomorphic properties as Pedersen commitments.

2.6 Zero Knowledge Proofs

Zero-knowledge proofs, as first described Goldwasser *et. al.* [23] are a verification method that allows one party to prove to another that they know a specific piece of information without revealing the information itself. A type of zero-knowledge proof has in fact been already explored in the previous sections: digital signatures. Recall that digital signatures can verifiably prove that someone is in possession of a private-key for a corresponding public-key, without revealing the private-key itself.

To further understand how it is possible to prove knowledge of a piece of information without revealing it, let us consider a very common example of proving to a colourblind verifier that two balls in your possession are red and green.
Firstly, the prover gives the verifier both balls and asks him to shuffle them behind their back. Secondly, the prover asks the verifier to pick one of the balls and show it to him; the prover then declares whether the verifier switched balls or not. This process repeats until the colourblind verifier is convinced that the balls are of different colours (due to the prover correctly guessing the colour every time), even though he's unable to verify this fact for himself.

More formally, a zero-knowledge proof of some statement must satisfy the following three properties:

Completeness. If a statement is true, the verifier will always be convinced by the prover

Soundness. The prover must not be able to convince the verifier of an untrue statement, except with some very small probability

Zero-knowledge. The verifier must not learn any information about the statement from the proof.

Interactive protocols (also known as Σ-protocols) are a type of protocol that supports zero-knowledge proofs. They involve three rounds of communication:

1. *Prover:* The prover commits to a statement and sends it to the verifier.
2. *Verifier:* With the commitment, the verifier challenges the prover to produce a proof.
3. *Prover:* On receiving the verifier's challenge, the prover creates a verifiable response and sends it back to the verifier.

After this interaction, the verifier can be convinced that the prover knows a specific piece of information, without ever learning what the information is. That is because the prover cannot know ahead of time what challenge the verifier is going to send, and since the values have already been committed to during the first phase, it can only construct a valid proof with the original. Such interactive protocol can be turned into a non-interactive protocol by using the Fiat-Shamir heuristic [24], which uses a random oracle (i.e. a hash function) to generate challenges.

The following sections will introduce two zero-knowledge proof protocols used later in this dissertation: *range proofs* and *one-out-of-many proofs*.

2.6.1 Range proofs

Range proofs are a type of zero-knowledge proof that allows one to prove to another party that a secret committed value lies within a certain range, without revealing the value itself. Formally, let $v \in Z_p$ and let $C \in G$ be a Pedersen commitment to v using the randomness r. Then the proof system will convince the verifier that the committed value $v \in [0, 2^n)$.
Bulletproofs [25] and its improvements Bulletproofs+ [26] and Bulletproofs++ [27] are a specific type of range proofs, created to improve the efficiency and scalability of the scheme. Due to their quick generation times and small proof sizes, they are ideal to be used in practical applications, such as verifying that *confidential transactions* in cryptocurrencies are non-negatives.

The following overview of how Bulletproofs work is largely based on an excellent blog post by Cathie Yun [28]. Please refer to the original paper [25] for a more thorough explanation.
Recall that the statement to be proven in zero-knowledge is the following: $0 \leq v < 2^n$. For this statement to be true, v must be a binary number of length n. Assuming $v = 3$ and a range proof is checking whether v is in the range $[0, 2^4)$, v should be able to broken up into a bit representation that is 4 bits long, as shown in Equation 2.2.

$$v = \sum_{i=0}^{n-1} \left(\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 2^3 \\ 2^2 \\ 2^1 \\ 2^0 \end{bmatrix} \right) = 2^1 + 2^0 = 3 \tag{2.2}$$

Bulletproofs want to represent the aforementioned claim in the form of an inner product, which is the total sum of the products of each corresponding element in the two vectors. Taking equation 2.2 as an example, assuming the two vectors are $a = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ and $b = \begin{bmatrix} 2^3 \\ 2^2 \\ 2^1 \\ 2^0 \end{bmatrix}$, the inner product of a and b is written as $\langle a, b \rangle$ and is equal to 3 ($2^1 + 2^0$).

The statement $0 \leq v < 2^n$ can be therefore be broken into two separate, smaller statements:

1. $v = \langle v_{bits}, 2^n \rangle$ – i.e. v is represented in binary (see Figure 2.2)
2. $v_{bits} \circ (v_{bits} - 1^n) = 0^n$ – i.e. bits are in fact bits (0s and 1s)

These two statements are then combined using challenge scalars and blinding resulting in an inner proof statement $\langle a, b \rangle$. Thanks to Bulletproofs' extremely efficient inner product proof, this statement can be proven true in $O(log(n))$ space and time rather than the usual $O(n)$ needed for inner product calculation; this is what makes the Bulletproofs especially fast and efficient. A full explanation of how Bulletproofs achieve their faster inner product proof can be found in the `bulletproofs` library's notes [29].

2.6.2 One-out-of-Many proofs of a commitment to zero

A One-out-of-Many proof is a zero-knowledge membership proof that one out of N public commitments $C_0, ..., C_{N-1}$ opens to 0. It has been first introduced by Jens Groth and Markulf Kohlweiss [4] in 2015 and has been further improved by Bootle *et al.* [30], who generalised their system to work for n-ary trees in order to achieve better performance and smaller proof sizes. The latter is especially a very efficient construction of membership proofs, where the proof size only grows logarithmically with the size of the anonymity set.

In their paper [30], Bootle *et al.* offer various *Sigma* protocols, of which two are relevant for this work. The first *Sigma* protocol can be used to prove a commitment B has an opening consisting of a sequence of bits, where for each sequence there is exactly a single 1 bit. Such relation can be seen in equation 2.3:

$$\mathcal{R}_1 = \left\{ \begin{matrix} (B, (b_{0,0}, \ldots, b_{m-1,n-1}, r)) : \\ (\forall i, j : b_{j,i} \in \{0,1\}) \wedge (\forall j : \sum_{i=0}^{n-1} b_{j,i} = 1) \wedge B = Com_{ck}(b_{0,0}, ..., b_{m-1,n-1}; r) \end{matrix} \right\} \tag{2.3}$$

In simpler terms, this relation aims to prove that the committed numbers are in fact bits (similarly to Bulletproofs) and that the sum of each matrix column is exactly one (i.e. each column has a single "1" bit). More formally, the relation is trying to prove $b_{j,i}(1 - b_{j,i}) = 0$ for every i, j and that $\sum_{i=1}^{n-1} b_{j,i} = 1$. The complete Σ protocol is shown in Protocol 1.

Protocol 1: Σ-protocol for relation R_1. Adapted from [30].

$\mathcal{P}_1(ck, B, (b_{0,0}, \ldots, b_{m-1,n-1}, r))$		$\mathcal{V}_1(ck, B)$
$r_A, r_C, r_D, a_{j,1}, \ldots, a_{j,n-1} \leftarrow \mathbb{Z}_q$		
$\forall j : a_{j,0} := -\sum_{i=1}^{n-1} a_{j,i}$		
$A := Com_{ck}(a_{0,0}, \ldots, a_{m-1,n-1}; r_A)$	$\xrightarrow{A,C,D}$	
$C := Com_{ck}(\{a_{j,i}(1 - 2b_{j,i})\}_{j,i=0}^{m-1,n-1}; r_C)$		Accept if and only if
$D := Com_{ck}(-a_{0,0}^2, \ldots, -a_{m-1,n-1}^2; r_D)$	$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$	$A, B, C, D \in \mathbb{G}$
		$f_{0,1}, \ldots, f_{m-1,n-1}, z_A, z_C \in \mathbb{Z}_q$
$\forall j, i : f_{j,i} := b_{j,i}x + a_{j,i}$		$\forall j : f_{j,0} := x - \sum_{i=1}^{n-1} f_{j,i}$
$z_A := rx + r_A$	$\xrightarrow{f_{0,1}, f_{1,1}, \ldots, f_{m-1,n-1}, z_A, z_C}$	$B^x A = Com_{ck}(f_{0,0}, \ldots, f_{m-1,n-1}; z_A)$
$z_C := r_C x + r_D$		$C^x D = Com_{ck}(\{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}; z_C)$

Another Σ-protocol described in the paper can be used to prove that a list of homomorphic commitments $\{C_0, ..., C_{N-1}\}$ includes a commitment to 0. More formally, the protocol can be used to prove relation R_2 shown in equation 2.4:

$$\mathcal{R}_2 = \left\{ ((\{c_i\}_{i=0}^{N-1}), (l, r)) \mid (\forall i, c_i \in \mathbb{C}_{ck}) \wedge (l \in \{0, \ldots, N-1\}) \wedge (c_l = Com_{ck}(0; r)) \right\} \quad (2.4)$$

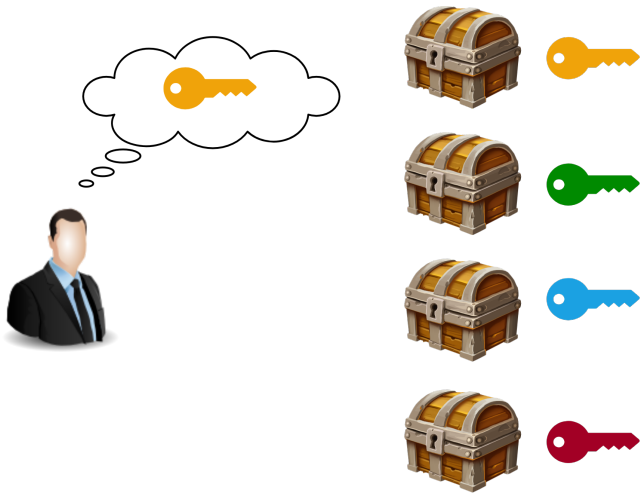


Figure 8: One-out-of-Many proof visualisation.

The main idea of this Σ-protocol is to prove knowledge of an index ℓ for which the product $\prod_{i=0}^{N-1} c_i^{\delta_{l,i}}$ is a commitment to 0, where δ is the Kronecker delta function, evaluating to 1 if $i = j$ and 0 otherwise.

A summary of the Σ-protocol based on [30] is now given:

1. *Prover* – Commit to m sequences of n bits $(\delta_{\ell,0}, \ldots, \delta_{\ell,n-1})$ and then run the Protocol 1 to prove the commitment is well formed.
2. *Verifier* – Verify the bit commitment and return challenge x.
3. *Prover* – Disclose elements $f_{j,i} = \delta_{l_j,i}x + a_{j,i}$ (see Protocol 1). Note that for every $i \in \{0, \ldots, N-1\}$, the product $\prod_{j=0}^{m-1} f_{j_{ij}}$ is the evaluation at x of polynomial $p_i(x) = \prod_{j=0}^{m-1}(\delta_{\ell,i}x + a_{j_i})$. For $0 \leq i \leq N-1$, the resulting polynomial is shown in the following equation 2.5:

$$p_i(x) = \prod_{j=0}^{m-1} \delta_{g_{l_j},g_{i_j}} x + \sum_{k=0}^{m-1} p_{i,k}x^k = \delta_{g_{l_j},i}x^m + \sum_{k=0}^{m-1} p_{i,k}x^k \tag{2.5}$$

Keep in mind that $p_{i,k}$ is dependent on ℓ and $a_{j,i}$, can be computed by the prover independently of the x chosen by the verifier, and most importantly, that $p_\ell(x)$ is the only degree m polynomial amongst $p_0(x), \ldots, p_{N-1}(x)$. From these coefficients and random noise values p_k, the prover can then compute the ciphertexts $G_k = \prod_{i=0}^{N-1} c_i^{p_{i,k}} \cdot Com_{gk}(0; p_k)$ and include them in the transcript. Such ciphertexts are then used to cancel out the low degree terms in the polynomial shown in equation 2.5. Namely, if c_ℓ is a commitment to 0, the following product will also be a commitment to 0 for any x, as shown in equation 2.6. The complete scheme is shown in the below Protocol 2.

$$\prod_{i=0}^{N-1} c_i^{\prod_{j=0}^{m-1} f_{j,i_j}} \cdot \prod_{k=0}^{m-1} G_k^{-x^k} = \left(\prod_{i=0}^{N-1} c_i^{\delta_\ell,i}\right)^{x^m} \tag{2.6}$$

Protocol 2: Σ-protocol for relation R_2. Adapted from [30].

$\mathcal{P}_2(ck, (c_0, \ldots, c_{N-1}), (l, r))$		$\mathcal{V}_2(ck, (c_0, \ldots, c_{N-1}))$
$r_B, \rho_k \leftarrow \mathbb{Z}_q$		
$B := Com_{ck}(\delta_{g_{l_{0,0}}}, \ldots, \delta_{g_{l_{m-1,n-1}}}, r_B)$		
$(A, C, D) \leftarrow \mathcal{P}_1(ck, B, (\{\delta_{g_{l_j},i}\}_{j,i=0}^{m-1,n-1}; r_B))$		
	$\xrightarrow{A,B,C,D,\{G_k\}_{k=0}^{m-1}}$	
For $k = 0, \ldots, m-1$		
$G_k = \prod_{i=0}^{m-1} c_i^{p_{i,k}} \cdot Com_{ck}(0; \rho_k)$		**Accept if and only if**
using $p_{i,k}$ from (2.5)		$f_{0,1}, \ldots, f_{m-1,n-1}, z_A, z_C, z \in \mathbb{Z}_q$
	$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$	$\mathcal{V}_1(ck, B, x, A, B, C, \{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z_A, z_C) = 1$
$(f_{0,1}, \ldots, f_{m-1,n-1}, z_A, z_C) \leftarrow \mathcal{P}_1(x)$		$\forall j : f_{j,0} := x - \sum_{i=1}^{n-1} f_{j,i}$
$z := rx^m - \sum_{k=0}^{m-1} \rho_k x^k$	$\xrightarrow{f_{0,1}, f_{1,1}, \ldots, f_{m-1,n-1}, z_A, z_C, z}$	$\prod_{i=0}^{N-1} c_i^{\prod_{j=1}^{m} f_{j,g_{i_j}}} \cdot \prod_{k=0}^{m-1} G_k^{-x^k} = Com_{ck}(0; z)$

CHAPTER 3

Literature Review

This chapter will provide a summary of the literature reviewed for this dissertation. Firstly, blockchain technology will be introduced, followed by an overview of the history of privacy in cryptocurrencies. A discussion will then be provided on the techniques are currently being leveraged to de-anonymise users of a cryptocurrency. Finally, the chapter will conclude with a discussion of the main solutions that have been proposed to address such challenges.

3.1 Blockchain and Cryptocurrencies

In his 2008 paper [10], Satoshi Nakamoto solved the double-spending problem by introducing the blockchain: the data structure that today is at the foundation of all cryptocurrencies. The following section will cover how a blockchain works at a high-level, as well as how it is used to make cryptocurrencies secure.

3.1.1 Blockchain Technology

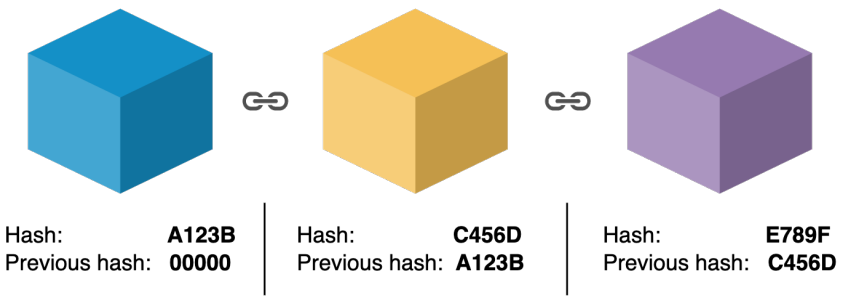


Figure 9: The blockchain data structure visualised.

At a high-level, a blockchain "block" can be split into 3 components:

1. **Data.** As the blockchain is an immutable database at its core, it can be used to store any type of information. For cryptocurrencies, the data to be stored is in fact all the transactions that have been validated by the network.
2. **Hash fingerprint.** As hash functions are inherently good at indexing and ensuring data integrity, a hash function is used to create a digest of the block's data, which is then used as the header of the block. This "fingerprint" will be used to identify the block within the blockchain.
3. **Previous block hash.** By including the hash previous block in the hash fingerprint, these blocks become linked: if one block is modified, its hash will change and not match the *previous hash* field of the next block. If even a single link between blocks mismatches, the whole chain is considered invalid: this is the property that makes the blockchain immutable.

3.1.2 Private vs Public Blockchains

The terms *blockchain* and *cryptocurrency* are often seemingly used interchangeably. While it is true that a cryptocurrency is a blockchain, not all blockchains are necessarily cryptocurrencies.

For instance, in a *permissioned* setting – where the network is centralised and controlled by a single entity or organisation – the blockchain can simply serve as a data structure storing immutable information (i.e. a ledger). Many such solutions (often referred to as "private blockchain") have been developed, the most famous of which are the Hyperledger project by the Linux foundation [31] and the Corda blockchain by R3 [32].

A *permissionless* blockchain however – where anyone is allowed to join the network and propose new blocks – introduces many new problems regarding securing it in a trustless, distributed environment. Some newfound problems that have to be addressed are:

- Choosing which block gets selected to be appended to the blockchain next
- Ensuring that every participant stores the same copy of the blockchain
- Incentivising people to validate transactions and not cheat the system

3.1.3 Cryptocurrencies

Indeed the latter problem provides an explanation as to why public blockchains need a cryptocurrency backing them: there needs to be a monetary incentive for participants to secure the network.

In Bitcoin, security and consensus are both achieved through the use of HashCash's Proof-of-Work algorithm [7], where the PoW problem is formulated so that on average, a new block is mined every 10 minutes. As the system rewards the miner solving the puzzle first, the network as a result is incentivised to keep mining new blocks. In this digital lottery, miners with greater computing power will have a higher chance of winning and earning the reward (i.e. minting new currency).

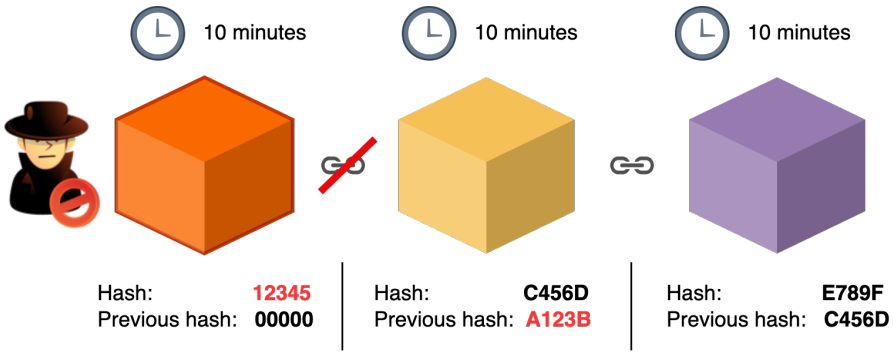


Figure 10: An example of attacking the immutability of a blockchain.

To understand how Proof-of-Work makes a blockchain secure, refer to Figure 10, where a malicious actor wants to modify the data of an old block in the blockchain to re-take ownership of some old coins (i.e. steal). As this operation would change the block's data, its *fingerprint hash* would also be updated, essentially starting a chain reaction causing all subsequent blocks to become invalid, due to the next block's *previous block hash* field not matching. At this stage, the attacker would find himself with a shorter chain than the original copy, where the head of his modified blockchain is represented by his fraudulent block.

Given that the "true" Bitcoin blockchain is defined as the longest valid chain of blocks that has been accepted by the network, the attacker would need to regenerate all the invalidated blocks (and some more) to convince other nodes to accept his chain as the valid one. Considering the attacker is competing against the whole the network, he would need to have more than 50% of the computing power on its own to be able to generate blocks at a faster average rate than everyone else combined. Since it takes real electricity to mine blocks, this is a very costly attack to perform on a large blockchain network, to the point of being economically unfeasible; thus rendering the blockchain secure.

Other cryptocurrencies have however adopted different strategies (consensus algorithms) to secure their networks, the most popular of which is the Proof-of-Stake algorithm. While such solutions can be more energy efficient than PoW, they often sacrifice security and decentralisation in favour of performance, and therefore only ultimately offer a trade-off.

3.1.4 UTXO vs Account model

Cryptocurrencies can pick between two models for representing and tracking their transactions on the blockchain: the *UTXO* (unspent transaction output) model or the *account* model.

UTXO model

The UTXO model, as the name suggests, involves using the outputs of previous transactions as inputs for later transactions. In essence, during a transaction, a user has to provide some UTXOs (i.e. coins) in his possession, which once used are considered spent and removed from the UTXO set. The values of these inputs are then transferred to the outputs provided in the transaction, which become part of the UTXO set instead. Note that as every input is considered spent when used, it is not possible to only use the partial contents of an UTXO in a transaction.

As such, when a user wants to give himself back some change, they must create a new UTXO of which they have possession of (see Figure 11).

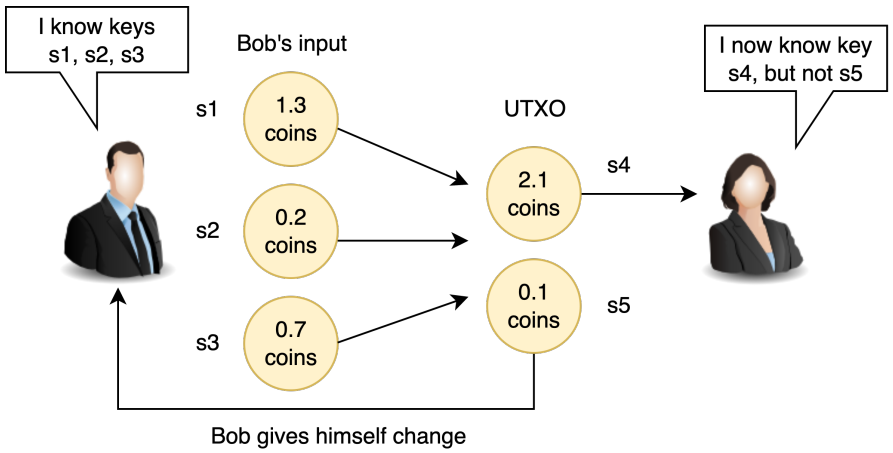


Figure 11: A transaction where Bob wants to pay 2.1 coins to Alice in the UTXO model.

As it is very hard to track all the outputs that belong to a single person, the UTXO model is considered ideal for achieving privacy in a blockchain setting. Indeed, although a user's available amount is represented by all the UTXOs he has the private key to, during a transaction a sender only has to prove ownership of the UTXOs he wants to spend. Popular examples of cryptocurrencies using the UTXO model include Bitcoin [10] and Litecoin [33].

Account model

The account model works similarly to a bank account: each user has an account with a balance, and this balance changes when money is either received or sent. Compared to the UTXO model, this model is more flexible and easier to work with, but offers less privacy due to linking all the transactions a user makes to their "account" (i.e. public-key). Popular examples of cryptocurrencies that use the account model include Ethereum [34] and EOS [35].

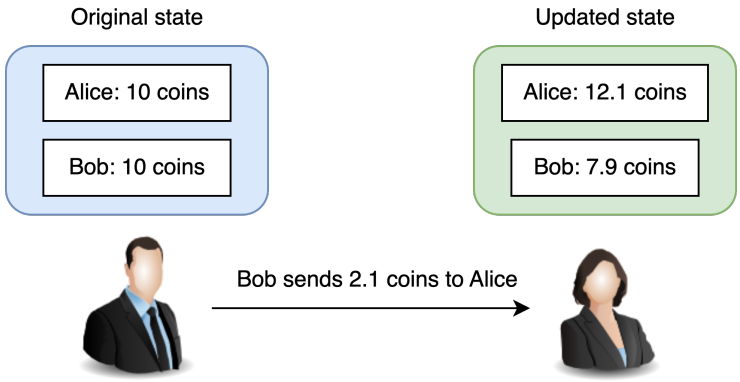


Figure 12: An example of the same transaction in Figure 11 using the account model.

3.2 Privacy in Blockchain

Having discussed what a blockchain is and how it works, the following sections will cover the key role that privacy plays in cryptocurrencies.

3.2.1 The importance of privacy in a cryptocurrency

Anonymity has been cited as a key property of a digital cash system decades before the advent of cryptocurrencies. For instance, in his 1995 paper "Digital Cash and Monetary Freedom" [36], Jon Matonis expands upon the work of Tatsuaki Okamoto and Kazuo Ohta [37] to propose ten properties of an ideal digital cash system achieving "monetary freedom":

1. **Secure** (unable to alter or reproduce)
2. **Anonymous** (untraceable)
3. **Portable** (physical independence)
4. **Infinite duration** (until destroyed)
5. **Two-way** (unrestricted)
6. **Off-line usability** (no network required)
7. **Divisible** (fungible)
8. **Wide acceptability** (universal)
9. **User-friendly** (simple)
10. **Unit-of-value freedom** (non-political)

When Satoshi Nakamoto introduced his peer-to-peer digital cash system over blockchain technology, the first implementation to satisfy all the above aspects of monetary freedom was thought to be achieved. However, since the development of blockchain analysis, Bitcoin no longer matches the definition of a digital cash system achieving monetary freedom, due to its compromised *anonymity* – and consequently *fungibility* – properties. Despite this, studies have shown that the majority of Bitcoin users show weak concerns regarding anonymity [38].

Privacy coins

The first privacy coin was introduced in 2014 by the community in order to address the growing anonymity concerns regarding Bitcoin. Today, many privacy coins are available to the general public, the most popular solutions being Monero [1], Zcash [2], and Dash [39]. Despite each project having its own motivation to achieve privacy [40], they all share the same philosophy of privacy as a human right.

In spite of best intentions, the privacy aspect of a cryptocurrency naturally attracts criminals and law enforcement. It's undeniable that many people use cryptocurrencies as a tool for breaking the law – an example of which is that ransomware payments in 2021 climbed to over $600 millions [41]. The advantages of privacy in cryptocurrency can however outweigh the risks. In a world where 85% of people are concerned that their online data may not be secure [42], it is of the utmost importance to safeguard the privacy of the users, especially regarding crucial information like financial transactions.

Legislation on privacy coins

In 2022, privacy coins have fallen under great scrutiny by governments and regulators worldwide citing tax fraud and money laundering. One of the most notable examples is the sanctioning of cryptocurrency tumbler Tornado Cash and its founder, who was charged with money laundering and operating an unlicensed money transmitter [43]. Furthermore, privacy coins are getting delisted from several exchanges [44][45][46], including coins offering optional privacy and claiming to be compliant with money laundering regulations across the world, such as Zcash [47].

This crackdown coupled with the speculative nature of cryptocurrencies has caused a decrease in price, as well as popularity, of privacy coins. As a result, development of privacy technologies has slowed down, as developers are either scared of the regulatory backlash or do not see a financial incentive to continue working on a project.

Privacy as a guarantor of fungibility

Fungibility serves as the strongest example of how privacy helps cryptocurrencies better fit their "medium of exchange" role.

A fungible good is defined as interchangeable with another good of the same type. An example of this is the cash of fiat money: it is irrelevant which series of banknotes one uses, as long as the denomination is of the same value.
The same example could be applied to Bitcoin, where one coin should be worth another single coin. A problem however lies in Bitcoin's public history, due to which "clean" Bitcoins tend to be of more value than "tainted" Bitcoins. An example of the discrimination different Bitcoins can face are the recent instances of exchanges freezing unrelated accounts that receive Bitcoins having a history of illicit activities [48][49].

In essence, the problem is that if specific Bitcoins can be blacklisted or arbitrarily deemed "not Bitcoin", censorship attacks become a possibility, which could lead to arbitrary seizures of funds. Such actions might appear reasonable and lawful, but consider the above case where an innocent recipient is sent "tainted" Bitcoins. When the recipient will try to spend his money, the transaction will get blocked, and the user could possibly be wrongly associated with a crime, or worse, face criminal charges. Such cases are becoming increasingly common [50] and a cause for significant concern.

In conclusion, as long as the entire history of every Bitcoin transaction is publicly available to be monitored and traced, the cryptocurrency cannot be considered fungible. To be able to compete with fiat money's fungibility property, it is necessary to ensure both *unlinkability* and *untraceability* for every transaction.

3.2.2 Transaction Graph Analysis

Having established the vital role privacy plays in a digital currency, it is also important to understand the different types of privacy attacks that can undermine it. The following section will therefore describe de-anonymisation techniques at the blockchain level (network-level attacks will not be considered).

To identify the real owner of a cryptocurrency address, blockchain analysis firms use a variety of techniques, most of which rely on *transaction graph analysis*. A transaction graph is a method of visualising the flow of funds on a blockchain network by representing addresses and transactions as nodes and edges in a directed graph. An example of a transaction graph is shown on the left of Figure 13. From such a graph, address clustering can be performed through the use of heuristic to generate an address cluster graph as shown on the right Figure 13.

This type of analysis is extremely simple for account-based cryptocurrencies: as there is only one address per account, the transaction graph is the same as the address graph. To generate a transaction graph for a UTXO-based cryptocurrency however, different techniques are usually employed. To defend against transaction graph analysis, it is first important to understand the heuristics that are used to create the address graph. For instance, for Figure 13, heuristics which have been proven to be effective in literature [51][52] – and that are described in the following page – have been used to derive the address graph:

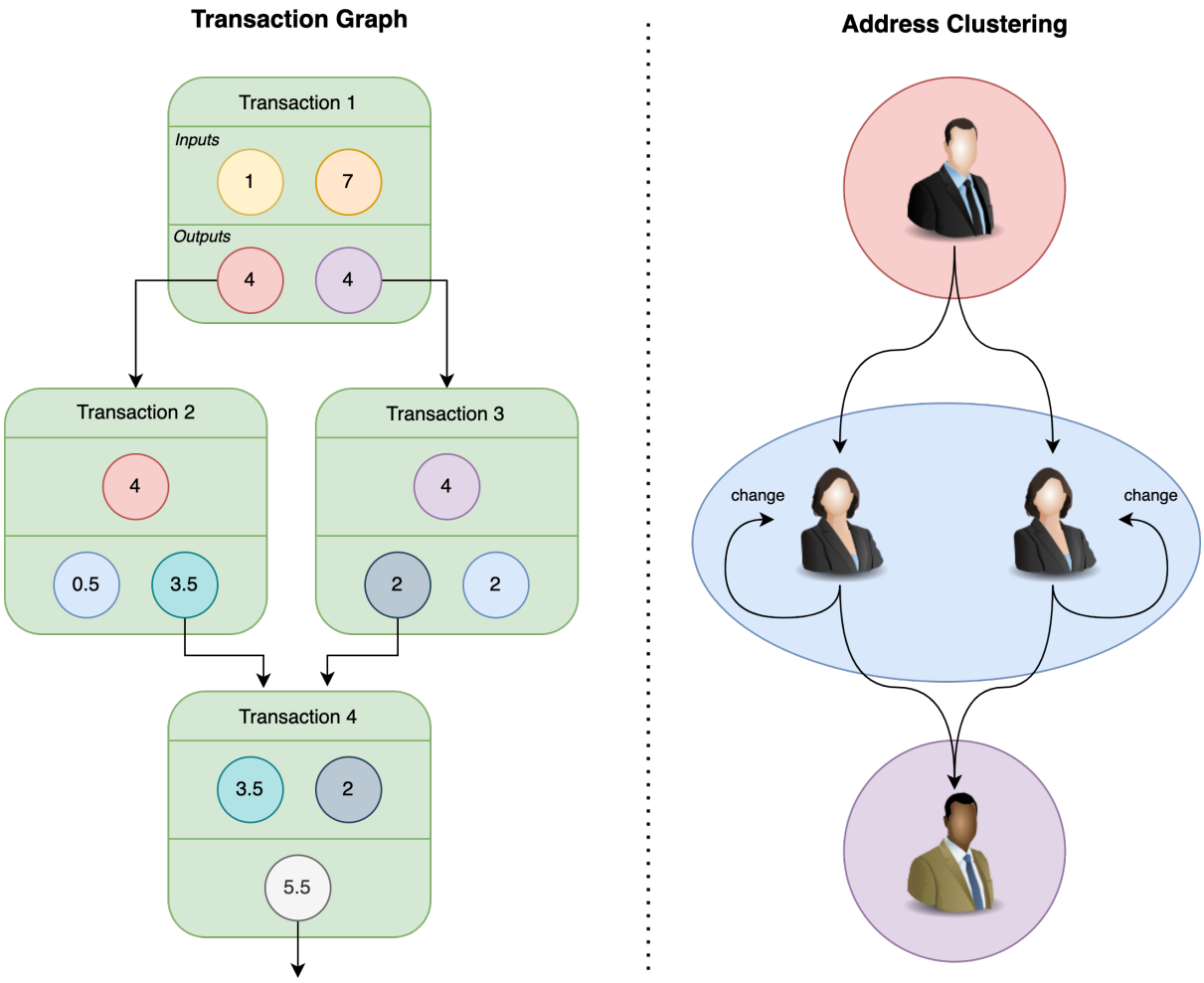


Figure 13: An example of a transaction graph on the left and an address graph on the right. Colors represents the address the UTXOs are sent to.

Heuristic 1 – Multi-source-same-owner. As Satoshi Nakamoto stated in his Bitcoin paper [10]: "Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner". To this day this is the most basic and powerful heuristic, which allows for a strong link between addresses used in separate transactions to be formed.
Take *Transaction 4* in Figure 13. As it spends UTXOs generated from the two separate *Transaction 2* and *Transaction 3*, it follows that those addresses belong to the same entity. It is also possible to derive that a user (Bob) sent multiple transactions to the same recipient (Alice), although it is not possible to link the addresses to a real life identity yet. As this heuristic relies on the fact that the input addresses are linked to the output addresses, it can be contained by implementing the *untraceability* property.

Heuristic 2 – One-time change-address Given the nature of the UTXO model, it is extremely common for a change address to be present in a transaction. Notice how, if it was possible to always identify the change address, the send addresses could be clustered with it as being owned by the same entity.
A simple heuristic used to identify change addresses is looking for a small amount sent to a never-used address. For instance, in *Transaction 2*, the relatively small amount of 0.5 (along with other data) could be used to identify the change address in the transaction and therefore group it with the inputs. As this heuristic relies on the amounts being transacted, implementing *confidential transactions* severely mitigates it.

Heuristic 3 – Change-address reuse. Some users reuse the same change address for multiple transactions. This allows for instance for the the second output of *Transaction 3* to be identified as a change output, as the same address has been used as a change output in *Transaction 2*. This heuristic could be contained by implementing *unlinkability*, which would prevent the reuse of addresses.

While these heuristics are the most common, many clustering techniques and pattern recognition algorithms have been proposed in the literature [53][54][55][56]. Once addresses have been clustered, blockchain analysis firms can use a variety of techniques to associate addresses with real-world entities. For instance, in [51], Meiklejohn *et al.* note that collusion with a KYC (Know Your Customer) exchange can be used to trivially associate clusters to real-world entities. In their paper [57], Michael Fleder *et al.* also show how scraping bitcoin addresses from public forums can be used to associate addresses with a person's true name or username. In conclusion, if an address graph has been formed and an address contained within it is leaked, the entire transaction history of a user can be revealed.

Nonetheless, as blockchain analysis firms are private companies, it is impossible to determine the exact methods employed to de-anonymise users. Despite this, the abundance of techniques that can be used to build a transaction graph highlights how crucial it is for a cryptocurrency to implement *complete privacy*, as any piece of information can potentially be utilised to de-anonymise its users.

3.2.3 Privacy-preserving Techniques

Following a discussion of the possible attacks on blockchain anonymity, this section will focus on the solutions born to defend and achieve privacy in practice.

Tumblers and CoinJoin

The CoinJoin [58] protocol, as described by Dr. Maxwell, is a technique that allows multiple users to combine multiple cryptocurrency payments into a single transaction, thus making it more difficult to determine which input was used to pay which recipient. Note that within a CoinJoin transaction, all inputs used in a must be of the same value, to ensure that the transaction output amounts are not easily identifiable as belonging to any particular participant.

While early implementations used a centralised server called a "tumbler" to coordinate the mixing of coins, delegating the mixing of coins to a third party has several drawbacks. Firstly, as the tumbler acts as a custodian, there is no guarantee that it will not steal the funds of users. Secondly, despite payments being obfuscated in the blockchain, the tumbler owner retains knowledge of the pre-mixed inputs used and can easily collude or sell this information to a third party. Notably, later iterations of tumblers such as Tornado Cash [43], fixed these issues by offering non-custodial solutions though the use of smart contracts.

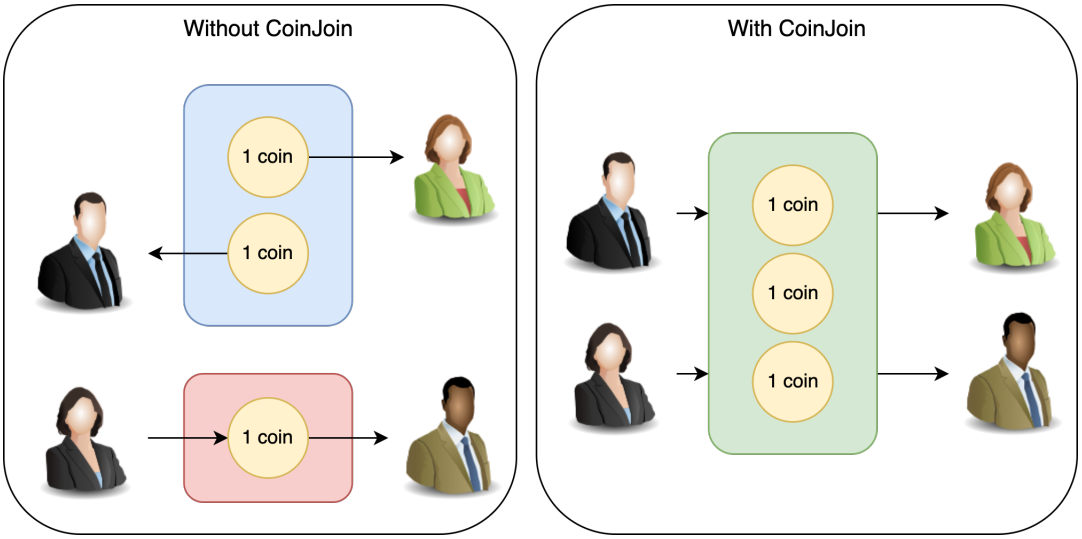


Figure 14: Comparison between a normal transaction and a CoinJoin transaction.

The first decentralised coin mixing protocol, Coinshuffle [59], was created by Ruffing *et al.* and allowed for the first trustless implementation of CoinJoin. This protocol was later improved upon by Coinshuffle++ [60], which introduced a new method of mixing coins that is more efficient and secure than the original protocol by using a mixnet to hide the identity of the participants. The protocol is currently implemented in the cryptocurrency Decred [61].

Although CoinJoin is a useful technique for enhancing privacy in cryptocurrency transactions, it has certain limitations. For instance, CoinJoin's interactive nature and need for participant coordination impose significant constraints on the number of mixes that can occur in a given time. Furthermore, the degree of anonymity that CoinJoin can provide is limited by the number of participants involved the mix. Dash's "PrivateSend" implementation [39], for example, usually only involves 3 participants, which is not enough to provide strong anonymity.

A notable advantage to CoinJoin however is that it can be easily implemented on top of any UTXO-based cryptocurrency. Such versatility allows for otherwise pseudo-private cryptocurrencies to benefit from the privacy features provided by CoinJoin. For instance, the Wasabi [62] Bitcoin wallet allows for built-in CoinJoin transactions via their WabiSabi [62] protocol.

Overall, whilst tumblers and CoinJoin are a good starting point, they only offer *untraceability* and cannot therefore be considered a complete privacy solution. Furthermore, fungibility can not be ensured, as even CoinJoin transactions are starting to be treated as "suspicious" [63][64].

CryptoNote, Ring Signatures and RingCT

In 2012, pseudonym "Nicolas van Saberhagen" released the CryptoNote whitepaper [14], which introduced a design for a private cryptocurrency using *ring signatures* and one-time addresses (today known as *stealth addresses*).

Ring signatures, firstly described by Rivest *et al.* [65], are a type of digital signature that allows a user to sign a message so that a third part can verify that the message was signed by a particular user in a group, without revealing which person in the group signed it. A transaction implementing ring signatures automatically selects several similar inputs from the blockchain, rendering it impossible to determine which one is being spent by the transaction. It also allows plausible deniability, as someone's input could be used as a decoy in another transaction. As ring signatures obfuscate which inputs are being spent in a transaction, they provide *untraceability*.

A stealth address is a one-time, unique public address that is generated for every transaction in order to receive the funds. This means that the receiver's actual public address is never revealed on the public blockchain, making it more difficult to link transactions to specific individuals or addresses. As it protects the recipient's identity, this features provides *unlinkability*.

The first implementation of the CryptoNote protocol, Monero [1], was first released in 2014 and is to this date the most popular privacy cryptocurrency solution. Monero pioneered implementing private-by-default features in cryptocurrencies, but over time, such features have faced challenges from increasingly sophisticated adversaries. For instance, as initial implementations of Monero did not include *confidential transactions*, the privacy of many users was compromised due to transactions not being able to find a suitable decoy with the same amount in the set of available inputs. As a result, studies had shown that 65% of transactions were performed with a ring of 1 (offering no privacy protection), and 95% of the true inputs could be detected by further leveraging the *multi-source-same-owner* heuristic, by noticing that different decoy inputs in a transactions were created in a same previous transaction [66].

Eventually, Monero implemented the *RingCT* [67] protocol in 2017, which introduced *confidential transactions* and allowed for decoy sources to be picked from the blockchain irrespective of the value of the inputs. Despite the improvements, the decoy selection algorithm used in RingCT has proven to be challenging to implement in a privacy-preserving manner. Security researchers have on many occasions used statistical analysis and input age to determine the true input used in a transaction. For example, a 2017 study [68] found that 90% of transactions could be linked to their true input by guessing the most recently used input, due to how the decoy selection algorithm worked. In 2021 another error in the decoy selection algorithm caused all users that spent their inputs too quickly after receiving them to be exposed [69].

Monero developers have since changed the decoy selection algorithm and increased the number of decoys from 10 to 16. While statistical analysis could be severely hindered by implementing even larger ring sizes, the size of transactions grows linearly with the number of decoys used. In [70], it is also argued that increasing ring sizes will not fully eliminate statistical vulnerabilities: "In order for the camouflage to conceal user privacy, merely having lots camouflage is not enough. We must also ensure that the camouflage is placed in the right locations".

Creating a perfect decoy selection algorithm is still an active area of research and still represents the biggest weakness of RingCT-style privacy. Despite these drawbacks, Monero and RingCT have proven to be one of the better and more well-reviewed privacy-preserving technologies in the crypto-space and are still evolving in order to provide better privacy against new threats.

QuisQuis

QuisQuis is a novel design for anonymous cryptocurrencies that was introduced by Fauzi *et al.* in 2019. No current cryprocurrency implements QuisQuis, but it is worth mentioning due its novel use of *updatable public keys* to try and solve the problem of monotonically increasing UTXO sets in Monero [1] and Zcash [2].

A QuisQuis transaction, similarly to RingCT, uses ring signatures with a set of decoy inputs; however, there are a few key differences. Firstly, the *account model* is used with an Elgamal commitment encrypting the public key and the amount stored in the account. Secondly, every input account's public key P is updated by the sender to P' — where P and P' are computationally indistinguishable (i.e. *unlinkable*) — and the account values of the sender and receiver are updated accordingly. Two zero-knowledge proofs are also used to verify that all accounts were updated correctly, and that the sender knows the *private key* of one of the accounts. An example of a QuisQuis transaction is shown in Figure 15.

Recall that in Monero, due to the inability to determine with certainty when an input has been spent, it is not possible to remove inputs from the ever-growing UTXO set. In QuisQuis however, since every transaction updates the public key of all the inputs, the old inputs can be replaced by the corresponding outputs in the UTXO set. Moreover, since a public key appears only twice in the blockchain (at its creation and when spent), several privacy attacks that were feasible in Monero — and which relied on addresses being reused as decoys several times — are no longer possible.

Allowing users to update other's private keys can however have some unintended consequences. For instance, two users trying to use the same decoy at once (i.e. update the same key), causes a race condition where one transaction will be accepted and the other will be rejected. Furthermore, it is possible for an attacker to repeatedly update a user's public key, until to the user loses track of those keys; no solution to this problem has been proposed in the protocol. Finally, a bug in the implementation of the zero-knowledge proof could also trivially allow a user to steal funds from other accounts. Nevertheless, QuisQuis offers a *complete privacy* solution and is the first protocol offering a solution to the monotonically increasing UTXO sets in *untraceable* cryptocurrencies.

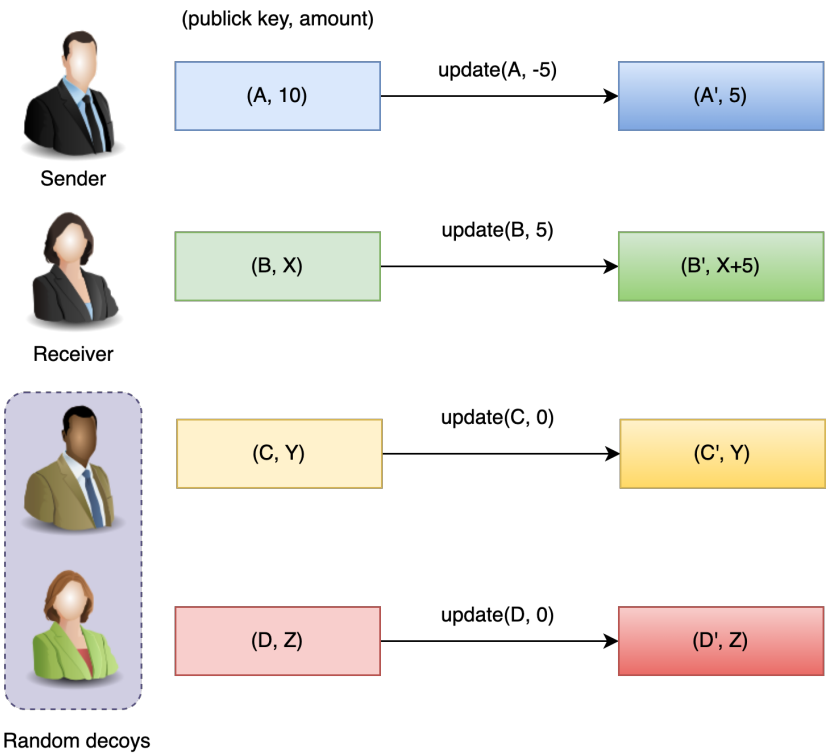


Figure 15: An example of a QuisQuis transaction.

Mimblewimble

Anonymously posted by pseudonym "Tom Elvis Jedusor" (the French name for Voldemort) to a Bitcoin IRC channel in 2016, *Mimblewimble* (named after a Harry Potter spell tying the target's tongue) was later fully formalised by Andrew Poelstra [71] in the same year. Mimblewimble is a unique cryptocurrency protocol which removes the need for addresses by introducing an authentication schemes on top of *confidential transactions*. A Mimblewimble transaction requires interaction between a sender and a receiver in order to create a Schnorr signature proving that the sender knows the blinding factors used and that the transaction does not create or destroy any coins. The resulting outputs' blinding factors will only be known to the corresponding parties, therefore acting as *private keys* for future transactions. Mimblewimble also supports non-interactive CoinJoin by default, as well as pruning of intermediate outputs, which can be used to attain a very small blockchain.

Implemented by cryptocurrencies such as Grin [72] and Beam [73], Mimblewimble was initially thought to offer privacy on par with Monero and Zcash. Although the Mimblewimble protocol ensures *untraceability* by performing CoinJoin at the blockchain level, it was theorised that an attacker at the network level could still potentially construct a transaction graph by monitoring individual transactions as they are broadcasted, before they are mixed with other transactions in a block.

To address this vulnerability, cryptocurrencies implementing Mimblewimble attempt to use the Dandelion++ protocol [74] to aggregate multiple transactions before they are broadcasted to the entire network. The protocol operates in two phases, beginning with the stem phase where a transaction is initially sent to random peers. From there, each consecutive peer has a 10%

chance of advancing to the "fluff phase", in which the transaction is propagated to other peers. Figure 16 provides an illustration of this process. By implementing the Dandelion protocol, the hope is that nodes waiting in the "stem phase" will receive multiple transactions that can be aggregated together before revealing them to the network.

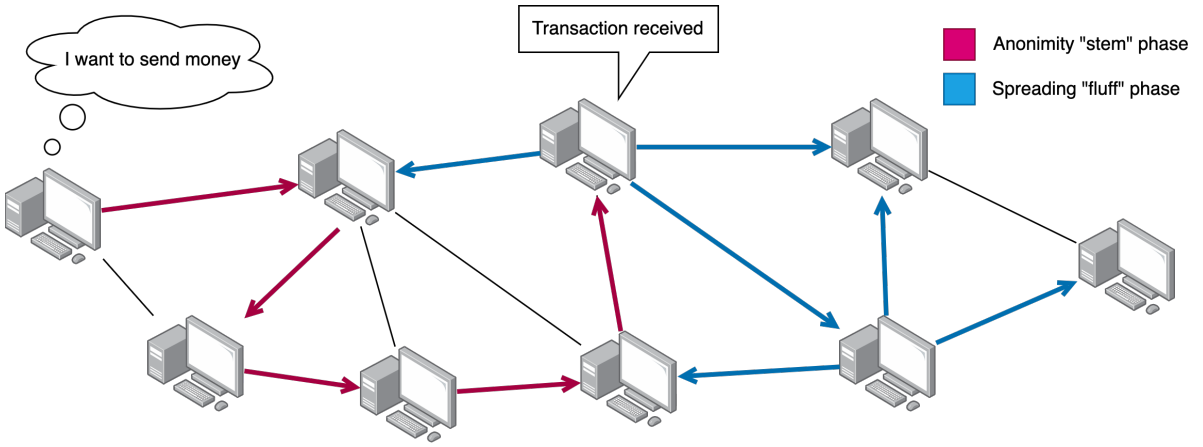


Figure 16: A visualisation of the Dandelion++ protocol.

While in theory this should make it hard to learn about how transactions were aggregated, a 2019 study [3] showed that despite the use of Dandelion, a node connected to lots of peers could still learn about 96% of transactions before they were aggregated, building a transaction graph in the process. The Mimblewimble protocol is therefore lacking in the most desirable property of *untraceability*, which constitutes the main defense against deanonymisation attacks. As a result, the privacy offering of Mimblewimble-based cryptocurrencies is significantly weaker than other solutions.

Lelantus

Lelantus, as implemented by Firo [75], is a privacy-preserving protocol based on the Zerocoin protocol [76] and the *zero-knowledge* One-out-of-Many proofs [4].

Zerocoin [76] is a proposed extension of the Bitcoin protocol which adds *untraceability* to selected transactions by allowing users to *mint* private "Zerocoins" in exchange for their public Bitcoins. The Zerocoins can then be *redeeemed* from a pool by way of a zero-knowledge proof involving a serial number, proving that the Zerocoins have not been spent before. When redeeemed in a transaction, a Zerocoin is turned back into the so-called *basecoin* (e.g. Bitcoin) without revealing which Zerocoin was spent during the process. Note that this burn-and-redeem scheme requires users to burn coins in fixed denominations, as otherwise the amount value would leak information about which coins were spent.

Lelantus improved upon the Zerocoin protocol by introducing *confidential transactions* for the source and change amounts, allowing for burning of arbitrary amounts. To achieve this, Lelantus' "shielded outputs" use the serial number as a second blinding factor in a double-blinded commitment scheme. To spend a private coin in the "shielded pool", a user must first reveal the serial number of the coin, as well as a signature revealing knowledge of its discrete logarithm to prove ownership. This serial number is then stored in a set to prevent future double-spends and

is used to generate a *One-out-of-Many proof* [4] to prove knowledge of one double-blinded commitment from the shielded pool (note that the original Jens Groth's proof had to be extended to support double-blinded commitments for this). An original, zero-knowledge, "Balance proof" is also required to prove that the extracted coins cancel out the spent coins, and that no new coins are created in the process.

While Lelantus still partly suffered from Zerocoin's limitations, such as lacking *full* confidential transactions (no support for receiver) and *unlinkability*, the protocol was later improved upon with Lelantus Spark [77], which added *stealth addresses* and support for direct anonymous payments not involving a conversion to a basecoin (which would leak information about the amount).

Currently, the protocol satisfies all properties of *complete privacy*, and its main limitation comes from the usage of the One-out-of-Many proof, due to which anonymity sets are limited to a size of around 64,000 before the proof generation and verification time becomes unacceptable in practice.

Zerocash

The leading privacy scheme is arguably the Zerocash protocol [2], as implemented by the Zcash [78] cryptocurrency. Similarly to Lelantus, Zerocash aims to improve upon the Zerocoin protocol by adding support for *confidential transactions* and *unlinkability*. To achieve such properties, Zerocash initially used the Groth16-zkSNARK scheme [79] and a *trusted setup* to support succint proof constructions.

Like Zerocoin, Zcash's blockchain supports a private coin though "shielded" addresses — which use zero-knowledge proofs to spend coins — as well as a public coin through standard "transparent" addresses. All coins kept in shielded addresses form the *shielded pool*, and since it is impossible to know which coins are spent from the shielded pool, the entire pool represents an ever-growing anonymity set. A further set is additionally needed to keep track of revealed serial numbers, to ensure that a coin in the shielded pool can not be spent twice.

The main drawback of Zerocash is its use of highly complex and experimental cryptography. Unlike other protocols, which make use of standard cryptographic assumptions such as the discrete logarithm or factorisation hardness problems, Zerocash relies on the much newer Knowledge of Exponent (KoE) assumption for bilinear groups [80]. This effectively makes the protocol complex to understand, audit and implement.

Zerocash has in fact suffered from a number of vulnerabilities arising from cryptographic flaws, including two bugs that could have generated unlimited amounts of coins [81][82]. While the bug described in [82] never made it to production, [81] remained undetected for two years before it was patched in the "Sapling" upgrade. As it is impossible to detect supply in a cryptocurrency implementing *confidential transactions*, to date it is not known if counterfeit coins were generated by the vulnerability. In their vulnerabilty disclosure, the Electronic Coin Company stated that: "Discovery of the vulnerability would have required a high level of technical and cryptographic sophistication that very few people possess. The vulnerability had existed for years but was undiscovered by numerous expert cryptographers, scientists, third-party auditors, and third-party engineering teams who initiated new projects based upon the Zcash code."

While another general critique of Zerocash was its necessary trusted setup ceremony, with the most recent "Orchard" upgrade [83] Zcash has moved to the Halo 2 system [84], a new PLONK [85] based circuit design that does not require a trusted setup. Despite this new construction sacrificing performance in order to remove the trusted setup, the new circuit remains extremely competitive. Currently, the latest audit of the Zerocash protocol has found no significant vulnerabilities, but they evidenced that "there is no overarching proof of security" and generally commented on the complexity of reviewing such a complex system.

Despite these limitations, the Zerocash protocol remains the most advanced and favourable privacy-scheme for cryptocurrency, offering complete privacy, the largest anonymity set and the fastest verification times. However, some time will be needed for the wider academic and technical community to fully understand and audit the protocol in its entirety.

Comparison

The previous sections has presented six different designs for achieving privacy in cryptocurrencies. At a high-level, these designs can be grouped into two models: *decoy-based* and *zero-knowledge* schemes. CoinJoin, RingCT, QuisQuis and Mimblewimble all fall under the decoy-based category. In these schemes, the anonymity set is generally limited to a small number of participants, due to decoy data being embedded within the transaction itself. Lelantus and Zerpcash on the other hand use the zero-knowledge model, where only a zero-knowledge proof – proving that the coins being spent are valid – is present in the transaction. In these schemes, the anonymity set can be arbitrarily large, and is usually either the entire shielded pool or a large subset of private coins. Table 3.1 summarises the main privacy properties offered by each protocol, as well as a comparison of their anonymity set sizes.

Protocol	**Unlinkability**	**Untraceability**	**Conf. Tx**	**Anonymity set size**
CoinJoin	×	✓	×	# particpants in tx
RingCT	✓	✓	✓	16
QuisQuis	✓	✓	✓	16
Mimblewimble	✓ (no addresses)	×	✓	# tx in block
Lelantus Spark	✓	✓	✓	Up to 64,000
ZeroCash	✓	✓	✓	# shielded outputs

Table 3.1: Comparison of the various cryptocurrency privacy-enabling schemes.

Overall, while it may seem logical to assume that a larger anonymity set size is always the ideal choice when selecting a privacy scheme, this is not necessarily the case. In fact, to this date, no protocol meets all the criteria of providing high performance, ease of implementation and auditability, as well as flexibility in supporting various use-cases (e.g. direct payments, atomic swaps and smart contracts). While zero-knowledge based models hold great promise for a future "perfect" protocol, the current regulatory oversight of private cryptocurrencies could potentially hinder further progress in research and development of privacy-enhancing technologies.

CHAPTER 4

Mimblewimble

While the Mimblewimble protocol was introduced briefly in section 3.2.3, this chapter will delve deeper into its workings, with a specific emphasis on the formation and aggregation of Mimblewimble transactions. This will provide a comprehensive understanding of the scheme, which is crucial for comprehending the novel protocol described in chapter 5.

4.1 Overview

The Mimblewimble protocol uses a group $\mathbb{G}$ of prime order p with two generators G and H. As with confidential transactions [86], a Mimblewimble output is represented by a Pedersen commitment in the form $C = v.G + r.H$, where v is the number of coins and r is a blinding factor used to prevent an attacker from deducing value v from list of pre-calculated values. Since the output is a commitment, the values v and r are not obfuscated to the public and include no information about the owner of the output (unlike public keys in other cryptocurrencies).

4.2 Mimblewimble Transactions

To provide an example of a Mimblewimble transaction, take *Alice* and *Bob*. Alice has 3 coins stored in unspent output $C_a = 3.G + r_a.H$ whom she wants to send to Bob. As Mimblewimble uses the UTXO model, she creates a transaction T with the following properties: T spends input C_a and creates a new output for Bob C_b. In this transaction, ownership of the coins should pass from Alice to Bob. This is achieved by letting Bob secretly choose a blinding factor r_b for his output, such that the final output results in $C_b = 3.G + r_b.H$.

At this point, Alice and Bob have created a transaction such as the one shown in Table 4.1.

Inputs	Outputs
$3.G + r_a.H$	$3.G + r_b.H$

Table 4.1: Alice sending 3 coins to Bob.

For the transaction to be valid, the following properties must be satisfied to achieve a valid Mimblewimble transaction:

1. **Non-inflationary.** Alice and Bob need to prove that the inputs' amounts minus the outputs' amounts equal to to zero, i.e. no new money was created from thin air
2. **Input Ownership.** To spend the input C_a, Alice has to prove ownership of v_a and blinding factor r_a (which serves as Alice's *private key*)

The following sections will explain how these properties are achieved in Mimblewimble.

Proving non-inflationary transaction

In most cryptocurrencies, the value of coins being spent is public, making it easy to verify the transaction is not creating money. For a transaction to be valid, the sum of inputs must always be equal to the sum of outputs, such that $output - input$ or $3coins - 3coins = 0$.
In Mimblewimble, the problem gets trickier as the value of coins is obfuscated. Thanks to the homomorphic property of Pedersen commitments however, it is still possible for verifiers to sum and subtract encrypted values without requiring knowledge of the values themselves.

For instance, a sender could prove to a external verifier that no new coins were created in a transaction by using an output C_{out} such that $C_{out} - C_{in} = 0$. Note that this entails C_{out} being equal to C_{in}, including the value v and blinding factor r. Assuming Bob wants to send his newfound 3 coins to Charli and further chose $r_b = 32$ when transacting with Alice, to naively prove that no new coins are being created, he could create a transaction using such C_{out}:

Inputs	Outputs
$3.G + 32.H$	$3.G + 32.H$

Table 4.2: Bob's naive transaction.

There is however a problem: since the blinding factor is known by both Bob and Charli, it can not act as *private key* for Charli; Bob would be able to steal the coins anytime he wants.
To solve this, Bob lets instead Charli choose a blinding factor of her own, like Alice did when transacting with him. Given Charli chooses $r_c = 43$, the final transaction would look as such:

$$C_{charli} - C_b = (3.G + 43.H) - (3.G + 32.H) = (3 - 3).G + (43 - 32).H = 0.G + 11.H \quad (4.1)$$

In this transaction, the transacted values cancel out ($0.G$), but an excess value $11.H$ remains, representing the difference in blinding factors. So how can nodes, who only see encrypted commitments, verify that the transaction's output and input values cancel out?

The solution requires both Bob and Charli to build a Schnorr signature together, using the difference between their private keys ($r_b - r_c$). Note that during this process, Bob and Charli do not learn of each other's blinding factors.
This signature achieves two critical properties: it proves the difference between inputs and outputs is in the form $x.H$ only (i.e. the values have canceled out) and it ensures that both Bob and Charli are in agreement and in possession of their corresponding private keys (i.e. their chosen blinding factors). The resulting transaction is shown in Table 4.3.

Inputs	**Outputs**	**Schnorr signature**
$C_b = 3.G + 32.H$	$C_{charli} = 3.G + 43.H$	σ for public key 43.H - 32.H = 11.H

Table 4.3: Updated Bob to Charli transaction.

With the information in 4.3, a node can now verify that:

- $C_{charli} - C_b$ is a valid public key for H. This can only be true if values v cancel out such as in $0.G + 11.H = 11.H$, where the remaining $11.H$ is the *public key.*
- The Schnorr signature is valid against the resulting transaction "excess" of $11.H$. This can only be true if both parties have signed with *private keys* (blinding factors) that subtract to 11.

Proving ownership

Charli is now in possession of $C_{charli} = 3.G + 43.H$, while Bob has spent his coins C_b. To better understand how knowing the blinding factor for an output proves ownership, let's assume Bob wants to steal back the 3 coins he just sent Charli.

For starters, Bob knows that C_{charli} contains $3.G$ coins, since he sent them in the first place. He however is not sure about the r_c that Charli picked (43), so he takes a guess and picks $r_c = 20$. He can now form a transaction sending the coins to an output of his own (with blinding factor $r = 4$) from his guessed input of C_{charli}:

True Input	**Bob's Input**	**Bob's Output**	**True excess**	**Bob's excess**
$3.G + 43.H$	$3.G + 20.H$	$3.G + 4.H$	$(43 - 4).H = 39.H$	$(20 - 4).H = 16.H$

Table 4.4: Data related to Bob's fraudulent transaction.

Since Bob is acting as both the sender and receiver in the transaction, he simply signs the calculated excess 16 and sends the transaction to the network. Keep in mind that even though Bob calculated the excess with a fraudulent input, when sending the fully formed transaction, he is required to point at an unspent output in the blockchain; he cannot simply provide an input of his own.

When a node receives the transaction, it will compute the blinding factor excess by subtracting Bob's output from input C_{charli}, resulting in $39.H$. Since Bob has signed for $16.H$, the node will consider the transaction invalid and reject it.

The above example illustrates how, in Mimblewimble, ownership of outputs is proven by possession of private key r for a particular unspent input: Bob will never be able to create a valid transaction as long as he doesn't know what value Charli picked for r_c. This is different from other cryptocurrencies, where coins are tied to addresses (i.e. *public keys*) and ownership is proven by providing a digital signature.

Change and Rangeproofs

Recall that in the UTXO model, it is not possible to partially spend an output. This means that if Bob wanted to send 2 coins to Charli instead, and he only owned the output containing the 3 coins sent to him by Alice, he must include an owned output with his remaining 1 coin change in the transaction.

The new transaction including Bob's change would look as follows:

Inputs	**Outputs**	**Schnorr signature**	**Range proofs**
$C_b = 3.G + 32.H$	$C_{charli} = 2.G + 43.H$	σ for public key	π_1
	$C_{change} = 1.G + 34.H$	(34+43).H - 32.H = 45.H	π_2

Table 4.5: Bob to Charli transaction with change.

To enable Charli to form her partial signature, Bob passes her the difference between his change and input excesses $(34 - 32).H = 2.H$. The resulting public excess can therefore be formed as $(2 + 43).H = 45.H$, without Charli ever learning about any of Bob's blinding factors.

Also note how the transaction in 4.4 includes two range proofs, π_1 and π_2, for each of the outputs. A range proof proves a value v is in range $[0, 2^{32}]$, which prevents buffer overflow attacks and negative values from entering transactions. The below Table 4.5 provides an example of how using negative values in the outputs can be used to create a valid, yet fraudulent, transaction.

Inputs	**Outputs**
$10.G + 1.H$	$10.G + 2.H$
	$-1000.G + 3.H$
	$1000.G + 4.H$

Table 4.6: A transaction without range proof creating 1000 coins out of thin air.

In the above transaction all the values cancel out to zero; a verifier would have no way of knowing that 1000 coins were created. This is why Mimblewimble requires range proofs for each output.

A full Mimblewimble transaction

The detailed Mimblewimble transaction protocol is provided in Protocol 3. In general, a transaction using n unspent inputs C_{in} and creating m new outputs C_{out} will consist of:

- A list of Pedersen commitments $(C_{in_1}, \ldots, C_{in_n})$, representing the inputs
- A list of Pedersen commitments $(C_{out_1}, \ldots, C_{out_m})$, representing the outputs
- A Schnorr signature proving that no money was created and that the coins are owned by the sender
- A list of of range proofs $(\pi_1, \ldots, \pi_n)$ verifying that the committed values are not negative

Protocol 3: Forming a Mimblewimble transaction

Sender(v_{in}, r_{in})		**Receiver**
$C_{in} = v_{in}.G + r_{in}.H$		
$r_s, k_s \leftarrow\$ \mathbb{Z}_p$		
$C_{change} = v_c.G + r_s.H$		
$v_r = v_{in} - v_c$		
$x = r_s - r_{in}$		
	$C_{in}, C_{change}, x.H, k_s.H, v_r \longrightarrow$	
		$r_r, k_r \leftarrow\$ \mathbb{Z}_p$
		$e = \mathscr{H}(k_s.H + k_r.H \| x.H + r_s.H)$
		$s_r = k_r + e * r_r$
		$C_r = v_r.G + r_r.H$
	$\longleftarrow C_r, s_r, k_r.H, r_r.H, \pi_r$	
$e = \mathscr{H}(k_s.H + k_r.H \| x.H + r_s.H)$		
$s_r.H \stackrel{?}{=} k_r.H + e * r_r.H$		
$s_s = k_s + e * r_s$		
$s = s_s + s_r$		
$k = k_s + k_r$		
		Node(UTXO)
	$C_{in}, C_{change}, C_r, (s, k.H), \pi_c, \pi_r \longrightarrow$	
		$C_{in} \stackrel{?}{\in}$ UTXO set
		$excess = (C_{change} + C_r) - C_{in}$
		$e = \mathscr{H}(k.H \| excess)$
		$s.H \stackrel{?}{=} k.H + e * excess$
		$\mathsf{Vf}(C_{change}, \pi_c) \stackrel{?}{=} 1$
		$\mathsf{Vf}(C_r, \pi_r) \stackrel{?}{=} 1$
		remove C_{in} from UTXO set
		add C_{change}, C_r to UTXO set

Transaction aggregation

Thanks to its lack of addresses and its unorthodox spend protocol, Mimblewimble can also leverage some additional scalability and privacy techniques, the most important of which is *transaction aggregation*, which involves treating multiple transactions as a single transaction. To build an intuition of how this is possible, recall that the excess of a transaction is simply the difference between output and input values:

$$\text{excess} = \sum_{i=1}^{m} C_{out_i} - \sum_{j=1}^{n} C_{in_j} \tag{4.2}$$

Furthermore, a block containing multiple transactions can be simply considered as an extended set of inputs and outputs. To validate the block, an excess can still be created by calculating the difference between all the outputs and all the inputs, as Equation 4.3 shows. The resulting value would be a commitment equal to the homomorphic sum of the excesses of every transaction in the block:

$$\sum_{h=1}^{l} excess_h = \sum_{i=1}^{n_{all}} C_{out_i} - \sum_{j=1}^{m_{all}} C_{in_j} \tag{4.3}$$

This property be easily shown by using two Mimblewimble transaction, T_1 and T_2, as shown in the below Table 4.7:

Inputs	**Outputs**	**Signatures**
$20.G + 45.H$	$11.G + 13.H$	σ for $17.H$
$10.G + 68.H$	$12G + 26.H$	
	$7.G + 91.H$	

Inputs	**Outputs**	**Signatures**
$13.G + 56.H$	$10.G + 25.H$	σ for $60.H$
	$2.G + 35.H$	
	$1.G + 56.H$	

Table 4.7: The Mimblewimble transactions T_1 and T_2.

The inputs and outputs of T_1 and T_2 can be aggregated into the single block shown in Table 4.8. As can be seen, the sum of the transaction excess $T_{1excess} + T_{2excess}$ is 77, which is equal to the total "excess" of outputs and inputs of the block. Nodes can therefore verify the block validity by aggregating the Schnorr signatures and verifying them against the resulting block excess, irrespective of which transaction the inputs and outputs originally belonged to.

Inputs	**Outputs**	**Signatures**
$10.G + 68.H$	$1.G + 56.H$	σ for $60.H$
$13.G + 56.H$	$2.G + 35.H$	σ for $17.H$
$20.G + 45.H$	$7.G + 91.H$	
	$10.G + 25.H$	
	$11.G + 13.H$	
	$12.G + 26.H$	

Table 4.8: Final block containing the aggregated and indistinguishable transactions T_1 and T_2.

The resulting block in 4.8 actually represents an non-interactive CoinJoin [58], where multiple parties combine their inputs into a single transaction. As a result of this, building a transaction graph for a Mimblewimble is unfeasible at the blockchain level, as it is impossible to determine which inputs (and outputs) belong to the same transaction and which to a different one. As previously discussed however, as this aggregation process happens at the network level, this is not enough to ensure *complete privacy*.

To allow for transaction aggregation, it is however necessary to include the blinding excess for each transaction, as it won't be able to be derived, given that all inputs and outputs are mixed. A block will therefore consist of the mixed inputs and outputs of all the transactions in the block, and a list of "kernel excesses" for each transaction.

Inputs	**Outputs**	**Kernels**
in1	(out1, rangeproof)	(kernel excess, sig, fees)
in2	(out2, rangeproof)	(kernel excess, sig, fees)
...	...	...

Table 4.9: Mimblewimble block format.

A problem exists with this approach however: despite inputs and outputs being mixed, it is still possible to try all combinations of $outputs - inputs$ to derive a transaction signing to one of the kernel excesses (i.e. finding original inputs and outputs for a transaction).

To address this issue, a *kernel offset* is used to add a *blinding factor* to the kernel excess. The new kernel excess is then computed as: $(r - kernel\_offset).G$. When transaction are aggregated, all kernel offsets are summed together, which prevents any of the original offsets from being uncovered.

4.3 Transaction cut-through

Similarly to how multiple transactions can be aggregated within a block, multiple transactions can be aggregated from different blocks as well. This aggregation introduces the possibility of some output commitments having an equal input commitment, which would essentially signify that an output in a block was used as an input in a later block. Note that by applying this pruning (or transaction cut-through) technique to all "intermediate" (i.e. spent) transactions, the blockchain's size can be significantly reduced. An example of transaction cut-through can be seen in Table 4.10.

Inputs	**Outputs**
input1(out1)	out2 ◇
input2(out2) ◇	out3 •
input3(out2) ◇	out4
input4(out3) •	out5

Inputs	**Outputs**
input1(out1)	out4
	out5

Table 4.10: A Mimblewimble block before and after cut-through.

Since all spent transactions can be safely removed, the only information that needs to be stored in the blockchain are the current UTXO set, the total amount of coins created by mining, and every transaction kernel. The whole blockchain can still be validated as if it was one transaction: add all unspent outputs, subtract the values $k.G$ from them add explicit mined coins times H (see equation 4.4). If the result is 0, the entire chain is considered valid.

$$\sum_{i=1}^{n} C_{out} = (\sum_{h=1}^{m} coins).G + (\sum_{j=1}^{l} kernel\_excess).H \tag{4.4}$$

This is why Mimblewimble is usually referred to as a very scalable protocol: its blockchain size is represented by the size of the UTXO set, rather than infinitely scaling with the number of transactions like other cryptocurrencies.

CHAPTER 5

NetNotes: a fully private Mimblewimble scheme

This section presents the NetNotes protocol, an extension of the Mimblewimble protocol aimed at satisfying the *untraceability* property even under network monitoring attacks.

5.1 Overview and Intuition

Recall the problem with Mimblewimble is that it does inherently provide a solution to achieve *untraceability* (i.e. only the spender should know which inputs are being spent in the transaction) at the protocol level. Dandelion++ has already been described in previous sections to be an insufficient workaround, but other ideas have also been proposed to try and improve Mimblewimble's privacy guarantees. For instance, Beam [73] implements decoy inputs carrying zero value, to ensure a CoinJoin is feasible even when the network is experiencing low traffic. Two-coin systems based on Lelantus have also been developed [87][88], but rather than aiming at fixing the underlying lack of *untraceability* in the Mimblewimble protocol, they instead aim at providing the option of achieving additional privacy on top of a Mimblewimble blockchain.

Lelantus, with its simple usage of One-out-of-Many proofs, is especially a good candidate to add the zero-knowledge layer needed to achieve *untraceability* in Mimblewimble. Recall how Lelantus works:

- It uses modified version of One-out-of-Many proofs [4] for double blinded commitments to prove that a valid coin is being spent from a set of coins, without revealing which one.
- It uses a serial number to prevent double-spends, which also acts like a public key through which a digital signature can be used to prove ownership of the coin.
- It uses a proprietary "Balance proof" to ensure that the extracted inputs and outputs cancel out.

The intuition is that if an input can be safely extracted from the double-blinded commitment pool, the Mimblewimble protocol itself guarantees that any output used in conjunction with the extracted input will not be able to generate money. This in turn means that the original "Balance proof" used by Lelantus can be removed. Furthermore, by picking the serial number to be a special commitment rather than a scalar, it is possible to make use of the original Groth-Kohlweiss's proof [4], whilst ensuring ownership and preventing double-spends at the same time. The following sections will explore how NetNotes achieves this in practice.

5.2 Constructions

In this section, the required cryptographic and data structures constructions needed to implement and secure the NetNotes protocol are presented.

Generalised Pedersen commitments

Sample a prime order group $\mathbb{G}$ of prime order p. Compute (G, H, J) as a tuple of generators to be used in a generalised Pedersen commitment $C = v.G + r.H + s.J$ where s is a second blinding factor.

$(G, H, J) \leftarrow\$ \mathbb{G}$
$Com_{gh}(v, r) = v.G + r.H$
$Com_{hj}(r, s) = r.H + s.J$
$Com(v, r, s) = v.G + r.H + s.J$

It is important to note that due to the structure of a Pedersen commitment, $Com(v, r)$ is identical to $Com(v, r, 0)$, and $Com(v, r, s) - Com(0, r, s) = Com_{gh}(v, 0)$.

Generalised Schnorr Proofs

A scheme is needed to prove that a commitment C is in a certain form (i.e. values have been committed to the right generators). To prove knowledge of the discrete logarithm of a scalar v for a blinded value $v.G$, a Schnorr signature can be used.
Similarly, to prove possession of the opening for a Pedersen commitment $C = v.G + r.H$, an extension of the Schnorr protocol can be used to prove knowledge of the discrete logarithm for a tuple (v, r). The generalised protocol is as follows:

GenProof (v, r)

1 : $a, b \leftarrow\$ \mathbb{Z}_p$
2 : $R = a.G + b.H$
3 : $e = \mathscr{H}(C||R)$
4 : $s1 = a + e * v \in \mathbb{Z}_p$
5 : $s2 = b + e * r \in \mathbb{Z}_p$
6 : **return** $(s1, s2, R)$

Verify$(C, (s1, s2, R))$

1 : $e = \mathscr{H}(C||R)$
2 : **accept if** $s1.G + s2.H \stackrel{?}{=} R + e.C$

This can be used to prove that a commitment C is in the form $C = v.G+r.H$, which is important to validate the correctness of a revealed NetNotes input, as will be explained in the spending protocol.

Shielded set

Define a shielded set $STXO$ as a list of generalised Pedersen commitments in the form $C = v.G + r.H + s.J$ where $v.G + r.H$ represents a coin and $s.J$ a randomly sampled blinding factor. Note that the blinding factor s, unlike Lelantus, does not represent a serial number by itself and is never revealed during the spending protocol.

A node must therefore maintain the following unprunable data structures:

1. The *STXO set*, which replaces the functionality of the UTXO set used in the Mimblewimble protocol.
2. The *"spent inputs" set*, which is a list of coins in the form $v.G + r.H$. Since a coin can only be spent by revealing its commitment in the form $v.G + r.H$ (i.e. without $s.J$), this set is needed to check for attempted double-spends.

5.3 Spend protocol

To spend a coin, the owner must provide proof that they know the blinding factors r and s and the value v of the coin without revealing the values themselves or which output in the STXO set they correspond to. To achieve this, a One-out-of-Many proof **oenmany** can be used to prove knowledge of a single element within the set in the form $r.H$ only.

For Alice to spend a coin $C = Com(v, r)$ saved in the STXO set in the form $Com(v, r, s)$, the scheme is as follows:

1. Alice reveals $C_{in} = Com_{gh}(v, r)$ along a Schnorr signature proving that the revealed commitment is in the form $v.G + r.H$ and is therefore not hiding any additional value $s.J$.
2. Alice subtracts C_{in} from every commitment in the STXO set. The new set will contain commitments in the form $\dfrac{Com(v, r, s)}{Com(v, r, 0)} = Com_{hj}(0, s)$
3. Alice forms a one-out-of-many proof π_{ooom} that she has knowledge of a commitment $Com_{hj}(0, s)$ in the resulting set.
4. Alice passes C_{in} and σ to a blockchain node, who can verify that Alice has knowledge of a commitment $Com_{hk}(0, s)$ in the resulting set. It follows that Alice also has knowledge of both v and r; or she would not be able to cancel them out from the generalised commitments to derive a commitment in the form of $s.J$ only.
5. If the proof is valid, Alice is free to use the coin C_{in} as input in a NetNotes transaction.

Note that the first reveal of the commitment C_{in} is necessary to be accompanied with a Schnorr signature proving the revealed commitment to be in the correct form $v.G+r.H$. This is to ensure the commitment is not hiding any value $s'.J$, as that would allow Alice to prove knowledge of infinite commitments $Com_{hj}(0, s - s')$ in the resulting set and thus double-spend.

Also recall how in Mimblewimble, ownership of an input is given by the fact that knowing the blinding factor r of a commitment C_{in} allows the sender to sign to the correct excess public key. Since NetNotes nodes cannot know the original output used as input, and therefore cannot calculate the correct excess blinding factor, it becomes impossible to prove ownership of an input through the Schnorr signature. The nodes can however verify that an input was correctly derived from a generalised commitment in the STXO set, which can only be achieved if the spender has knowledge of the opening of such commitment — thus proving ownership. Nevertheless, a generalised Schnorr signature is still needed to be created during the transaction building process, to check that no money is created or destroyed by ensuring that the difference between outputs and inputs is a public key for $r.H + s.J$, with no leftover value $v.G$.

5.4 Transaction data

A transaction using n unspent inputs C_{in} and creating m new outputs C_{out} for the proposed scheme would be modified to contain the following:

Inputs:

- A list of Pedersen commitments $(C_{in_1}, \ldots, C_{in_n})$ representing the input coins; each revealed C_{in} is to be stored in a set to prevent future attempts at double-spending.
- A list of Schnorr signatures $(\sigma_{gh_1}, \ldots, \sigma_{gh_n})$ proving that every revealed commitments in the form $v.G + r.H$ and not hiding any blinding factor $s.J$.
- A list of One-out-of-Many zk-proofs $(\pi_{ooom_1}, \ldots, \pi_{ooom_n})$ proving that for every C_{in}, the owner has knowledge of a commitment $Com_{hj}(0, s)$ in the resulting set.

Outputs:

- A list of generalised Pedersen commitments $(C_{stxo_1}, \ldots, C_{stxo_m})$, each representing a Mimblewimble output committed to an additional blinding factor s. These commitments are to be stored in the STXO set.
- A list of of range proofs $(\pi_1, \ldots, \pi_m)$ verifying that the committed values are not negative.
- A list of Schnorr signatures $(\sigma_{hj_1}, \ldots, \sigma_{hj_m})$ proving that the transaction did not generate any money.

The complete scheme for a NetNotes transaction is presented in Protocol 4.

Protocol 4: A NetNotes transaction

Sender$(v_{in}, r_{in}, s_{in}, pos)$		**Receiver**$(STXO, spent)$
$C_{in} = v_{in}.G + r_{in}.H$		
$r_s, s_s, a_s, b_s \leftarrow\$ \mathbb{Z}_p$		
$C_{change} = v_c.G + r_s.H + s_s.J$		
$v_r = v_{in} - v_c$		
$r_{diff} = r_s - r_{in}$		
$X_s = r_{diff}.H + s_s.J$		
$K_s = a_s.H + b_s.J$		
	$\xrightarrow{C_{in}, C_{change}, X_s, K_s, v_r}$	
		$r_r, s_r, a_r, b_r \leftarrow\$ \mathbb{Z}_p$
		$C_r = v_r.G + r_r.H + s_r.J$
		$X_r = r_r.H + s_r.J$
		$K_r = a_r.H + b_r.J$
		$e = \mathscr{H}(X_s + X_r \|\| K_s + K_r)$
		$sig_{r1} = a_r + e * r_r$
		$sig_{r2} = b_r + e * s_r$
	$\xleftarrow{C_r, r_r.H, s_r.J, a_r.H, b_r.J, (sig_{r1}, sig_{r2}), \pi_r}$	
$e = \mathscr{H}(X_s + X_r \|\| K_s + K_r)$		
$sig_{r1}.H \stackrel{?}{=} a_r.H + e * r_r.H$		
$sig_{r2}.J \stackrel{?}{=} b_r.J + e * s_r.J$		
$sig_{s1} = a_s + e * r_{diff}$		
$sig_{s2} = b_s + e * s_s$		
$R = K_s + K_r$		
$excess = X_s + X_r$		
$s1 = sig_{r1} + sig_{s1}$		
$s2 = sig_{r2} + sig_{s2}$		
$\pi_{ooom} = ((s_{in}, pos), \frac{STXO}{C_{in}})$		**Node**
	$\xrightarrow{C_{in}, C_{change}, C_r, (s1, s2, R), excess, \sigma_{gh}, \pi_c, \pi_r, \pi_{ooom}}$	
		$C_{in} \stackrel{?}{\notin}$ spent inputs set
		$e = \mathscr{H}(R \|\| excess)$
		$s1.H + s2.J \stackrel{?}{=} R + e * excess$
		$\mathsf{Vf}(C_{in}, \sigma_{gh}) \stackrel{?}{=} 1$
		$\mathsf{Vf}(\frac{STXO}{C_{in}}, \pi_{ooom}) \stackrel{?}{=} 1$
		$\mathsf{Vf}(C_{change}, \pi_c) \stackrel{?}{=} 1$
		$\mathsf{Vf}(C_r, \pi_r) \stackrel{?}{=} 1$
		add C_{in} to spent set
		add C_{change}, C_r to STXO set

5.5 Coinbase transactions

Lelantus is based on a two-coin protocol, where the base coin is publicly known and can be "burned" in exchange for a private coin. NetNotes is instead intended to be a *one-coin* protocol, where the base coin is by default private.

Recall that when a new block is mined, the miner should receive some coins as reward, known as a *coinbase transaction*. These transactions are the only ones allowed to create money in the system, and due to their inflationary nature, it is impossible to hide the amount of money added to the supply.

To take ownership of such public value, this protocol requires the miner to include a commitment $Com_{cb} = v.G + r.H + s.J$ to a mined block, with v being the value of the coinbase transaction and r and s being randomly sampled blinding factors.
To prove that the commitment encodes the value v, the miner must include a Schnorr proof signing for $Com_{cb} - v.G$ (i.e. $r.H + s.J$, to prove that v was cancelled out).
If the Schnorr signature is successfully verified, nodes will include the commitment Com_{cb} in the STXO set and ownership will be therefore granted to the miner. Despite the fact that the public knows how much value is encoded in Com_{cb}, due to the spend protocol, it is now impossible to tell which input will be used to spend it.

CHAPTER 6

Protocol Implementation

This chapter will focus on the implementation of the Mimblewimble and NetNotes protocols. A discussion of the design choices made during the implementation phase will be provided, followed by a description of developed Rust modules. Finally, an overview of how benchmarking was implemented in order to evaluate the performance of the protocols will be presented.

6.1 Design choices

This section will discuss the cryptographic and technological choices made during the implementation stage.

Rust

To implement the NetNotes protocol, the Rust programming language [89] was chosen. Rust is becoming increasingly popular for implementing cryptographic protocols thanks to its very secure memory management features, which ensure common exploits are mitigated. Alongside a growing community of cryptographic developers is also a great ecosystem of easy-to-use cryptographic libraries, which will be leveraged for this project.

Elliptic curve

Despite other implementations of Mimblewimble using the secp256k1 elliptic curve [72][90], Curve25519 was instead chosen for elliptic curve operations due to the more performant Rust library `curve25519-dalek` [91], which benchmarks to be ∼3x faster at signing and ∼2x faster at verifying ECDSA signatures [92].

The library supports group operations on both Ed25519 and the prime-order Ristretto group. As Ed25519 is not a prime-order group, it is not suitable for all zero-knowledge proof schemes, such as Ring signatures; this issue famously created a double-spend vulnerability in all CryptoNote-based cryptocurrencies [93], such as Monero [1]. To avoid this vulnerability, the Ristretto group of `curve25519-dalek` was chosen as the prime-order group for Schnorr signatures, Pedersen commitments and One-out-of Many proofs in both protocols. The `curve25519-dalek` library is licensed with the open-source *BSD-3-Clause license*.

Hash function

The official Rust implementation of the cryptographic hash function BLAKE3 [94] was used in the protocol for the purposes of generating Schnorr challenges. BLAKE3 was chosen over common alternatives such as SHA256 due to its speed; for instance benchmarks show BLAKE3 to be 14 times faster than SHA-256 [94]. The library is under the open-source *Apache 2.0 license*.

One-out-of-many proof

The `one_of_many_proofs` library was used to create and verify One-out-of-Many proofs. The library is a pure Rust implementation of the original protocol by Groth and Kohlweiss [4], further adding some improvements from [30] and [95]. The library is not an official implementation and is therefore not provably secure as it has not been independently audited; nevertheless, for the purpose of this project, this was not a concern. The library is under the open-source *MIT license*.

Bulletproofs

The Rust library for bulletproofs, simply called `bulletproofs` [96], and also developed by the "dalek-cryptography team", is known to offer the fastest implementation of the protocol [25]. Although initially intended for use in the Mimblewimble and NetNotes protocols, the library was dropped due to the absence of support for rangeproofs over generalised Pedersen commitments.

No other libraries supporting generalised Pedersen commitments were found and an attempt at generalising the bulletproofs protocol was unfeasible within the time scope of this project, due to the protocol's complexity. Given that bulletproofs could not be implemented in the NetNotes protocol, to ensure a fair comparison between the two protocols, the decision to not implement bulletproofs in the Mimblewimble protocol either was made.

6.2 Repository structure

Listing 6.1 shows the directory structure of the project, which largely follows the standard Rust project structure.

Listing 6.1: Project directory structure

```
netnotes/
  |- .github/ // CI/CD definition for deploying benchmarks on GitHub pages
  |- benchmarks/
  |- src/  // unit tests and the source code for the project
    |- lib.rs // crate entry point
    |- mimblewimble.rs
    |- netnotes.rs
    |- pedersen.rs
    |- schnorr.rs
  |- tests/ // integration tests for the project
    |- mimblewimble_tests.es
    |- netnotes_tests.rs
  |- Cargo.toml // metadata about the project and its dependencies
  |- README.md
```

The external libraries (or *crates*) discussed in the previous chapter are defined in the *Cargo.toml* file. The project's source code was divided into various Rust modules, which will be covered in the the following sections.

6.3 Pedersen module

Given that the `one-of-many-proofs` library includes a Pedersen generation and commit function, the implementation of the Pedersen module was done by providing a wrapper around the library and extending it with the necessary functionality. Such functionality includes generalised commitments, Pedersen commitments over different generators and commitments over a single generator.

Listing 6.2: Pedersen module

```
lazy_static! {
    static ref J: RistrettoBasepointTable = RistrettoBasepointTable::create(
        &RistrettoPoint::hash_from_bytes::<Keccak512>(G.compress().as_bytes())
    );
    pub static ref GENS: Pedersen = Pedersen::new(2, 13);
    pub static ref GENS_MEDIUM: Pedersen = Pedersen::new(8, 5);
    pub static ref GENS_LARGE: Pedersen = Pedersen::new(4, 8);
}

pub struct Pedersen(pub ProofGens);
```

```
impl Pedersen {
    pub fn new(n_bits: usize) -> Self {
        Pedersen(ProofGens::new(n_bits).unwrap())
    }

    // s.G + r.H + v.J
    pub fn generalised_commit(&self, value: Scalar, r: Scalar, s: Scalar) ->
      GeneralisedCommitment {
        self.commit(r, s) + self.commit_J(value)
    }

    // v.H + r.G
    pub fn commit(&self, value: Scalar, r: Scalar) -> Commitment {
        self.0.commit(&value, &r).unwrap()
    }

    // r.H + v.J
    pub fn commit_hj(&self, value: Scalar, r: Scalar) -> Commitment {
        self.commit_H(value) + self.commit_J(r)
    }

    pub fn commit_G(&self, value: Scalar) -> Commitment {
        self.commit(Scalar::zero(), value)
    }

    pub fn commit_H(&self, value: Scalar) -> Commitment {
        self.commit(value, Scalar::zero())
    }

    pub fn commit_J(&self, value: Scalar) -> RistrettoPoint {
        &*J * &value
    }
}
```

As can be seen in Listing 6.2, generator point J was chosen to be the `Keccak512` hash of the compressed representation of the Ristretto basepoint G (BLAKE3 could not be used due to not implementing trait `curve25519_dalek::digest::Input`). Generator point H was provided by the library and is is the `Sha3_512` hash of the compressed representation of the Ristretto basepoint G.

The `lazy_static` crate [97] was used to define the base point J and Pedersen generators as lazily evaluated static variables. This was done to avoid the overhead of re-computing the generators for each Pedersen commitment. Due to globally mutable singletons being an anti-pattern in Rust, different sized generators had to be defined separately, for use during benchmarks of the NetNotes protocol. Also note that the number of bits supported by the Pedersen generators defines the size of the anonymity set able to be used in the *one-of-many proof*.

Most importantly, it should be noted how the generalised commitment for values (v, r, s) was implemented over (J, H, G) generators, rather than the (G, H, J) described in chapter 5. This was done to ensure that the generalised commitments were compatible with the `one-of-many-proofs` library: as the proof is implemented over the Ristretto basepoint G, the second blinding factor "s" had to be committed over the Ristretto basepoint G as well. In order to allow for a Schnorr signature over G, the commitment scheme for Mimblewimble was also implemented over $r.G + v.H$ rather than $v.G + r.H$.

6.4 Schnorr module

Schnorr signatures were implemented from scratch by using the Ristretto group provided by the `curve25519-dalek` crate. Although various Rust libraries were available for Schnorr signatures, none could be found that implemented generalised Schnorr signatures over custom generators.

The created module contains both implementations for the standard Schnorr signature scheme and the generalised scheme, as represented by the following `structs` in Listing 6.3.

Listing 6.3: Schnorr structs

```
pub struct Signature {
    s: Scalar,
    R: PublicKey,
}

pub struct GeneralisedSignature {
    s1: Scalar,
    s2: Scalar,
    R: PublicKey,
}
```

Different implementations are provided for each `struct` to support the various operations required by the protocol. For instance, the standard Schnorr signature implements the following methods for signature generation and verification over generator point G:

Listing 6.4: Schnorr signature implementation

```
impl Signature {
  pub fn new(nonce: &Keypair, private_key: &PrivateKey, e: Scalar) -> Self {
    let signature = nonce.private + e * private_key;  // s = r + e * x
    Signature { s: signature, R: nonce.public }
  }

  pub fn verify(&self, public_key: &PublicKey, e: Scalar) -> bool {
    let sG = PublicKey::from_private_key(self.s).0;
    sG == self.R.0 + e * public_key.0  // s.G == R + e.X
  }
}
```

The `GeneralisedSignature struct` instead implements signing and verifying over generator pairs (G, H) and (H, J), as seen in Listing 6.5. This is due to the fact that NetNotes protocol requires a signature on the excess $s.G + r.H$ to verify non-inflation and a second signature to verify an input used is of the form $r.H + v.J$

Listing 6.5: Generalised schnorr signatures implementation

```
impl GeneralisedSignature {
  // For signatures in the form s.G + r.H
  pub fn new_excess_proof(
      nonce_G: &Keypair,
      nonce_H: &KeypairH,
      value: &PrivateKey,
      blinding_factor: &PrivateKey,
      challenge: Scalar,
  ) -> Self {
      // s1 = a + e * v
      let signature1 = nonce_G.private + challenge * value;
      // s2 = b + e * r
      let signature2 = nonce_H.private + challenge * blinding_factor;

      GeneralisedSignature {
          s1: signature1,
          s2: signature2,
          R: nonce_G.public + nonce_H.public,
      }
  }

  pub fn verify_excess_proof(&self, commitment: Commitment, e: Scalar) -> bool {
      let s1G = PublicKey::from_private_key(self.s1);
      let s2J = PublicKeyH::from_private_key(self.s2);
      s1G.0 + s2J.0 == self.R.0 + e * commitment // s1.G + s2.J == self.R + e.C
  }

  // For signatures in the form r.H + v.J
  pub fn new_input_proof(value: PrivateKey, blinding_factor: PrivateKey)
    -> GeneralisedSignature {
      let nonce_H = Scalar::random(&mut OsRng);
      let nonce_J = Scalar::random(&mut OsRng);
      let commitment = GENS.commit_hj(blinding_factor, value);
      let nonces_com = GENS.commit_hj(nonce_H, nonce_J);

      let challenge = Self::calculate_challenge(&commitment, &nonces_com);
      let s1 = nonce_H + challenge * blinding_factor;
      let s2 = nonce_J + challenge * value;
      let R = PublicKey(nonces_com);

      GeneralisedSignature { s1, s2, R }
  }
```

```
    // s1.H + s2.J == R + e.C
    pub fn verify_input_proof(&self, comm: &Commitment) -> bool {
        let e = Self::calculate_challenge(&comm, &self.R.0);
        GENS.commit_H(self.s1) + GENS.commit_J(self.s2) == self.R.0 + e * comm
    }
}
```

The `schnorr` module also includes various helper functions used to generate keypairs for all the curve generators used, as well as an `aggregate` function which can be used to aggregate the partial signatures used during the Mimblewimble and NetNotes protocols. In general, for Mimblewimble, the challenge is generated using the `lock_height` and `tx_fee` fields of a transaction; since these fields have not been implemented in either protocol, the signature has been performed on an empty message instead.

Limitations

Schnorr batch verification was not implemented as it was deemed out of scope for the project. Given the small number of signatures that need to be validated during benchmarks, the performance impact of not implementing batch validation is however negligible.

6.5 Mimblewimble module

The decision to implement the NetNotes protocol from scratch was made, despite Grin [72] already providing a core Mimblewimble library for Rust. This decision was made due to the Grin library using the *secp256k1* curve, rather than intended *Ed25519* curve, which would have required significant refactoring. Creating a minimal Mimblewimble implementation also has another advantage: it allows for the comparison between Mimblewimble and NetNotes to be more accurate, as under the hood both protocols would be using the same underlying `mimblewimbe` module.

Taking into account the interactive nature of the Mimblewimble protocol, the module has been separated into three phases, plus an "initiation" phase. These phases are as follows:

1. **Init** – The initialisation phase of the protocol, where the sender generates a transaction.
2. **Send** – The sender sends the necessary transaction data to the receiver.
3. **Respond** – The receiver receives the transaction data from the sender, verifies it, forms his partial signature and sends it back to the sender.
4. **Finalise** – The sender receives the receiver's partial signature, verifies it, forms his own partial signature and finalises the transaction.

Each phase has a corresponding data structure and implements a method needed to be called in order proceed to the next phase. The corresponding data structures are shown in Listing 6.6.

Listing 6.6: Mimblewimble protocol data structures

```
pub struct Kernel { excess: PublicKey, signature: Signature }

pub struct Transaction {
  inputs: Vec<Input>,
  outputs: Vec<Output>,
  kernel: Kernel,
}

// === Intermedary transaction data ===
// Transaction data generated by the sender
pub struct TxData {
    inputs: Vec<Input>,
    amount: Scalar,
    change_output: Commitment,
    nonce_keypair: Keypair,
    blinding_keypair: Keypair,
}

// Data sent to the receiver
pub struct SendData {
    amount: Scalar,
    public_nonce: PublicKey,
    public_blinding_diff: PublicKey,
}

// Response data sent back to the sender
pub struct ResponseData {
    partial_sig: Signature,
    output_commitment: Commitment,
    public_nonce: PublicKey,
    public_blinding: PublicKey,
}
```

Init phase

The `Transaction struct` represents a fully formed Mimblewimble transaction and in general implements both the `init` and `verify` methods, which are used at beginning and end of the protocol respectively.

The `init` method is used to generate the nonces and blinding factors that will be used in the protocol by the sender, as well as set the amount and change of the transaction. It returns a `TxData` struct, which keeps track of the data that will need to be reused later in the protocol.

The `verify` method instead verifies that a Mimblewimble transaction was formed correctly. It mainly checks that the kernel excess provided is derived correctly, as well as validating that the kernel signature is valid for the calculated excess.

Both implementation of `init` and `verify` be found in Listing 6.7.

Listing 6.7: Transaction struct

```
impl Transaction {
  /// Sender commits to amount, fee, and a public nonce, public excess.
  pub fn init(
      amount: Scalar,
      change: Scalar,
      input_blinding_factors: Vec<Scalar>,
      inputs: Vec<Input>,
  ) -> TxData {
      // Generate change_output
      let r_output = Scalar::random(&mut OsRng);
      let change_output: Commitment = GENS.commit(change, r_output);
      // Calculate the difference between outputs' and inputs' blinding factors
      let r_input = input_blinding_factors.iter().sum::<Scalar>();
      // Generate nonces and blinding factors
      let nonce_keypair = Keypair::generate();
      let blinding_keypair = Keypair::from_private_key(r_output - r_input);

      TxData {
          inputs,
          amount,
          change_output,
          nonce_keypair,
          blinding_keypair,
      }
  }

  pub fn verify(&self) -> bool {
      let excess = self.kernel.excess;
      let signature = &self.kernel.signature;

      // get outputs - inputs
      let expected_excess = PublicKey(
          self.outputs.iter().sum::<RistrettoPoint>()
              - self.inputs.iter().sum::<RistrettoPoint>(),
      );

      let challenge = Signature::calculate_challenge(&signature.R, &excess);
      let verify_kernel_excess =
          Signature::verify(signature, &excess, challenge);

      (expected_excess == excess) && verify_kernel_excess
  }
}
```

Send phase

The `TxData` struct implements a `send` method used to simulate the sender communicating the necessary data to the receiver. Its implementation can be seen in Listing 6.8.

Listing 6.8: Send method

```
impl TxData {
    pub fn send(&self) -> SendData {
        SendData {
            amount: self.amount,
            public_nonce: self.nonce_keypair.public,
            public_blinding_diff: self.blinding_keypair.public,
        }
    }
}
```

Respond phase

The respond method is called by the receiver, and accepts a `SendData struct` as input to generate a `ResponseData struct` as output. Its implementation is shown in Listing 6.9.

Listing 6.9: Respond method

```
impl SendData {
    /// The receiver adds their output, commits to their
    /// public nonce and excess, and signs for their half of the kernel.
    pub fn respond(data: &SendData) -> ResponseData {
        let nonce_keypair = Keypair::generate();
        let blinding_keypair = Keypair::generate();
        let challenge = Signature::calculate_challenge(
            &(data.public_nonce + nonce_keypair.public),
            &(data.public_blinding_diff + blinding_keypair.public),
        );
        let partial_sig = Signature::new(
            &nonce_keypair,
            &blinding_keypair.private,
            challenge
        );
        ResponseData {
            partial_sig,
            output: GENS.commit(data.amount, blinding_keypair.private),
            public_nonce: nonce_keypair.public,
            public_blinding: blinding_keypair.public,
        }
    }
}
```

Finalise phase

In the finalise phase, the sender receives the `ResponseData struct` from the receiver, and uses it to build the final transaction to obtain the final `Transaction struct`, with which network nodes can call the `verify` method to validate the transaction. Its implementation can be seen in Listing 6.10.

Listing 6.10: Finalise method

```
impl ResponseData {
  /// The sender adds their half of the signature,
  /// aggregates the 2 partial signatures, and builds the final transaction.
  pub fn finalise(tx: &TxData, resp: &ResponseData) -> Transaction {
      let nonce_sum = tx.nonce_keypair.public + resp.public_nonce;
      let blinding_sum = tx.blinding_keypair.public + resp.public_blinding;

      let challenge = Signature::calculate_challenge(&nonce_sum, &blinding_sum);
      let verify = Signature::verify(
          &resp.partial_sig,
          &resp.public_blinding,
          challenge
      );

      if verify == false {
          panic!("Signature␣verification␣failed");
      }

      let partial_sig = Signature::new(
          &tx.nonce_keypair,
          &tx.blinding_keypair.private,
          challenge
      );
      let partial_sigs = vec![&partial_sig, &resp.partial_sig];
      let signature = Signature::aggregate(partial_sigs);

      Transaction {
          inputs: tx.inputs.clone(),
          outputs: vec![tx.change_output, resp.output_commitment],
          kernel: Kernel {
              excess: blinding_sum,
              signature,
          },
      };
  }
}
```

Limitations

This module only acts as a minimal implementation of the Mimblewimble protocol, and many features that would make this implementation secure in a permissionless blockchain setting – such as kernel offset, fees and lock height – have been omitted. Its biggest limitation is definitely the missing support for bulletproofs, which represent a vital part of the scheme's security, as well as its main performance bottleneck.

6.6 Netnotes module

The `netnotes` module shares many similarities with the `mimblewimble` module, including similarities in their respective methods and its encapsulation into 4 separate phases. As such, the only implementation worth covering is the addition of the One-out-of-many proofs and generalised Schnorr signature for the `init` method of the `Transaction struct`. This implementation of the initial phase of the NetNote's spend protocol can be seen in Listing 6.11.

Listing 6.11: Netnotes' Transaction struct implementations

```
impl Transaction {
    pub fn init(
        amount: Scalar,
        change: Scalar,
        inputs: Vec<Input>,
        inputs_values: Vec<Scalar>,
        inputs_blinding_r: Vec<Scalar>,
        inputs_blinding_s: Vec<Scalar>,
        stxo_positions: Vec<usize>,
        stxo_set: Vec<GeneralisedCommitment>,
    ) -> TxData {
        let gens = Pedersen(derive_gens(&stxo_set).clone());
        // Generate Schnorr nonces
        let nonce_G = Keypair::generate();
        let nonce_H = KeypairH::generate();
        // Generate blinding factors for change output
        let r_output = Scalar::random(&mut OsRng);
        let s_output = Scalar::random(&mut OsRng);

        // Generate the senders' output commitment (change)
        let change_output: Commitment =
            gens.generalised_commit(change, r_output, s_output);

        // Calculate commitments to send to receiver for
        // generating partial signature
        let blinding_diff = r_output - inputs_blinding_r.iter().sum::<Scalar>();
        let nonces_com = (nonce_G.public + nonce_H.public).0;
        let excess_com = gens.commit(blinding_diff, s_output);
```

```
        // Generate schnorr proofs for every input proving the commitment
        // is in the form r.H + v.J
        let schnorr_proofs =
            Self::generate_schnorr_proofs(inputs_values, inputs_blinding_r);

        // Generate ooom proofs for every input proving the knowledge of the
        // openings of a commitment in the STXO set
        let ooom_proofs =
            Self::generate_ooom_proofs(
                stxo_set,
                stxo_positions,
                &inputs,
                inputs_blinding_s
            )
        };
        TxData {...}
    }
}
```

6.7 One-of-many-proofs module

Due to its usage of Gray codes [98], the `one-of-many-proofs` library [99] used in the NetNotes implementation was implemented for binary (2-ary) commitment only. While this approach can speedup batch verification by 60% for small sets, generating and verifying proofs for larger sets proved to be significantly slower than the C++ implementation of One-out-of-Many proofs [100]. It was therefore decided to implement n-ary commitments as described in [30]. For this, the repository at [98] was forked.

Most of the work comprised of generalising operations performed on vectors to work on matrices of size $n \times m$, where n and m are used to derive the total size of the anonymity set $N = n^m$. As can be seen in listing 6.12, this was necessary as variables such as `a_j_i` and `b_j_i` were hardcoded to return a matrix of size $2 \times m$.

Listing 6.12: Generalising bit proof proofs

```
#[Original code]
let a_j_i = iter::once(a_j.clone())
.chain(iter::once(a_j.iter().map(|a| -a).collect()))
.collect::<Vec<Vec<Scalar>>>();
#[Modified code]
let mut a_j_i = vec![vec![Scalar::default(); self.n_bits]; self.n_base];
for j in 0..self.n_bits {
    a_j_i[0][j] = -a_j_1.iter().map(|a| a[j]).sum::<Scalar>();
    for i in 0..a_j_1.len() {
        a_j_i[i + 1][j] = a_j_1[i][j];
    }
}
```

As the challenge vector `f_j_i` is calculated based on `a_j_i` and `b_j_i`, and the vector `f_j_0` is omitted from the communication between prover and verifier, this essentially allowed the original library to reduce the matrix to a single vector of size m for most operations. This meant that most operations in the verification and proving phase had to be refactored in order work on matrixes rather than single vectors, as shown in Listing 6.13.

Listing 6.13: Example of changes made to implement n-ary commitments

```
#[Original]
// Proof generation
let f1_j = a_j_i[1]
.iter()
.zip(b_j_i[1].iter())
.map(|(a, b)| b * x + a)
.collect();

// Proof verification - inflate f1_j to include reconstructed f0_j vector
let f_j_i = iter::once(proof.f1_j.iter().map(|f| x - f).collect())
    .chain(iter::once(proof.f1_j.clone()))
    .collect::<Vec<Vec<Scalar>>>();

#[Modified code]
// Proof generation
let mut f_j_1 = vec![vec![Scalar::default(); b_j_i[0].len()]; b_j_i.len() - 1];
for i in 1..b_j_i.len() {
    for j in 0..b_j_i[0].len() {
        f_j_1[i - 1][j] = b_j_i[i][j] * x + a_j_i[i][j];
    }
}

// Proof verification - inflate f1_j to include reconstructed f_j vector
let mut f_j_i = vec![vec![Scalar::default(); self.n_bits]; self.n_base];
for j in 0..self.n_bits {
    f_j_i[0][j] = x - f_j_1.iter().map(|f| f[j]).sum::<Scalar>();
    for i in 0..f_j_1.len() {
        f_j_i[i + 1][j] = f_j_1[i][j];
    }
}
```

Once all the code was generalised to work with n-ary commitments, the library was changed to support an `n_base` parameter that allows to arbitrarily pick n when instantiating the generators for the protocol: `ProofGens::new(n_base, pow)`.

Further optimisations

The main bottleneck of the protocol is computing the polynomial G_k, which requires $mN + 3m$ group exponentiations. Thus, it follows that the implementation of n-ary commitments allowed for a significant perfomance optimisation, as now large anonymity sets can be achieved by using

an arbitrarily large n paired with a small m value. This possible optimisation was in fact noted by the creator of the library, and listed as one of the 6 enhancements in the repository's Github issues [101].

Aside from n-ary commitments, another 4 enhancements from the list have been further implemented, namely:

1. "Scalar multiplication may be accelerated with basepoint tables"
2. "Performance can be improved with multi-threaded computation"
3. "Use RistrettoPoint multiscalar multiplication utilities in proof and verification"
4. "Missing benchmarks"

The following sections will explore how each optimisation was implemented.

Basepoint tables

The `curve25519-dalek` library, which the `one-of-many-proofs` library uses for elliptic curve operations, provides a `RistrettoBasepointTable` containing pre-computed multiples of a basepoint. This `struct` can be used instead of the standard `RistrettoPoint` in order to accelerate scalar multiplication.

As a result, `RISTRETTO_BASEPOINT_POINT` was refactored to `RISTRETTO_BASEPOINT_TABLE` in the Pedersen commitment implementation, which resulted in an overall minor speedup.

Multi-threaded computations

To implement multi-threading, the `rayon` library [102] was used to parallelise iterators across multiple cores. The best operations to parallelise were narrowed down to generating the polynomial coefficients and calculating the polynomial itself, as they contain $O(N^2)$ operations and interate over the entire anonymity set. This meant that the Gray codes optimisation had to be removed, as it relied on sequential computation rather than parallel. The performance gains from this change however are very significant for a large anonymity set when performed on a multi-threaded machine.

The refactored code for generating the polynomial coefficients can be seen in Listing 6.14. Notice how the operations were separated into three different algorithms, namely one to create a matrix, and two to reduce it to vector sums needed for the following calculations. Also observe how the parallel iterator `par_iter` was only used to parallelise operations involving N elements, as benchmarking showed that parallelising shorter iterators led to an overall slowdown of the protocol.

Listing 6.14: Multi-threading implementation

```
let coeffs = self
    .par_iter()
    .enumerate()
    .map(|(i, _)| {
```

```
        proofs
            .iter()
            .enumerate()
            .map(|(p, _)| {
                (0..gens.n_bits)
                    .map(|j| f_p_j_i[p][bit(i, j, gens.n_base)][j])
                    .product::<Scalar>()
            })
            .collect::<Vec<Scalar>>()
    })
    .collect::<Vec<Vec<Scalar>>>();

let horizontal_sum = coeffs
    .par_iter()
    .map(|vec| vec.iter().sum::<Scalar>())
    .collect::<Vec<Scalar>>();

let vertical_sum = (0..coeffs[0].len())
    .into_iter()
    .map(|i| {
        coeffs
            .par_iter()
            .map(|row| row[i])
            .reduce(|| Scalar::zero(), |a, b| a + b)
    })
    .collect::<Vec<Scalar>>();
```

RistrettoPoint multiscalar multiplication

The `curve25519-dalek` library also implements a `multiscalar_mul` method, which is optimised for constant-time multiscalar multiplication with no precomputation. This method was used whenever possible, and was especially useful to improve the previous implementation of vector commitments. As can be seen by Listing 6.15, whenever possible the largest number of elements were multiplied together (in this case achieved by transposing the matrix) to make better use of the optimisation provided by the method. As such, the implementation of `multiscalar_mul` across the codebase lead to significant performance gains during benchmarks.

Listing 6.15: Multiscalar multiplication optimisations

```
let p_i_k = (0..self.len())
    .map(|i| compute_p_i(i, l, &a_j_i))
    .collect::<Vec<Vec<Scalar>>>();
let p_k_i = transpose(&p_i_k);

G_k.par_iter_mut().enumerate().for_each(|(k, coeff)| {
    *coeff += RistrettoPoint::multiscalar_mul(&p_k_i[k], &points_minus_offset);
});
```

Missing benchmarks

Given the small amounts of benchmarks available, several new benchmarks were added to the library in order to measure *proving time* and establish *proof size* for various combinations of n and m.

Compressed Ristretto points

Although not mentioned as a possible enhancement in [101], `curve25519-dalek`'s Ristretto points can be encoded into a 32-byte `CompressedRistretto` `struct`. By using the `compress()` method for every single Ristretto point in the One-out-of-Many proof – including all polynomial points – the overall proof size was reduced significantly. During verification, the resulting `CompressedRistretto` points have been decompressed in order to allow for operations to be performed on them.

6.8 Testing

This section outlines the testing strategy employed during the development of the Mimblewimble and NetNotes implementations.

6.8.1 Unit testing

Unit testing was performed to ensure correctness of the implemented modules. As per good Rust practice, the `std::test` library was used and tests were written in the same module they were testing, as well as prepended with the `#[cfg(test)]` attribute.

An example of a unit test for the `schnorr` module in shown code listing 6.16.

Listing 6.16: A unit test for the 'Schnorr' module

```
#[cfg(test)]
mod tests {
    use super::*;
    use curve25519_dalek::constants::RISTRETTO_BASEPOINT_POINT as G

    #[test]
    fn test_keypair_generation_1() {
        let keypair = Keypair::generate();
        let expected_public_key = PublicKey(G * keypair.private);
        // Verify that the public key can be reconstructed from the private key
        assert_eq!(keypair.public, expected_public_key);
    }
}
```

Unit testing was performed for every module, as the following breakdown shows:

- `schnorr.rs`: 12 tests
- `pedersen.rs`: 7 tests
- `mimblewimbe.rs`: 7 tests
- `netnotes.rs`: 11 tests

6.8.2 Integration testing

Integration testing was also performed through the Rust standard library as evidenced by integration tests found in the `tests/` directory. The tests have been written for both the Mimblewimble and NetNotes protocols to ensure the overall correct integration between them and the rest of the cryptographic modules. The total number of integration tests per implemented protocol is as follows:

- `mimblewimble_test.rs`: 5 tests
- `netnotes_test.rs`: 5 tests

An example of an integration test can be seen in Listing 6.17. This test can be seen to be using the `#[should_panic]` attribute, as it is testing whether the protocol successfully rejects an over-spend attempt.

Listing 6.17: Mimblewimble integration tests

```
#[test]
#[should_panic]
fn test_transaction_over_amount() {
    let (blinding, values, inputs) = setup();

    // pick change as 1
    let change = Scalar::one();

    // sum values to an integer and subtract change
    let amount = values.iter().fold(Scalar::one(), |acc, x| acc + x) - change;

    // Simulate transaction
    let tx_data = Transaction::init(amount, change, blinding, inputs.clone());
    let send_data = tx_data.send();
    let response_data = SendData::respond(&send_data);
    let transaction = ResponseData::finalise(&tx_data, &response_data);

    assert!(transaction.verify());
}
```

6.9 Benchmarking

The `criterion.rs` crate [103] was chosen for benchmarking over the usual `cargo bench`. This choice was made to leverage the crate's more advanced features, such as its plotting capabilities, which are useful for visualising the performance of the protocols.

In total, three benchmarking files have been created and are located in the `benchmarks/` directory:

- `comparison_benchmark.rs` – contains benchmarks comparing the performance of the Mimblewimble and Netnotes protocols for the same input sizes.
- `mimblewimble_benchmark.rs` – contains benchmarks for the generation and verification of the implemented Mimblewimble transactions for different input sizes.
- `netnotes_benchmark.rs` – contains benchmarks for the generation and verification of the implemented Netnotes transactions for different input sizes and anonymity sets sizes.

Listing 6.18 shows the benchmarking function used to test the NetNotes protocol's performance.

Listing 6.18: Netnotes benchmarking example

```
  fn setup() { ...} // mocks input data needed for a Netnotes transaction

  fn inputs() -> Vec<InputData> {
    let small = setup(SetSize::Small, 2);
    let medium = setup(SetSize::Medium, 2);
    let large = setup(SetSize::Large, 2);
    vec![small, medium, large]
}

  pub fn netnotes_benchmark(c: &mut Criterion) {
    let mut group = c.benchmark_group("netnotes");
    for i in inputs().iter() {
        group.bench_with_input(
            BenchmarkId::new("Generation", i.stxo_set.len()),
            i,
            |b, i| b.iter(|| gen_netnotes_transaction(black_box(i.clone()))),
        );
        let transaction = gen_netnotes_transaction(i.clone());
        group.bench_with_input(
            BenchmarkId::new("Verification", i.stxo_set.len()),
            i,
            |b, _i| b.iter(|| transaction.verify(i.stxo_set.clone())),
        );
    }
    group.finish()
}
```

In the above benchmark, the function used to generate and verify NetNotes proofs is called with varying inputs repeatedly in order to minimise statistical anomalies, before grouping the average computed time – as well as other statistical data – under the "netnotes" group. This data was then used to plot graphs offering a visual comparison of the function performance in relation to the different inputs sizes.

Proof sizes were also measured during benchmarks with the use of Rust's `std:mem`, which returns the underlying memory used by a variable. Heap-allocated elements were made sure to be derefenced before measurement, to ensure accurate results.

All benchmarks have been run on an Apple M1 Pro system with 8 cores and 16GB memory. All elliptic curve operations were optimised by compiling to use the SIMD backend. This was achieved by setting the environment variable `export RUSTFLAGS="-C␣target_cpu=native"`.

For instructions on how to run the benchmark, refer to the `README.md` file for the repository at [104]. The full report of benchmarking results and statistical analysis is also hosted at [105].

CHAPTER 7

Performance and Evaluation

This chapter will present the performance results for the improved `one-of-many-proofs` library, as well as for the implementations of the NetNotes and Mimblewimble protocols. Following, the security properties of the NetNotes protocol will be discussed, and a comparison between NetNotes and other Mimblewimble/Lelantus-based implementations will be offered.

7.1 Performance Analysis

7.1.1 One-out-of-Many proof performance

Table 7.1 showcases performance data of the original `one-of-many-proofs` library for large anonymity sets, while Table 7.2 contains the benchmark result for the improved library containing the modifications discussed in the previous chapter. Note that the results in Table 7.2 have been separated into "Single core" and "Parallel" to provide a fairer comparison with the old library.

$N = n^m$	n	m	Proof size (bytes)	Proof Time (ms)	Verification Time (ms)
8192	2	13	3232	3461	265
16384	2	14	3424	7440	521
32768	2	15	3616	15978	1054
65536	2	16	3808	34206	2063

Table 7.1: Performance data for the original "one-of-many-proofs" library.

As can be seen from the two tables, overall the improved library is much more performant on

$N = n^m$	n	m	Proof size (bytes)	Proof Time (ms)		Verification Time (ms)	
				Single core	Parallel	Single core	Parallel
8192	2	13	1056	1270	334	100	93
16384	4	7	1120	1283	293	189	183
32768	8	5	1504	1805	444	371	364
65536	4	8	1248	5837	1362	760	725
100000	10	5	1824	5494	1318	1130	1106
262144	8	6	1760	17393	3655	3013	2894
262144	64	3	6368	8665	3179	2930	2858
1000000	10	6	2144	66279	16115	11542	10821

Table 7.2: Performance data for the improved "one-of-many-proofs" library.

a single core for both proving and verification times for every given set size. This is mainly thanks to the implementation of n-ary commitments, although even when using identical n and m values, the improved library still performs better thanks to the several optimisations applied.

Proving time has especially been improved, with the modified library achieving 65% faster proof generation times for an anonymity set size of 65536, which is further increased to 96% when multi-threading is enabled. Verification times for the same set size has also improved by 63% and 65% for single core and multi-core respectively. Thanks to the compression of the Ristretto points, the proof size has also been reduced by 65% on average, which is critical for an ever-growing data structure like blockchain, where historical data cannot be pruned. Overall, thanks to the optimisations performed the new library has better performance across all measurements compared to the old library.

$N = n^m$	n	m	Proof size (bytes)	Proof Time (ms)	Verification Time (ms)
8192	2	13	1564	1368	124
16384	4	7	1412	1257	240
32768	8	5	1573	1895	454
65536	4	8	1576	5253	947
100000	10	5	2044	4615	1359
262144	8	6	2016	14468	3576
262144	64	3	6516	7214	3559
1000000	10	6	2400	54691	13592

Table 7.3: Performance data for Lelantus' C++ implementation of One-out-of-Many proofs. Adapted from [100].

There is however another library that the improved Rust implementation can be compared to: the Lelantus C++ implementation of One-out-of-Many proofs [106]. Comparing these two libraries is of relative interest, as not only are they the only two implementations of One-out-of-Many proofs, but also due to Lelantus' usage of the popular `secp256k1` C library [107], which is supposedly slower than `curve25519-dalek` [91]. Table 7.3 showcases performance data for the C++ protocol. As can be seen, the improved Rust library was benchmarked over the same set sizes in order to provide a fair comparison between the two implementations.

As can be seen by comparing Table 7.2 and Table 7.3, the improved `one-of-many-proofs` library has a slight performance advantage compared to the C++ implementation, even when not taking into account parallelisation. However, whilst the Rust implementation's proof generation times are generally faster, the C++ implementation is slightly more performant on larger sets. Nevertheless, when considering the multi-threading benchmarks, the Rust implementation is 80% faster than its counterpart for the largest set of 1000000 elements.

Looking at proof verification times, the Rust implementation is 17% faster on average across all set sizes for the single-core benchmarks. It can also be observed how enabling multi-threading offers minimal performance gains, due to the fairly simple verification involving a single proof. Surprisingly, the Rust implementation's proof sizes on average 10% smaller than those in the C++ implementation. This could be due to the `secp256k1` compressed Ristretto points being 33 bytes, in comparison to `curve25519-dalek`'s 32 bytes.

Batch verification

Batch verification is also a critical aspect for a privacy-preserving cryptocurrency protocol, as it represents the main bottleneck for network nodes receiving and verifying thousands of transactions per second. In essence, if a node cannot keep up with the verification process, it will be unable to form its own copy of the blockchain and will therefore be disqualified from participating in the network. This can lead to centralisation, as only powerful nodes become able to process the incoming transactions at the necessary speed.

The Lelantus paper [100] includes various performance data for batch verification over anonydid mity sets of sizes 16384, 32384, 65536 and 100000. The latter set size was used to compare Lelantus' implementation to the optimised `one-of-many-proofs` Rust library's batch verification. Performance data for the Lelantus implementation can be found in Table 7.4, while performance data for the Rust implementation can be found in Table 7.5.

Batch size	Verification Time (ms)	Average cost per verification (ms)
5	1578	315.6
10	1853	185
50	4242	85
100	6997	70
500	51483	103
1000	92263	92.3

Table 7.4: Lelantus' batch verification performance for set size 100000. Adapted from [100].

In general, as both implementations use the Lelantus' batch verification technique [100], performance results are fairly similar, with a difference of 9.5% on average across verification times in favour of the Rust implementation (parallelisation not enabled). These results are not surprising, as even the Lelantus paper states: "Our protocol works over any homomorphic group and presumably a RUST implementation over the Ristretto points may result in better performance.". Table 7.5 also highlights how significant multi-threading becomes for bigger batches: while turning on multi-threading for a batch size of 5 only gives a performance boost of 19.25%, for the bigger batch size of 1000, this is increased to 81.6%.

Batch size	Verifcation Time (ms)		Average cost per verification (ms)	
	Single core	Parallel	Single core	Parallel
5	1397	1128	279.4	225.6
10	1719	1181	171.9	118.1
50	4315	1672	86.3	33.4
100	8086	2299	80.8	22.9
500	38589	7639	77.2	15.3
1000	74298	13742	74.3	13.7

Table 7.5: Optimised Rust library's batch verification performance for set size 100000.

The improved `one-of-many-proofs` library is therefore generally more performant than its C++ counterpart, especially when using multi-threading. It is however impossible to draw a conclusion on whether it is the fastest implementation of One-out-of-Many proofs, as the benchmarking data for Lelantus comes from its 2019 paper and their implementation might have significantly improved since.

7.1.2 NetNotes protocol performance

Overall, as the NetNotes protocol is highly dependant on the membership proof, the optimisations performed in the previous sections were necessary to significantly improve its performance.

Figure 17 shows single proof verification times for different set sizes. As the One-out-of-Many proof is the main computational bottleneck for the protocol, these results are very similar to Table 7.2 and largely scale linearly to the set size.

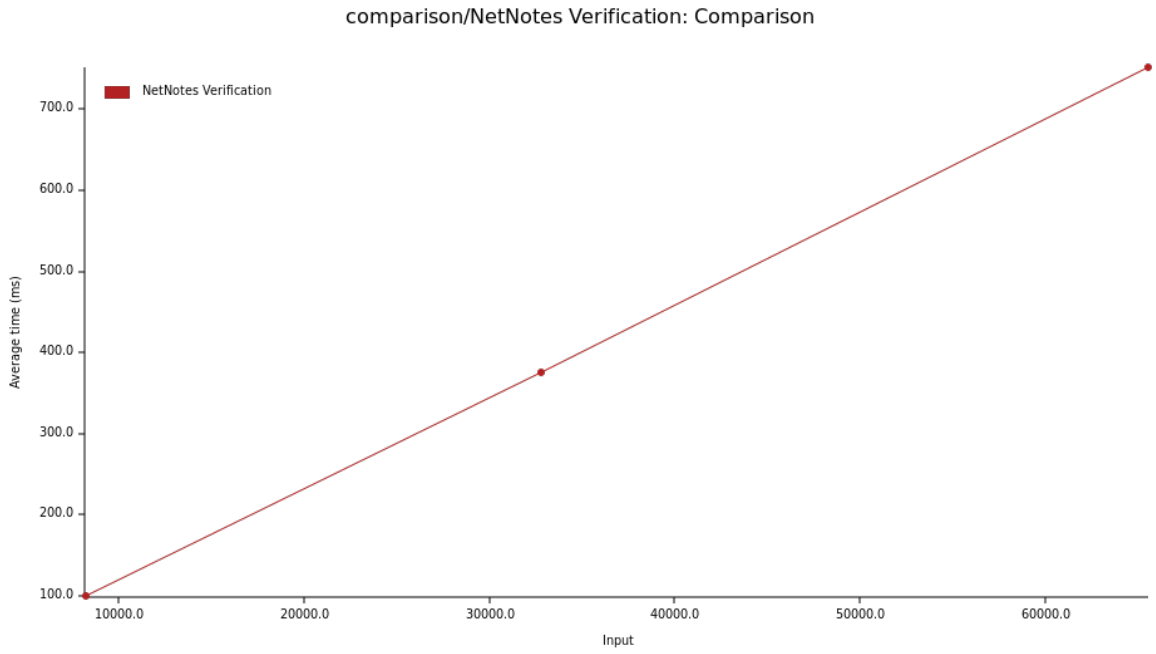


Figure 17: NetNotes verification times for set sizes of 8192, 32768 and 65536.

Figure 18 instead shows a violin plot for NetNotes' transactions generation times for various anonymity set sizes. These can be seen to scale logarithmically and overall show a wide distribution across different benchmarks.

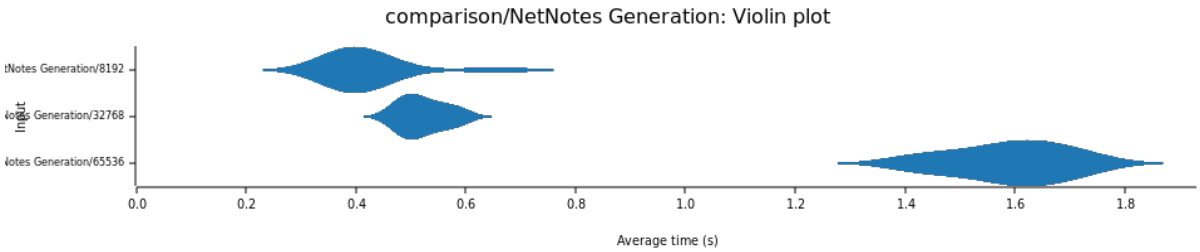


Figure 18: NetNotes generation times for set sizes of 8192, 32768 and 65536.

To offer a comparison between NetNotes and other protocols, the data from [100] was used and is shown in Table 7.6. Please note that some data might be outdated and is only meant to provide a rough comparison between NetNotes and other schemes.

Privacy Solution	Set Size	Proof Size (KB)	Proof Time (s)	Verification Time (ms)
Monero	16	2.1	1	47
Zerocash	2^{32}	0.3	1-20	8
Zerocoin	2^{13}	25	0.2	200
Lelantus_{16384}	2^{14}	1.5	1.2	12*
Lelantus_{65536}	2^{16}	1.5	5.2	35*
NetNotes_{8192}	2^{13}	2.0	0.4	99
NetNotes_{32768}	2^{15}	2.0	0.5	376
NetNotes_{65536}	2^{16}	2.0	1.5	750
Mimblewimble	#tx/block	0.2	0.3	0.08

Table 7.6: Comparison of different privacy solutions. Adapted from [100].

As can be seen, the protocol is competitive in terms of *anonymity set size*, *proof size* and *proof time*, but is lacking in *verification time*. Although at first it might appear that the protocol is much slower than Lelantus, it must be noted that Lelantus used the average cost of verifying a batch of 1000 proofs, rather a single verification. If the same assumption was made to calculate NetNotes's verification times for a batch of 10 transactions and an anonymity set of 32768, the average verification time would only result in 40ms. This is faster than Monero's implementation, whilst providing a lot more privacy given the much bigger anonymity set.

Due to the missing range proofs, the Mimblewimble implementation only requires to verify a single Schnorr signature for its spend scheme. Whilst remarkably fast, this also comes at the cost of not providing *untraceability*, which usually requires huge performance sacrifices to achieve. Lastly, the NetNotes proof size can be seen to be larger than Lelantus', despite its Rust implementation generating smaller proofs than its C++ counterpart. This is because NetNotes requires two extra Schnorr signatures, as well as an excess value, for its spend protocol.

All benchmark data for the NetNotes and Mimblewimble implementations can be found at [108].

7.2 Security

In the following section the security of the NetNotes protocol will be explored by discussing how common attacks are mitigated within the protocol.

7.2.1 Double-spending

As previously discussed, the most critical security property a cryptocurrency should possess is preventing double-spends. Mimblewimble achieves this by requiring users to provide a reference to an unspent output in the blockchain, which if validated can act as an input to a transaction.

Since the NetNotes scheme aims to hide which unspent output is being spent in a transaction, it is not possible to use the blockchain as a source of unspent outputs. Instead, the scheme has to rely on a set of "spent inputs", which keeps track of revealed commitments, to detect double-spends. NetNotes' defense against double-spending is therefore based on the assumption that for any output, a spender is only able to reveal a single valid commitment. If it was possible to reveal multiple valid commitments for the same coin, then it would become impossible for the network nodes to detect double-spends.

To ensure every output is associated with a single "serial number" (i.e. Pedersen commitment), the scheme leverages a second blinding factor s, used to obfuscate the output $v.G + r.H$ and commit to it. Knowledge of value s will be also required to successfully form a One-out-of-Many proof when a user attempts to spend the coin.

To prove that the commitment $v.G + r.H$ can be used as a serial number for commitment $v.G + r.H + s.J$ in the STXO set, recall how a spender needs to reveal a commitment $C_{in} = v.G+r.H$ paired with a One-out-of-Many proof π_{ooom} to be able to use an input for a transaction. More specifically, a valid proof π_{ooom} is required to prove ownership, and is formed on the set resulting from subtracting C_{in} from the STXO set.

Given that the One-out-of-Many-proof only allows to prove knowledge of a commitment to 0 in a set, the only way to obtain a valid commitment $Com_{hk}(0, s)$ in the resulting set is to cancel out the exact $v.G$ and $r.H$ from a single commitment within the set. Furthermore, thanks to the *binding* property of the commitment scheme, it is not possible to change the values v and r after the commitment has been added to the STXO set.

It is however possible to cheat this scheme by revealing multiple commitments in the form $C_{cheat} = v.G + r.H + s'.H$ for any $s' \in \mathbb{Z}_q$. When subtracting C_{cheat} from the STXO set, $v.G$ and $r.H$ will be cancelled out, allowing for a One-out-of-Many proof to be formed for any value $(s - s').H$. To prevent this attack, a Schnorr signature is needed to prove the input commitment provided, C_{in}, is not hiding any second blinding factor s, and is in fact of the form $v.G + r.H$ only. Given that the commitment is in such form, the value $s.H$ cannot now be modified during the subtraction, and therefore the *binding* property of Pedersen commitments is preserved.

In conclusion, taking into consideration that s cannot be modified during the spend protocol, only a single valid One-out-of-Many proof can be formed for a commitment $Com_{hk}(0, s)$. To derive such a commitment in the STXO set, the spender needs to reveal the commitment $C_{in} =$

$v.G + r.H$, such that values $v.G$ and $r.H$ are cancelled out for a commitment in the STXO set. Since there exists only one commitment C_{in} that will cancel out both values for a single commitment in the STXO set, C_{in} satisfies the definition of a valid "serial number" for an input. This commitment can used to ensure that no coin can be spent twice, by checking that its value is not already present in a set containing all previously revealed C_{in} commitments.

7.2.2 Theft of funds

The security of Pedersen commitments is based on the computational hardness of the discrete logarithm problem in the underlying group. This entails that given a value v for a commitment, it should be computationally unfeasible to find the blinding factor r used to construct the commitment. Brute force attacks are therefore unfeasible to try and claim a coin that is not owned by the user, especially considering the double blinded nature of NetNotes' outputs.

It is important to note however that Pedersen commitments are only computationally binding, meaning that it is possible for two commitments $C_a = v.G + r.H$ and $C_b = v'.G + r'.H$ to output the same value. This does not pose a problem in the context of double-spends, as the values of every individual factors are used separately, rather than together, to construct the spend proof.

There is however a case where a user might invalidate the spend proof of another user inadvertently. For instance, consider Alice's output $v.G + r.H + s.J$ and Bob's commitment $v'.G + r'.H + s'.J$, where $v.G + r.H = v'.G + r'.H$. If Alice were to reveal $v.G + r.H$ first to use as input for a transaction, then Bob would be unable to use $v'.G + r'.H$ as input for any future transaction, as it would appear already spent in the "spent inputs" set.

For this reason, it is important for a user to ensure that the chosen blinding factor r does not form a commitment $v.G + r.H$ colliding with any revealed past C_{in}'s. Not performing this check might result in a loss of funds if someone else invalidates the input first.

7.2.3 Undetectable inflation

In Mimblewimble, the whole blockchain can be easily validated as a single transaction by summing up all the UTXOs and comparing them to the known total supply of coins and the sum of transaction kernels (see equation 4.4).

In NetNotes however, due to the introduction of *input untraceability*, it is not possible to determine which outputs correspond to which inputs. Due to this uncertainty, it is also impossible to consider and validate the whole blockchain as a single transaction the same way that Mimblewimble can. In fact, any cryptocurrency implementing *confidential transactions* gives up the ability to audit its supply; indeed the only way left to ensure that no money is created out of thin air is to check that every historical transaction's range proof is valid.

To reason about if undetectable inflation is possible in NetNotes, consider that the only way to create money – aside from coinbase transactions – is during the spend protocol. The first thing to realise is that for the generalised outputs $v.G + r.H + s.J$ and inputs of the form $v.G + r.H$ the same security assumptions made for Mimblewimble hold. Namely, inflation is prevented by requiring a Schnorr signature proving that the difference between the outputs and inputs does not contain any leftover value $v.G$, and is instead in the form $r.H + s.J$.

It is also important to consider whether it is possible to cheat this scheme by passing inputs or outputs committed to different elliptic groups. In the case of the inputs, the additional requirement of a Schnorr signature for $v.G + r.H$ is meant to prevent this exact problem from occurring. In the case of the outputs, given that the inputs are provably of the form $v.G + r.H$, and given that the difference between them and the inputs must be a public key for $r.H + s.J$, it is also impossible for them to be in any form other than $v.G + r.H + s.J$.

Overall, the modified spend protocol and the addition of the One-out-of-Many proof have not changed the security assumptions of the scheme compared to Mimblewimble, and therefore the same security guarantees are provided. Despite maintaining it's security however, the scheme loses an efficient way to validate the entire blockchain compared to Mimblewimble, which makes joining the network a more expensive operation.

7.2.4 Linkability

It should be impossible for anyone but the participant to link a transaction input to a previous transaction output. In other words, an attacker given $Com(v, r)$ should not be able to link the revealed commitment to any past transaction output $Com(v, r, s)$ stored in the STXO set.

Take T_1 and T_2, where T_2 spends an input $C_{in} = v.G + r.H$ derived from the output $C_{out} = v.G + r.H + s.J$ of previous transaction T_1. An external observer could take the input to the transaction C_{in} and subtract it from every output in the STXO set. A single commitment should result in $s.J$, however he has no way of knowing which output in the STXO set this value belongs to, due to the *hiding* property of Pedersen commitments. Given that the value s is never revealed, the observer cannot calculate $s.J$ to find the matching output in the STXO set either.

Another case is a malicious participant in a transaction. For instance, say Alice wants to send malicious actor Bob some coins, all stored within a single output; during the transaction Bob would come to learn the value v he is receiving. However, even with Bob having knowledge of v, he still has no way of linking Alice's revealed commitment C_{in} to any double blinded commitment in the STXO set. The only way to take advantage of v would be to also have knowledge of blinding factors r and s to recreate (and therefore reveal) the original output. It can be concluded that the only way for anyone to break the untraceability property is to have knowledge of the blinding factor s, which due to never being revealed, is impossible to obtain in practice.

7.3 Comparison to Mimblewimble

As previously discussed, NetNotes sacrifices the convenience of easily verifying the blockchain validity. Most importantly, it loses ability to perform cut-through on spent outputs from the blockchain. These tradeoffs are significant, but are inevitable when introducing untraceability to any blockchain. In a sense, it was only possible to achieve *scalability* in Mimblwimble by sacrificing *security*, which undermines the privacy aspect of the protocol, given a chain is only as strong as its weakest link. As long as Mimblewimble is missing the *untraceability* property, it will be difficult for it to compete with other privacy coins such as Monero [1], which offer a more complete privacy solution.

7.4 Comparison to similar schemes

To highlight the strengths and weakness of the proposed scheme, a comparison to the two existing schemes trying to address the *untraceability* problem in Mimblewimble is made in this section.

7.4.1 Lelantus-MW

The Lelantus-MW protocol [88], is an adaptation of Aram Jibanyan's Lelantus scheme [95] to Mimblewimble.

NetNotes' spend proof is inspired by Lelantus-MW's spend proof, which also involves subtracting a Pedersen commitment from a list of double blinded commitments and using a One-out-of-Many proof to prove knowledge of a commitment in terms of a single generator only (rather than two). This improved version of Lelantus' spend proof allows for using the smaller and faster unmodified One-out-of-Many proofs protocol **10.1007/978-3-662-46803-6_9**.

Despite this, the two schemes are fundamentally different, as Lelantus-MW is designed for non-interactive transactions, while NetNotes is designed for interactive transactions. While in the case of Lelantus-MW a series of additional computations and proofs are required to generate a serial number s in a way that cannot be spent by the sender, the NetNotes scheme leverages the fact that the receiver can choose his own secret r and s during the transaction building process. NetNotes therefore follows more in the footsteps of Mimblewimble, where ownership is represented by knowledge of the two blinding factors r and s, while Lelantus-MW is more akin to Lelantus by focusing on creating an output for someone else and proving ownership through revealing a serial number paired with a digital signature.

Even with NetNotes having a smaller transaction size, similar verification time (slightly faster due to having to verify less proofs) and a simpler overall protocol, Lelantus-MW has the big advantage of being a non-interactive protocol. Despite such property being highly desirable for a cryptocurrency, interactive payments can offer some interesting advantages compared to non-interactive payments, such as:

- Every transaction comes with the assurance that the funds end up with the recipient able to spend them; i.e. no coins can be lost
- Sender and receiver can agree on a payment proof stating the purpose of payment
- Allows for easy PayJoins [109], which can be used to obfuscate direction of payment
- Having seamless migration to use of payment channels and 2nd layer solutions (such as Bitcoin's interactive Lightning Network [110])
- Allows auditing incoming transactions, which can be used to prevent spam and dusting attacks

7.4.2 Oscausi

Oscausi [87] is a variation of the Lelantus scheme [95], which runs on top of Mimblewimble. Similarly to Lelantus, the scheme is designed on a two-coin system, with a confidential Mimblewimble basecoin and a fully private shielded coin.

While Oscausi is defined in terms of Lelantus' *Minting* (burning coins to create shielded outputs), *Spending* (spending coins from a shielded pool) and *JoinSplit* (transactions creating mixed outputs), NetNotes is defined in terms of classic input and outputs and has been designed to be an inherent part of the Mimblewimble protocol. Because of this, NetNotes does not suffer from having to burn basecoins to redeem shielded coins before being able to spend them.
Furthermore, full privacy is achieved at all times due to being an inherent property of the protocol itself, rather than an optional feature. These advantages however do come at the cost of storage space, as Oscausi's Mimblewimble outputs can be pruned (but not the shielded outputs), while NetNotes' outputs cannot.

Oscausi spend proof is similar to Lelantus [95], where a second blinding factor s is used as serial number and is computed by $\mathscr{H}(q.G)$, where q is a randomly sampled scalar. This is then revealed with a a signature for q and subtracted from the set of shielded outputs to prove knowledge of a commitment in the form $Com(v, r)$ within the resulting set.

In terms of performance, due to using Lelantus' [95] modified One-out-of-Many proof, it should be less efficient than NetNotes, which uses the original, slightly faster One-out-of-Many proof by Groth and Kohlweiss [4].

CHAPTER 8

Conclusion

8.1 Overview

This dissertation has presented a novel privacy-preserving cryptocurrency protocol based on Mimblewimble. The scheme improves upon Mimblewimble's privacy guarantees by implementing the *untraceability* property, thus achieving the definition of *complete privacy*. This heightened privacy is however achieved at the cost of efficiency and storage requirements. Despite this trade-off, the novel protocol's performance remains competitive with other privacy-preserving protocols and achieves the original aim of this project.

Another contribution was also achieved during the work performed for this dissertation: improving the only-existing Rust implementation of One-out-of-Many proofs. The optimised library now attains 65% smaller proof sizes and up to 96% and 65% improved proof generation and verification times respectively.

8.2 Future Work

The main focus for future work should be directed towards improving the performance of the One-out-of-Many proof, as this is by far the most computationally expensive part of the protocol. For instance, an implementation of Many-Out-Of-Many proofs by Benjamin E. Diamond [111] could help prove the membership of multiple inputs (as is usually the case in a UTXO-based transaction) in a single proof, reducing storage and computation requirements. Another promising approach is to use Hierarchical One-out-of-Many Proofs [112], which can significantly reduce the generation time to $O(N + TlogT + MlogM)$ group exponentiation operations, compared to the current implementation's requirement of $O(NlogN)$.

The underlying Mimblewimble protocol could also benefit from further research, such as the exploration of a 2-phase protocol, as described in an RFC by David Burkett [113]. This could simplify transaction building by reducing the number of steps involved. Finally, BLS signatures [114] have long be considered a potential candidate for implementing kernel signatures in Mimblewimble. This would allow the kernel – currently the only data structure that cannot be pruned – to be aggregated, reducing blockchain storage requirements.

Regarding the improved Rust implementation of One-out-of-Many proofs presented in this work, better benchmarking should be performed against Lelantus' latest C++ implementation to determine which library is the currently faster implementation of One-out-of-Many proofs.

8.3 Review of Project Objectives

Nr	Description	Achieved	Chapter
1	Research and learn about the cryptography used in cryptocurrencies	✓	2
2	Learn about the current state of the art in privacy-preserving cryptocurrency solutions	✓	34
3	Design a novel cryptographic scheme that achieves "complete privacy" and is competitive in terms of performance	✓	5
4	Implement the novel protocol in a chosen programming language	✓	6
5	Perform benchmarking of the novel protocol to provide a performance comparison with other privacy-preserving schemes	✓	7
6	Analyse the security properties of the novel protocol and provide a general evaluation of the scheme's strengths and weakness compared to other privacy-preserving protocols	✓	7

Table 8.1: Review of project objectives.

As can be seen by Table 8.1, all of the objectives described in chapter 1 have been achieved. Furthermore, the original aim of creating a competitive cryptographic protocol enabling private transactions has been met, marking a successful completion of the project.

8.4 Final Remarks

Undoubtedly, this dissertation project has proven very challenging and a tremendous learning experience at the same time. Finally gaining an understanding of how a cryptocurrency works at a very low-level, as well as learning cryptography for the first time, was extremely interesting and was definitely the highlight of the project. The most significant takeaway would be that blockchain privacy is, in my opinion, an under-researched field which deserves more of a spotlight. I have also come to believe that zero-knowledge proofs are the key to achieve a lightweight and performant privacy scheme, despite their use of complicated maths which regrettably hurts their accessibility and auditability. As for my next steps, I am excited to continue researching the latest blockchain technology and try to keep up with this relatively new field. In conclusion, I am confident the knowledge gained and skills learnt throughout this project – as well as University in general – will serve me well in any of my future endeavours.

References

[1] *Monero - secure, private, untraceable*. [Online]. Available: https://getmonero.org (visited on 25/10/2022).

[2] E. Ben-Sasson, A. Chiesa, C. Garman *et al.*, 'Zerocash: Decentralized anonymous payments from bitcoin,' Tech. Rep., 2014. [Online]. Available: http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf.

[3] I. Bogatyy, 'Breaking mimblewimble privacy model,' *Medium - Dragonfly Research*, 2019. [Online]. Available: https://medium.com/dragonfly-research/breaking-mimblewimble-privacy-model-84bcd67bfe52.

[4] J. Groth and M. Kohlweiss, 'One-out-of-many proofs: Or how to leak a secret and spend a coin,' in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 253–280, ISBN: 978-3-662-46803-6.

[5] D. Chaum, 'Blind signatures for untraceable payments,' Department of Computer Science, University of California, Tech. Rep., 1982. [Online]. Available: http://blog.koehntopp.de/uploads/Chaum.BlindSigForPayment.1982.PDF.

[6] D. Chaum, *Ecash*. [Online]. Available: https://chaum.com/ecash (visited on 21/09/2022).

[7] A. Back, 'Hashcash - a denial of service counter-measure,' Tech. Rep., 2002. [Online]. Available: http://www.hashcash.org/hashcash.pdf.

[8] W. Dai, *B-money*, 1998. [Online]. Available: http://www.weidai.com/bmoney.txt (visited on 21/09/2022).

[9] N. Szabo, *Bit gold*, 2005. [Online]. Available: https://nakamotoinstitute.org/bit-gold/ (visited on 21/09/2022).

[10] S. Nakamoto, 'Bitcoin – a peer to peer electronic cash system.,' Tech. Rep., 2008. [Online]. Available: https://bitcoin.modeapp.com/bitcoin-white-paper.pdf.

[11] J. Howarth, 'How many cryptocurrencies are there in 2022?' *Exploding Topics*, 2022. [Online]. Available: https://explodingtopics.com/blog/number-of-cryptocurrencies (visited on 22/09/2022).

[12] W. Yakowicz, 'Startups helping the fbi catch bitcoin criminals,' *Inc.*, 2018. [Online]. Available: https://www.inc.com/will-yakowicz/startups-law-enforcement-agencies-catch-criminals-who-use-cryptocurrency.html (visited on 13/10/2022).

[13] T. Robinson, 'Bitcoin blockchain analysis and doj indictment of russian hackers,' *Elliptic*, 2018. [Online]. Available: https://www.elliptic.co/blog/doj-indictment-russian-hackers-blockchain-analysis (visited on 13/10/2022).

[14] N. van Saberhagen, 'Cryptonote v 2.0,' Tech. Rep., 2013. [Online]. Available: https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf.

[15] W. Diffie and M. Hellman, 'New directions in cryptography,' *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. DOI: 10.1109/TIT.1976.1055638.

[16] R. L. Rivest, A. Shamir and L. Adleman, 'A method for obtaining digital signatures and public-key cryptosystems,' *Commun. ACM*, vol. 21, no. 2, 120–126, 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342. [Online]. Available: https://doi.org/10.1145/359340.359342.

[17] N. Koblitz, *Elliptic curve cryptography*, 1985. [Online]. Available: https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf.

[18] V. S. Miller, 'Use of elliptic curves in cryptography,' in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426, ISBN: 978-3-540-39799-1.

[19] K. S. McCurley, 'The discrete logarithm problem,' Tech. Rep., 1990. [Online]. Available: https://www.mccurley.org/papers/dlog.pdf.

[20] C. P. Schnorr, 'Efficient signature generation by smart cards,' *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991. DOI: 10.1007/BF00196725. [Online]. Available: https://doi.org/10.1007/BF00196725.

[21] T. P. Pedersen, 'Non-interactive and information-theoretic secure verifiable secret sharing,' Aarhus University, Denmark, Tech. Rep., 1992. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf.

[22] A. V.Meier, 'The elgamal cryptosystem,' Tech. Rep., 2005. [Online]. Available: https://wwwmayr.in.tum.de/konferenzen/Jass05/courses/1/papers/meier_paper.pdf.

[23] S Goldwasser, S Micali and C Rackoff, 'The knowledge complexity of interactive proof-systems,' in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85, Providence, Rhode Island, USA: Association for Computing Machinery, 1985, 291–304, ISBN: 0897911512. DOI: 10.1145/22145.22178. [Online]. Available: https://doi.org/10.1145/22145.22178.

[24] A. Fiat and A. Shamir, 'How to prove yourself: Practical solutions to identification and signature problems,' in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194, ISBN: 978-3-540-47721-1.

[25] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille and G. Maxwell, 'Bulletproofs: Short proofs for confidential transactions and more,' in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[26] H. Chung, K. Han, C. Ju, M. Kim and J. H. Seo, 'Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger,' *IEEE Access*, vol. 10, pp. 42 081–42 096, 2022. DOI: 10.1109/ACCESS.2022.3167806.

[27] L. Eagen, *Bulletproofs++*, Cryptology ePrint Archive, Paper 2022/510, https://eprint.iacr.org/2022/510, 2022. [Online]. Available: https://eprint.iacr.org/2022/510.

[28] C. Yun, *Building on bulleproofs*, 2019. [Online]. Available: https://cathieyun.medium.com/building-on-bulletproofs-2faa58af0ba8 (visited on 21/04/2023).

[29] Dalek Cryprography team, *Bulletproof notes - inner product proof*, 2017. [Online]. Available: https://doc-internal.dalek.rs/bulletproofs/notes/inner_product_proof/index.html (visited on 16/04/2022).

[30] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth and C. Petit, 'Short accountable ring signatures based on ddh,' in *Computer Security – ESORICS 2015*, G. Pernul, P. Y A Ryan and E. Weippl, Eds., Cham: Springer International Publishing, 2015, pp. 243–265, ISBN: 978-3-319-24174-6.

[31] *Hyperledger – open source blockchain technologies*. [Online]. Available: https://www.hyperledger.org (visited on 24/10/2022).

[32] M. H. anad Richard Gendal Brown, 'Corda: A distruted ledger,' Tech. Rep., 2019. [Online]. Available: https://www.r3.com/blog/corda-technical-whitepaper/ (visited on 24/10/2022).

[33] C. Lee, *Lite coin white paper*, 2011. [Online]. Available: https://whitepaper.io/document/683/litecoin-whitepaper.

[34] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform.* 2014. [Online]. Available: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[35] I. Grigg, *Eos: An introduction*, 2018. [Online]. Available: https://www.iang.org/papers/EOS_An_Introduction-BLACK-EDITION.pdf.

[36] J. W. Matonis, *Digital cash and monetary freedom*, 1995. [Online]. Available: https://oz.stern.nyu.edu/fall99/readings/digicash/#FN%205.

[37] T. Okamoto and K. Ohta, *Universal electronic cash*, 1992. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-46766-1_27.pdf.

[38] A. Gaihre, Y. Luo and H. Liu, *Do bitcoin users really care about anonymity? an analysis of the bitcoin transaction graph*, 2018. [Online]. Available: https://www.uml.edu/docs/Bitcoin_tcm18-302979.pdf.

[39] E. Duffield and D. Diaz, 'Dash: A privacy-centric crypto-currency,' Tech. Rep., 2014. [Online]. Available: https://www.exodus.com/assets/docs/dash-whitepaper.pdf.

[40] J. Harvey, *Why cryptocurrencies want privacy: A review of political motivations and branding expressed in "privacy coin" whitepapers*, 2019. [Online]. Available: https://dspace.stir.ac.uk/bitstream/1893/30048/1/Why%20cryptocurrencies%20want%20privacy%20-%2015h%20July.pdf.

[41] O. Adejumo, 'Report: Crypto ransomware payments in 2021 was over $600 million,' *CryptoSlate*, 2022. [Online]. Available: https://cryptoslate.com/report-crypto-ransomware-payments-in-2021-was-over-600-million/.

[42] G. Rodgers, 'Consumer wants: Privacy transparency, online security, better customer experience,' 2022. [Online]. Available: https://www.cmswire.com/customer-experience/consumer-wants-privacy-transparency-online-security-better-customer-experience/ (visited on 12/12/2022).

[43] T. Wright, 'Tornado cash dao goes down without explanation following vote on treasury funds,' *Cointelegraph*, 2022. [Online]. Available: https://cointelegraph.com/news/tornado-cash-dao-goes-down-without-explanation-following-vote-on-treasury-funds (visited on 12/12/2022).

[44] A. Singer, 'Regulators dial up the heat: Dash, zec and monero reach boiling point?' *Cointelgraph*, 2021. [Online]. Available: https://cointelegraph.com/news/regulators-dial-up-the-heat-dash-zec-and-monero-reach-boiling-point (visited on 10/12/2022).

[45] S. Malwa, 'Crypto exchange huobi to delist 7 privacy coins, including zcash, monero,' *CoinDesk*, 2022. [Online]. Available: https://www.coindesk.com/business/2022/09/12/crypto-exchange-huobi-to-delist-7-privacy-coins-including-zcash-monero/ (visited on 10/12/2022).

[46] P. Jha, 'Regulations and exchange delistings put future of private cryptocurrencies in doubt,' *Cointelgraph*, 2022. [Online]. Available: https://cointelegraph.com/news/regulations-and-exchange-delistings-put-future-of-private-cryptocurrencies-in-doubt (visited on 10/12/2022).

[47] 'Zcash - compliance,' [Online]. Available: https://z.cash/compliance/ (visited on 12/12/2022).

[48] P. Thompson, 'Exchange blacklist prevents twitter hacker from stealing more btc,' *CoinGeek*, 2020. [Online]. Available: https://coingeek.com/exchange-blacklist-prevents-twitter-hacker-from-stealing-more-btc/ (visited on 22/09/2022).

[49] W. Jones, 'Uniswap blacklists 253 crypto wallets linked to illicit activities,' *crypto.news*, 2022. [Online]. Available: https://crypto.news/uniswap-blacklists-253-crypto-wallets-linked-to-illicit-activities/ (visited on 22/09/2022).

[50] S. for Privacy, 'Bitcoin's fungibility graveyard,' *sethprivacy.com*, 2021. [Online]. Available: https://sethforprivacy.com/posts/fungibility-graveyard/ (visited on 22/09/2022).

[51] S. Meiklejohn, M. Pomarole, G. Jordan *et al.*, 'A fistful of bitcoins: Characterizing payments among men with no names,' *Commun. ACM*, vol. 59, no. 4, 86–93, 2016, ISSN: 0001-0782. DOI: 10.1145/2896384. [Online]. Available: https://doi.org/10.1145/2896384.

[52] Y. Zhang, J. Wang and J. Luo, 'Heuristic-based address clustering in bitcoin,' *IEEE Access*, vol. 8, pp. 210 582–210 591, Jan. 2020. DOI: 10.1109/ACCESS.2020.3039570.

[53] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer and S. Capkun, 'Evaluating user privacy in bitcoin,' in *Financial Cryptography and Data Security*, A.-R. Sadeghi, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 34–51, ISBN: 978-3-642-39884-1.

[54] Q. Feng, D. He, S. Zeadally, M. K. Khan and N. Kumar, 'A survey on privacy protection in blockchain system,' *Journal of Network and Computer Applications*, vol. 126, pp. 45–58, 2019, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.10.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518303485.

[55] N. Tovanich and R. Cazabet, 'Pattern analysis of money flows in the bitcoin blockchain,' in *Complex Networks and Their Applications XI*, H. Cherifi, R. N. Mantegna, L. M. Rocha, C. Cherifi and S. Micciché, Eds., Cham: Springer International Publishing, 2023, pp. 443–455, ISBN: 978-3-031-21127-0.

[56] T.-H. Chang and D. Svetinovic, 'Improving bitcoin ownership identification using transaction patterns analysis,' *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 9–20, 2020. DOI: 10.1109/TSMC.2018.2867497.

[57] M. Fleder, M. S. Kester and S. Pillai, *Bitcoin transaction graph analysis*, 2015. arXiv: 1502.01657 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1502.01657.

[58] G. Maxwell, *Coinjoin: Bitcoin privacy for the real world*, 2013. [Online]. Available: https://bitcointalk.org/?topic=279249 (visited on 12/10/2022).

[59] T. Ruffing, P. Moreno-Sanchez and A. Kate, 'Coinshuffle: Practical decentralized coin mixing for bitcoin,' in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds., Cham: Springer International Publishing, 2014, pp. 345–364, ISBN: 978-3-319-11212-1.

[60] T. Ruffing, P. Moreno-Sanchez and A. Kate, *P2p mixing and unlinkable bitcoin transactions*, Cryptology ePrint Archive, Paper 2016/824, https://eprint.iacr.org/2016/824, 2016. [Online]. Available: https://eprint.iacr.org/2016/824.

[61] *Decred - money evolved.* [Online]. Available: https://decred.org (visited on 26/10/2022).

[62] Ádám Ficsór, Y. Kogman, L. Ontivero and I. A. Seres, *Wabisabi: Centrally coordinated coinjoins with variable amounts*, Cryptology ePrint Archive, Paper 2021/206, 2021. [Online]. Available: https://eprint.iacr.org/2021/206.

[63] W. Suberg, *Binance returns frozen btc after user 'promises' not to use coinjoin*, 2019. [Online]. Available: https://cointelegraph.com/news/binance-returns-frozen-btc-after-user-promises-not-to-use-coinjoin.

[64] K. Redman, *Another crypto exchange discourages the use of bitcoin mixing services*, 2020. [Online]. Available: https://news.bitcoin.com/another-crypto-exchange-discourages-the-use-of-bitcoin-mixing-services/.

[65] R. L. Rivest, A. Shamir and Y. Tauman, 'How to leak a secret,' in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565, ISBN: 978-3-540-45682-7.

[66] A. Kumar, C. Fischer, S. Tople and P. Saxena, 'A traceability analysis of monero's blockchain,' in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann and E. Snekkenes, Eds., Cham: Springer International Publishing, 2017, pp. 153–173, ISBN: 978-3-319-66399-9.

[67] S. Noether, *Ring signature confidential transactions for monero*, Cryptology ePrint Archive, Paper 2015/1098, https://eprint.iacr.org/2015/1098, 2015. [Online]. Available: https://eprint.iacr.org/2015/1098.

[68] M. Möser, K. Soska, E. Heilman *et al.*, *An empirical analysis of traceability in the monero blockchain*, 2018. arXiv: 1704.04299 [cs.CR].

[69] jbernman, *Wallet2 decoy selection algorithm ignores very recent outputs.* [Online]. Available: https://github.com/mone,ro-project/monero/issues/7807 (visited on 22/03/2023).

[70] Rucknium, *Rucknium-ospead-fortifying-monero-against-statistical-attack.md.* [Online]. Available: https://repo.getmonero.org/monero-project/ccs-proposals/-/blob/c2cb28f099eeb87cd07cbc0697bda924dc552248/Rucknium-OSPEAD-Fortifying-Monero-Against-Statistical-Attack.md (visited on 22/03/2023).

[71] A. Poelstra, *Mimblewimble.* [Online]. Available: https://web.archive.org/web/20190304165553/https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[72] *Grin - privacy-preserving cryptocurrency.* [Online]. Available: https://grin.mw (visited on 12/10/2022).

[73] *Beam confidential cryptocurrency and defi platform.* [Online]. Available: https://beam.mw (visited on 11/10/2022).

[74] G. Fanti, S. B. Venkatakrishnan, S. Bakshi *et al.*, 'Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees,' *CoRR*, vol. abs/1805.11060, 2018. arXiv: 1805.11060. [Online]. Available: http://arxiv.org/abs/1805.11060.

[75] *Firo - privacy-preserving cryptocurrency.* [Online]. Available: https://firo.org (visited on 25/10/2022).

[76] I. Miers, C. Garman, M. Green and A. D. Rubin, 'Zerocoin: Anonymous distributed e-cash from bitcoin,' in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 397–411. DOI: 10.1109/SP.2013.34.

[77] A. Jivanyan and A. Feickert, *Lelantus spark: Secure and flexible private transactions*, Cryptology ePrint Archive, Paper 2021/1173, https://eprint.iacr.org/2021/1173, 2021. [Online]. Available: https://eprint.iacr.org/2021/1173.

[78] *Zcash: Privacy-protecting digital currency.* [Online]. Available: https://z.cash (visited on 25/10/2022).

[79] J. Groth, 'On the size of pairing-based non-interactive arguments,' in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326, ISBN: 978-3-662-49896-5.

[80] M. Bellare and A. Palacio, 'The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols,' in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 273–289, ISBN: 978-3-540-28628-8.

[81] R. Hackett, 'Zcash discloses vulnerability that could have allowed "Infinite Counterfeit" cryptocurrency,' 2019. [Online]. Available: https://web.archive.org/web/20201107223805/https://fortune.com/2019/02/05/zcash-vulnerability-cryptocurrency/ (visited on 25/10/2022).

[82] Z. Wilcox and T. Hornby, 'Fixing vulnerabilities in the zcash protocol,' 2016. [Online]. Available: https://electriccoin.co/blog/fixing-zcash-vulns/ (visited on 25/10/2022).

[83] D. E. Hopwood, J. Grigg, S. Bowe, K. Nuttycombe and Y. T. Lai, *Orchard shielded protocol*, 2021. [Online]. Available: https://zips.z.cash/zip-0224.

[84] S. Bowe, J. Grigg and D. Hopwood, *Recursive proof composition without a trusted setup*, Cryptology ePrint Archive, Paper 2019/1021, 2019. [Online]. Available: https://eprint.iacr.org/2019/1021.

[85] A. Gabizon, Z. J. Williamson and O. Ciobotaru, *Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge*, Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953.

[86] G. Maxwell, *Confidential transactions*, 2015. [Online]. Available: https://web.archive.org/web/20190326081546/https://people.xiph.org/~greg/confidential_values.txt (visited on 14/10/2022).

[87] L. Herskind, P. Katsikouli and N. Dragoni, 'Oscausi-practical private electronic cash from lelantus and mimblewimble,' English, *Journal of Internet Services and Information Security*, vol. 10, no. 2, pp. 16–34, May 2020, ISSN: 2182-2069. DOI: 10.22667/JISIS.2020.05.31.016.

[88] P. Chaidos and V. Gelfer, *Lelantus-mw*, 2020. [Online]. Available: https://docs.beam.mw/Lelantus-MW.pdf.

[89] *Rust programming language*. [Online]. Available: https://www.rust-lang.org (visited on 12/02/2023).

[90] valdok, *Beam - cryprographic primitives*. [Online]. Available: https://beam.mw/beampedia-item/elliptic-curve-cryptography (visited on 12/02/2023).

[91] Dalek Cryptography team, *Curve25519-dalek*. [Online]. Available: https://github.com/dalek-cryptography/curve25519-dalek (visited on 12/02/2023).

[92] T. Arcieri, *Elliptic curve libraries comparison*. [Online]. Available: https://twitter.com/bascule/status/1320183684935290882 (visited on 12/02/2023).

[93] T. Perrin, *[curves] cryptonote and equivalent points*, 2017. [Online]. Available: https://moderncrypto.org/mail-archive/curves/2017/000898.html (visited on 12/02/2023).

[94] BLAKE3 team, *Blake3*. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3 (visited on 12/02/2023).

[95] A. Jivanyan, *Lelantus: A new design for anonymous and confidential cryptocurrencies*, 2019. [Online]. Available: https://eprint.iacr.org/2019/373.pdf.

[96] Dalek Cryptography team, *Bulletproofs*. [Online]. Available: https://github.com/dalek-cryptography/bulletproofs (visited on 12/02/2023).

[97] Rust open-source community, *Lazy-static.rs*. [Online]. Available: https://github.com/rust-lang-nursery/lazy-static.rs (visited on 12/02/2023).

[98] Cargodog, *Efficient one-out-of-many proofs using gray codes*, 2020. [Online]. Available: https://github.com/cargodog/papers/blob/master/efficient-one-out-of-many-proofs-using-gray-codes/rendered.pdf (visited on 08/12/2022).

[99] cargodog, *One-of-many proofs*. [Online]. Available: https://github.com/cargodog/one-of-many-proofs (visited on 12/02/2023).

[100] A. Jivanyan, *Lelantus : Towards confidentiality and anonymity of blockchain transactions from standard assumptions*, 2019. [Online]. Available: https://allquantor.at/blockchainbib/pdf/jivanyan2019lelantus.pdf.

[101] cargodog, *Issues - cargodog/one-of-many-proofs*. [Online]. Available: https://github.com/cargodog/one-of-many-proofs/issues (visited on 08/12/2022).

[102] Rayon Contributors, *Rayon*. [Online]. Available: https://docs.rs/rayon/latest/rayon/ (visited on 12/02/2023).

[103] Rust open-source community, *Criterion.rs*. [Online]. Available: https://github.com/bheisler/criterion.rs (visited on 12/02/2023).

[104] *Netnotes repository*. [Online]. Available: https://github.com/endrizzimarco/NetNotes (visited on 12/02/2023).

[105] *Criterion netnotes benchmarks*. [Online]. Available: https://endrizzimarco.github.io/NetNotes/report/ (visited on 12/02/2023).

[106] Firo Team, *Firo*. [Online]. Available: https://github.com/firoorg/firo (visited on 12/02/2023).

[107] Bitcoin Core Developers, *Secp256k1*. [Online]. Available: https://github.com/bitcoin-core/secp256k1/tree/master (visited on 12/02/2023).

[108] *Benchmark results fot netnotes*. [Online]. Available: https://endrizzimarco.github.io/NetNotes/report/ (visited on 12/02/2023).

[109] rhavar, *[bitcoin-dev] bustapay bip :: A practical sender/receiver coinjoin protocol*, 2018. [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-August/016340.html (visited on 16/04/2022).

[110] J. Poon and T. Dryja, 'The bitcoin lightning network: Scalable off-chain instant payments,' Tech. Rep., 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf.

[111] B. E. Diamond, *Many-out-of-many proofs and applications to anonymous zether*, Cryptology ePrint Archive, Paper 2020/293, https://eprint.iacr.org/2020/293, 2020. DOI: 10.1109/SP40001.2021.00026. [Online]. Available: https://eprint.iacr.org/2020/293.

[112] A. Jivanyan and T. Mamikonyan, 'Hierarchical one-out-of-many proofs with applications to blockchain privacy and ring signatures,' in *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*, 2020, pp. 74–81. DOI: 10.1109/AsiaJCIS50894.2020.00023.

[113] D. Burkett, *Eliminate finalize step*, 2020. [Online]. Available: https://github.com/DavidBurkett/grin-rfcs/blob/eliminate_finalize/text/0000-eliminate-finalize.md (visited on 24/12/2022).

[114] D. Boneh, B. Lynn and H. Shacham, 'Short signatures from the weil pairing,' in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 514–532, ISBN: 978-3-540-45682-7.

[115] legislation.gov.uk, *Computer misuse act 1990*, 1990. [Online]. Available: https://www.legislation.gov.uk/ukpga/1990/18/contents (visited on 16/03/2023).

[116] legislation.gov.uk, *Data protection act 2018*, 2018. [Online]. Available: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted (visited on 16/03/2023).

[117] legislation.gov.uk, *Copyright, designs and patents act 1988*, 1988. [Online]. Available: https://www.legislation.gov.uk/ukpga/1988/48/contents (visited on 16/03/2023).

APPENDIX A

Statement of Ethics

The following sections will discuss the ethical implications of this dissertation project by using *basic ethics principles*, as well as the *BCS Code of Conduct*, as a framework.

Basic Principles

Do no harm

Do no harm is a principle that should be considered when designing and implementing technology. It states that applications should be safe and secure, and that in general no harm should occur to individuals or society.

Firstly, as the project does not involve human subject, no risk of harming individuals has been identified. It should be however noted that the definition of *harm* is quite general, and could also be defined as infringing intellectual property or interacting with technology in an unintended way (e.g. hacking). To address the aforementioned risks, legislation such as the 1990's Computer Misuses Act [115] and the Data Protection Act [116] exists to protect citizens. These two acts have been taken into consideration for the project, however, as the dissertation only involves the creation of a cryptographic protocol, no external data has been saved and no external system has been accessed. As such, these two legislation have not been identified as relevant to the project.

The only legislation deemed applicable for this project has been determined to be the Copyright, Designs and Patents Act of 1998 [117], which is relevant in the context of external code libraries used during the implementation of the protocol. To ensure compliance with this law, all Rust libraries utilised in the project were checked to ensure they possessed open-source licenses (as evidenced by section 6.1) and did not infringe on any intellectual property rights. However, it

should also be noted that every library published to Rust's package manager, `cargo`, must be open-source. As all third-party libraries used for this project were installed with `cargo`, this ensures that the CDPA has certainly been followed for the project.

Furthermore, since the `one-of-many-proofs` library is under the MIT license, the forked repository has also been ensured to be permissioned under the same license in order to comply with its notice. The repository for the NetNotes source code was additionally licensed under the open-source MIT license.

Finally, the developed protocol implementation is not able to directly bring any harm to humans. However, it is worth noting that harm could be indirectly caused by implementing the unaudited software into a safety-critical application, such as a cryptocurrency. To mitigate this risk, a usage warning has been included in the `README.md` of the NetNotes repository.

Informed Consent

Informed consent is a principle commonly used to maintain ethical practices in studies involving human subjects.

As the project consists of creating a cryptographic protocol and did not involve any human subjects, the principle of informed consent has not been deemed applicable in this case.

Data confidentiality

Data confidentiality is an ethical principle that refers to the obligation to protect sensitive and personal information.

As the project solely focuses on designing and developing a cryptographic protocol, no personal information was stored or processed, therefore no discernible issues concerning data confidentiality were identified.

Social responsibility

Social responsibility is an ethics principle stating that individuals must act in the best interest of society.

Based on the belief that privacy is a fundamental human right, it could be argued that the protocol contributes to society by way of providing users with the possibility of achieving private cryptocurrency transactions. While the protocol itself cannot cause any harm, there is always a possibility that it could be misused for unlawful activities if implemented and used in a private cryptocurrency. It should be noted however that such instances would fall beyond the scope of this project, which only intends to offer a cryptographic protocol.

BCS Code of Conduct

Computer Science students at the University of Surrey are among the British Computer Society members who follow ethical principles set by the BCS Code of Conduct. These principles aim to maintain high standards of professional conduct, and as such, the upcoming section has been dedicated to a discussion of how BCS principles were followed for this project.

Public Interest

Principle	Relevant	Actions/Comments
Have due regard for public health, privacy, security and wellbeing of others and the environment	✓	The primary aim of this project has been to design a protocol enhancing privacy of users in a cryptocurrency
Have due regard for the legitimate rights of third parties	✓	
Conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement	×	
Promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.	✓	This dissertation was written with the assumption that no prior knowledge was necessary, to make it as accessible as possible

Professional Competence and Integrity

Principle	Relevant	Actions/Comments
Only undertake to do work or provide a service that is within your professional competence	×	
NOT claim any level of competence that you do not possess	✓	All external sources used in this dissertation have been properly cited using the IEEE referencing style
Develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field	✓	Covered in the "Do no harm" section
Ensure that you have the knowledge and understanding of legislation and that you comply with such legislation, in carrying out your professional responsibilities	×	
Respect and value alternative viewpoints and seek, accept and offer honest criticisms of work	×	
Avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction	×	
Reject and will not make any offer of bribery or unethical inducement	×	

Duty to the Profession

Principle	Relevant	Actions/Comments
Accept your personal duty to uphold the reputation of the profession and not take any action which could bring the profession into disrepute	×	
Seek to improve professional standards through participation in their development, use and enforcement	✓	This could be interpreted as moving the field of cryptography forward, which could be argued was achieved by developing a novel protocol for private cryptocurrency transactions
Uphold the reputation and good standing of BCS, The Chartered Institute for IT	×	
Act with integrity and respect in your professional relationships with all members of BCS and with members of other professions with whom you work in a professional capacity	✓	Have acted with integrity and respect during my supervisor and interim meetings
Encourage and support fellow members in their professional development.	×	

Duty to relevant Authority

Principle	Relevant	Actions/Comments
Carry out your professional responsibilities with due care and diligence in accordance with the relevant authority's requirements while exercising your professional judgement at all times	×	
Seek to avoid any situation that may give rise to a conflict of interest between you and your relevant authority;	×	
Accept professional responsibility for your work and for the work of colleagues who are defined in a given context as working under your supervision	×	
NOT disclose or authorise to be disclosed, or use for personal gain or to benefit a third party, confidential information except with the permission of your relevant authority, or as required by legislation	×	
NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information), or take advantage of the lack of relevant knowledge or inexperience of others	×	

APPENDIX B

SAGE Form

The following pages contain the Self-Assessment for Governance and Ethics form *(SAGE)* that was submitted to the University as part of the ethical review process.

SAGE-HDR (v3.8 24/04/23)

Response ID	Completion date
1046015-1045997-109804311	28 Apr 2023, 10:48 (BST)

1	**Applicant Name**	Marco Endrizzi
1.a	**University of Surrey email address**	me00531@surrey.ac.uk
1.b	**Level of research**	Undergraduate
1.b.i	**Please enter your University of Surrey supervisor's name. If you have more than one supervisor, enter the details of the individual who will check this submission.**	Liqun Chen
1.b.ii	**Please enter your supervisor's University of Surrey email address. If you have more than one supervisor, enter the details of the supervisor who will check this submission.**	liqun.chen@surrey.ac.uk
1.c	**School or Department**	Computer Science
1.d	**Faculty**	FEPS - Faculty of Engineering and Physical Sciences

2	**Project title**	NetNotes: adding untraceability to Mimblewimble

3	**Please enter a brief summary of your project and its methodology in 250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview of the aims and objectives of your research.**	The aim of this project is to enhance the MimbleWimble cryptographic protocol and develop a new cryptographic protocol that enables completely confidential digital transactions. The objectives are: 1. Research and learn about the cryptography used in cryptocurrencies, with a focus on privacy-preserving schemes 2. Investigate the current state of the art in privacy-preserving cryptocurrencies 3. Design and implement a novel cryptographic scheme that achieves complete privacy and is competitive in terms of performance 4. Provide a performance comparison with other privacy-preserving protocols 5. Provide a formal security analysis for the protocol and general evaluation of the scheme's strengths and weakness compared to other privacy-preserving protocols

4	**Are you planning to join on to an existing Standard Study Protocol (SSP)? SSPs are overarching pre-approved protocols that can be used by multiple researchers investigating a similar topic area using identical methodologies. Please note, SSPs are only being used by 3 schools currently and cannot be used by other schools. Using an SSP requires permission and sign-off from the SSP owner**	NO

5	**Are you making an amendment to a project with a current University of Surrey favourable ethical opinion or approval in place?**	NO

6	**Does your research involve any animals, animal data or animal derived tissue, including cell lines?**	NO

8	**Does your project involve human participants (including human data and/or any human tissue*)?**	NO

9	**Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), military facilities, closed online forums, private social media pages etc. This also includes using University mailing lists (admin permission). If you are unsure, please contact ethics@surrey.ac.uk.**	NO

10	**Does your project involve any type of human tissue research? This includes Human Tissue Authority (HTA) relevant, or non-relevant tissue (e.g. non-cellular such as plasma or serum), any genetic material, samples that have been previously collected, samples being collected directly from the donor or obtained from another researcher, organisation or commercial source.**	NO

11	**Does your research involve exposure of participants to any hazardous materials e.g. chemicals, pathogens, biological agents or does it involve any activities or locations that may pose a risk of harm to the researcher or participant?**	NO

12	**Will you be importing or exporting any samples (including human, animal, plant or microbial/pathogen samples) to or from the UK?**	NO

13	**Will any participant visits be taking place in the Clinical Research Building (CRB)? (involving clinical procedures; if only visiting the CRB to collect/drop-off equipment or to meet with the research team (i.e. for informed consent/discussion) select 'NO').**	NO

14	**Will you be working with any collaborators or third parties to deliver any aspect of the research project?**	NO

15	**Are you conducting a service evaluation or an audit? Or using data from a service evaluation or audit?**	NO

16	**Does your funder, collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?**	NO

17	**Does your research involve accessing students' results or performance data? For example, accessing SITS data.**	NO

18	**Will ANY research activity take place outside of the UK?**	NO

19	**Are you undertaking security-sensitive research, as defined in the text below?**	NO

20	**Does your project require the processing of special category1 data?**	NO

21	**Have you selected YES to one or more of the above governance risk questions on this page (Q10-Q20)?**	NO

22	**Does your project process personal data2? Processing covers any activity performed with personal data, whether digitally or using other formats, and includes contacting, collecting, recording, organising, viewing, structuring, storing, adapting, transferring, altering, retrieving, consulting, marketing, using, disclosing, transmitting, communicating, disseminating, making available, aligning, analysing, combining, restricting, erasing, archiving, destroying.**	NO

23	**Are you using a platform, system or server external to the University approved platforms (Outside of Microsoft Office programs, Sharepoint, OneDrive Qualtrics, REDCap, JISC online surveys (BOS) and Gorilla)**	NO

24	**Does your research involve any of the above statements? If yes, your study may require external ethical review or regulatory approval**	NO

25	**Does your research involve any of the above? If yes, your study may require external ethical review or regulatory approval**	NO

26	**Does your project require ethics review from another institution? (For example: collaborative research with the NHS REC, the Ministry of Defence, the Ministry of Justice and/or other universities in the UK or abroad)**	NO

27	**Does your research involve any of the following individuals or higher-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research. Please note: the UEC reviewers may deem the nature of the research of certain high risk projects unsuitable to be undertaken by undergraduate students**	NOT APPLICABLE - none of the above high-risk options apply to my research.

28	**Does your research involve any of the following individuals or medium-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research.**	NOT APPLICABLE - none of the above medium-risk options apply to my research.

29	**Does your research involve any of the following individuals or lower-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research.**	NOT APPLICABLE - none of the above lower-risk options apply to my research.

30	**Declarations**	• I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these. • I confirm that I have provided accurate and complete information regarding my research project • I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies • I understand that if my answers to this form have indicated that I must submit an ethics and governance application, that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies. • I understand that if I have selected 'YES' on any governance risk questions and/or have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application (EGA) for review before conducting any research. If I have NOT selected any governance risks or selected any of the higher, medium or lower ethical risk criteria, I understand I can proceed with my research without review and

		acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance.

31	**If I am conducting research as a student:**	• I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct. • I confirm that if I am handling any information that can identify people, such as names, email addresses or audio/video recordings and images, I will adhere to the security requirements set out in the relevant Data Protection Policy