Contents

Chapter 1

Introduction

1.1. Cryptocurrencies

If the question **'What is a cryptocurrency?'** would have arised six years ago, only a few people would have known the answer. But with time, cryptocurrencies have seen a huge rise in popularity, they are getting more and more recognition, and they are certainly on their way to become widely used in many parts of our lives. **A cryptocurrency** is a digital currency that is secured by cryptography, thus making cryptocurrencies nearly impossible to counterfeit or double-spend. [1] Almost every crypto has as a root a decentralized network based on blockchain technology — a network that is distributed across a large number of computers. This decentralized structure allows them to exist outside the control of governments and central authorities.

1.2. Monero cryptocurrency

While Bitcoin is the state-of-the-art coin, most people tend to forget about the pros and improvements other cryptocurrencies bring.

Monero is a cryptocurrency project, widely known for its privacy-oriented features. Its blockchain is said to be opaque - meaning that the transaction details, like the identity of senders and recipients, and the amount of every transaction. With Monero anyone can send or broadcast transactions, but no outside observer can tell the source, amount, or destination. In Bitcoin the transaction history of each bitcoin is recorded on the blockchain, whereas in Monero the definition of untraceability steps up. [2]

1.3. My task

My task throughout the Tutored Research Project was to study and examine the main concepts regarding the Monero project. The mathematics underlying in cryptography is a key concept, thus the understanding of **modular arithmetic** is neccesary.

Monero uses a **public key cryptography with Elliptic Curves**. Throughout my project I had to examine this concept: had to be able to generate public-private key pairs, also its usage for different type of cryptographic schemes in Python.

In order to achieve privacy of the sender and receivers, so called **Ring Signatures** (Ring Signature, Linkable Ring Signature, Borromean Ring Signature) are being schemed in Monero, thus my assignment was to examine these thoroughly, and for a deeper understanding implement them in Python over time.

Chapter 2

Background and concepts

Before I get deeper into more complex parts of Monero, a few basic concepts and definitions have to be underlined to understand their usage in later sections.

2.1. Modular arithmetic

Cryptography is based on heavy mathematics, and Monero is not an exception: the concept of modular arithmetic plays a huge role in cryptography.
Modular arithmetic is basically doing different operations not on a line, as its usually done, but on a circle – the values always stay less than a fixed number, called the modulus. [3] The modulus operation is denoted 'mod'. In cryptography only the positive modulus part is important, which always returns a positive integer.
The number **a** is equivalent (congruent) to the number **b** modulo **n**, expressed by the equation below, if **a** differs from **b** by an exact multiple of n. [4] [5]

$$\mathrm{a} \equiv b \pmod{n}$$

thus

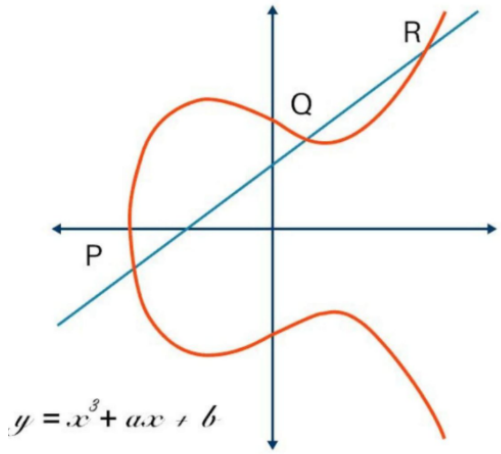
$$a = x * n + b \tag{2.1}$$

Several operations can be done this way: modular addition, modular multiplication, modular multiplicative inverse as well as solving modular equations. [6]

2.2. Elliptic curve cryptography

It can be said that public-key cryptography is based on the difficulty of certain mathematical problems. Early public-key systems based their security on the assumption that

it is difficult to factor a large integer composed of two or more large prime factors. [7] On the other hand, the root concept of elliptic curve cryptography is that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point (G) is computationally infeasible, therefore a system based on ellipic curves creates keys that are more difficult to crack. As a result elliptic curves proved to be a great solution for public-key system and became widely used in different projects. **nicksul**



[8]

An **elliptic curve** is a plane curve over a finite field. The concept "over finite fields" simply means that curve points are taken modulo some number. With the uppermentioned modular arithmetic this is easy to solve.
The curve consists of points, and the points have a X and Y coordinate: P(X,Y). The requirement for these points is to satisfy the equation:] [8]

$$y^2 = x^3 + ax + b$$

Based on the values given to **a and b** in the equation, they will determine the shape of the curve [9]
(2.2)

2.2.1 Public-key system

In Monero the used curve is the so called 'Ed25519' which is a Twisted Edward curve. Its constant paramaters are: [10]

q: a prime number; $q = 2^{255} - 19$;

d: an element of $\mathbb{F}_q$; $d = -121665/121666$;

E: an elliptic curve equation; $-x^2 + y^2 = 1 + dx^2y^2$;

G: a base point; $G = (x, -4/5)$;

l: a prime order of the base point; $l = 2^{252} + 27742317777372353535851937790883648493$;

$\mathcal{H}_s$: a cryptographic hash function $\{0,1\}^* \to \mathbb{F}_q$;

$\mathcal{H}_p$: a deterministic hash function $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$.

- q: is the total number of points on this curve
- d: an element used in the curve equation
- E: the equation for the Ed25519 curve.
- G: the base point or generator point.
- l: the order of the G point: it defines the maximum number of points we can use, and the maximum size our scalars can be.
- Hp and Hs are hash functions which map different inputs to a point and to a scalar respectively

In a public-key system We create two keys, a **public key**, and a **private key**. The public key is given freely, and any party can encrypt data by using it. However, the private key is kept secret and only those who hold it will have the ability to decrypt data.

{private ec-key} is a standard elliptic curve private key:its a scalar $x \in$ [1,l-1];
{public ec-key} is a standard elliptic curve public key: a point $A \doteq x * G$; [10]

Chapter 3

Signatures

In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to place a signatory to a message
Similarly, a digital signature is a technique that binds an entity to the digital data, it's a mathematical scheme for verifying the authenticity of digital messages or files. A valid digital signature gives the receiver a very strong reason to trust that the message was created by a known sender (a person who owns a private key), and that the message was not altered in transition. [11]

3.1. Schnorr signature

Schnorr published his whitepaper about a new authentication protocol in 1989.[12] The purpose of the signature is to prove to someone who knows your public information, that you have seen a particular value.Basically it proves that I am a possessor of a secret discrete logarithm x (private key) of a public group element P = x*G, thus I can prove knowledge of x without revealing anything about it.
Schnorr's solution for a person to prove a verifier that he knows his private key x is:

1. As a first step the prover chooses a random scalar k and sends **k*G** to the verifier.
2. The verifier chooses a random scalar **e** and sends it to the prover
3. The prover replies with the scalar **s <- k -x*e**
4. Lastly the verifier checks that sG+eP=kG

It can be proved mathematically that this scheme is safe:

$$\text{sG+eP} = (k - xe)\text{G+exG} = \text{kG-exG+exG} = \text{kG}$$

In simpler words being able to consistently pass verification implies knowledge of the secret **x**, my secret key.

3.1.1 Fiat-Shamir transformation

As it can be seen, the uppermentioned authentication requires two person to interact: **a prover and a verifier**. In order to make this autenthication non-interactive and publicly verifiable, they introduced the usage of a hash function to generate challenges, instead of the verifier

This was published worldwide by **Fiat and Shamir**.[13] They described that in order to "bind" our proofs of knowledge to a particular message M, a person should follow:

1. The prover chooses a random scalar k and computes e = H(kG || Message)
2. Lastly the prover computes the number s ← k +xe
3. The prover publishes his signature: (s, e)

It is **publicly verifiable** by anybody, because computing sG+eP and checking that e=H(sG+eP || M) means that the **prover owns the private key x** used for the signature.

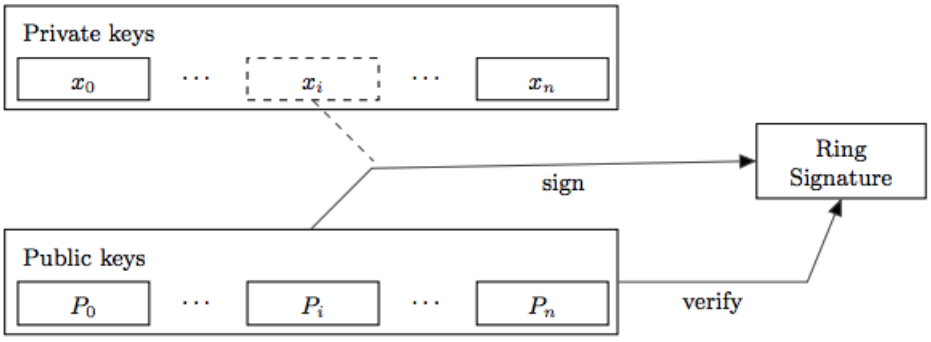
3.2. Ring Signatures

Ring signatures as the name states, contains a ring with members, and a signature generated from this ring. The Monero group implemented over time different ring signatures, each of them having an improvement, or changed for security reasons. I examined the most important ones:

3.2.1 Ring Signature

The first idea came from Rivest,Shamir, Tauman [14] They described Ring signatures as a variant of digital signatures in which the verification key is replaced by a ring (containing members) of public keys. Each **public key has a corresponding private key**, and only one private key. is used to produce a ring signature.

It is important to highlight, that all of the public keys play the same role in verifying the signature, so the **specific signing key(private key) used remains secret.**



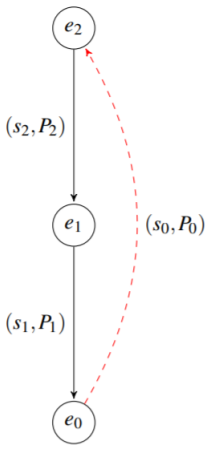
[10]

Two main underlying concepts regarding signatures are: [15]

Signer ambiguity: a third party should be able to determine that the signer must be a member of the ring, but the observer cannot specifically know which member

Linkability If two rings were signed by the same signer, the rings can be linked to each other with the use of a key image

Abe, Okhubo, Suzuki [15] published a whitepaper about ring signatures that shifted the focus on the discrete logarithm problem. It achieved **signer ambiguity**. It can be looked at as a generalization of a simple Schnorr Signature: The core idea of the scheme is that, for each public key, we compute an e value that depends on the previous e value.We also compute random s values, but the s value of the signer is computed using the signer's private key.



[16]

For a ring signature of the message **M** over the public keys **{P1,P2,. . . ,Pn}** and private keys **{x1, x2, ... xn}**, if the signer is denoted as index j with public and privatekey: P_j, X_j, the algorithm is:

1. Pick a random scalar k and let e_{j+1}=**H(k*G||M)**
2. From j+1 and wrapping around the modulus n, for every $i \neq j$, pick a random scalar **s** and compute e_{i+1}=H(s_iG+$e_i P_i$||M)
3. For the signer's part compute: s_j=**k-**$e_j x_j$
4. The published signature: (s, e)

The verification process consists of checking that some **s** value was calculated after all the **e** values were determined, which leads to the conclusion that a **private key was used while signing the message**

3.2.2 Linkable Ring Signature

The next improvement was the introduction of **linkability**. Joseph K. Liu, Victor K. Wei , and Duncan S. Wong. [17] published a whitepaper based on the original ring signature paper. The concept was the same as previous: the owner of a private key x, could produce one anonymous signature by selecting any set of co-members from a ring of public keys, without needing to collaborate with anyone.
The main difference consisted in the introduction of **key images**, which could prove that two rings were signed by the same signer, but not giving out which signer specifically.

With **j being the signer's index**, x_j being the signer's private key, R being the ring member's public keys, R =**{P1,P2,. . . ,Pn}** It could be calculated as:

$$KeyImage = x_j * H_p(R)$$

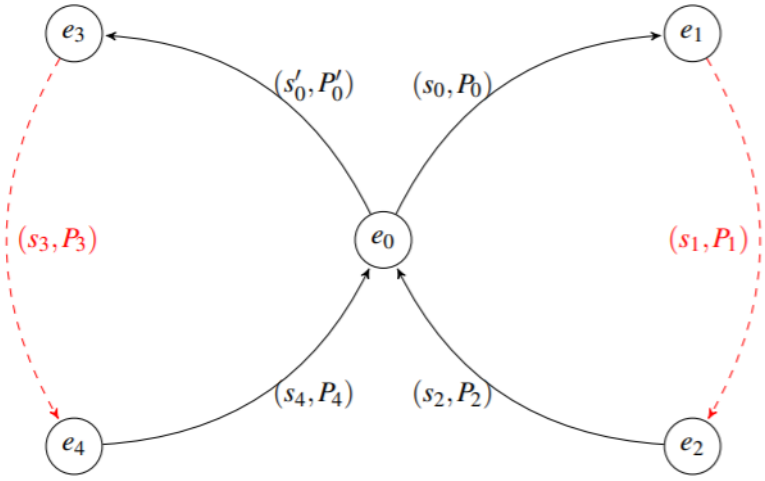
(3.1)

Here the hash function H_p takes in as input **the list of public keys from the ring members**, and hashes it to an **elliptic curve point**.

3.2.3 Borromean Ring Signature

In terms of space consumed and verification speed a more efficient ring signing algorithm was needed. This improvement was achieved by Gregory Maxwell, Andrew Poelstra [16], described as Borromean signature.

The basic idea is that there are multiple rings of public keys {A1,A2,A3}, each containing different members with public keys **(P0,P1,P2) - (P3,P4,P5) - (P6-P7-P8)** . Every ring has a selected private key, with which we would like to sign a message M in each of these rings. This basically proves that every ring signed the given message. The optimization brought was computing the **e0** as a shared **e** value for all of the rings.



[16]

The signature itself looks like:

$signature = (e0, (s_{0,0}, s_{0,1}, \ldots, s_{1,m01}), \ldots, (s_{n1,0}, \ldots, s_{n1,m_{n11}))}$ where the s's are made in each ring the similar way to the regular ring signatures.

Chapter 4

Python implementation

The other part of my research was to implement several methods in Python programming language. With the use of Google Colab and Visual Studio, and different Python cryptographic modules and libraries, I've reached the following:

4.1. Elliptic curve cryptography

4.1.1 Curve

To use the Twisted Edward curve Ed25519 and to get their parameters we simply: [18]

```
curve = Curve.get_curve('Ed25519')
G = curve.generator
Equation: -x^2 + y^2 = 1 -121665/121666 * x^2*y^2 (mod p)
Name = curve.name
Type = curve.type
size= curve.size
a=curve.a
d=curve.d
field=curve.field
order = curve.order
```

We achieved the next curve:

```
('Name: Ed25519, Equation: -x^2 + y^2 = 1 -121665/121666 * x^2*y^2 (mod p) '
 'Type: twistededward')
('Size: 256, a=-1, '
 'd=37095705934669439343138083508754565189542113879843219016388785533085940283555')
('G=(0x216936d3cd6e53fec0a4e231fdd6dc5c692cc7609525a7b2c9562d608f25d51a , '
 '0x6666666666666666666666666666666666666666666666666666666666666658), '
 'field=57896044618658097711785492504343953926634992332820282019728792003956564819949, '
 'order=7237005577332262213973186563042994240857116359379907606001950938285454250989\n')
```

4.1.2 Private-public keypair

We needed different public-private key pairs, in order to generate them: [19]

```
signer = EDDSA(hashlib.sha512)


for x in range(4):
  privKey = ECPrivateKey(secrets.randbits(32*8), curve)
  pubKey = signer.get_public_key(privKey, hashlib.sha512)
  publicKeys.append(privKey.get_public_key())
  privateKeys.append(privKey)
```

It resulted in pairs as:

```
ECPrivateKey:
  d: 3f29f59a5c4e793693946acc4c359ad234130e6be6db2a66e6c7ea06b69fcf26
ECPublicKey:
  x: 3e38f121c0c77f13148a794620a4279bc25c9b3fcb899c9a8c4d77719c73c9c4
  y: 1697bc7ccb32517b721f72e95cbcebcc0fb5d3d63f4996207c0c0dc766da6f65
```

4.2. Ring Signatures

The signing and verifying algorithms were implemented as the original whitepaper denoted: [15]

Ring Signature generation

```
def ring_sign(curve, message, public_keys, private_key, key_index):

   key_count = len(public_keys)
   e = [0] * key_count
   ss = [0] * key_count
   z_s = [0] * key_count

   alfa = random.randint(0,curve.order)
   Q = curve.mul_point(alfa, G)
   pi_plus_1 = (key_index+1) % key_count
   e[pi_plus_1] = H3(message, Q)

   for i in it.chain(range(key_index+1, key_count), range(key_index)):
      if i!=key_index:
        ss[i] = random.randint( 0,curve.order )
        next_i = (i+1) % key_count
        z_s[i] = curve.add_point(curve.mul_point(ss[i],G),
        curve.mul_point(e[i],public_keys[i].W))
        e[next_i] = H3(message, z_s[i])

   ss[key_index] = ( alfa - private_key[key_index].d
   * e[key_index] ) % curve.order

   return (public_keys, message, e[0], ss)
```

4.2.1 Linkable ring signature

The main difference was the so called key image generation. This was implemented as: [17]

```
public_keys_coords = list(map( lambda point: (point.W.x, point.W.y),
public_keyek))
public_keys_hash =H2(public_keys_coords )

H = find_ecc_point(public_keys_hash, curve.field)

##key_image
Y_tilde = curve.mul_point(private_keyek[key_index].d, H)
```

The linkability theorem could be checked: two rings were signed by the same signer:

```
keyimage1 =verify(curve, *ring_sign(curve, message,
publicKeys, privateKeys, key_index))

keyimage2= verify(curve, *ring_sign(curve, message2,
publicKeys, privateKeys, key_index))

is_Key_Image_Same = (keyimage1[1]==keyimage2[1])
```

```
LINKABILITY TEST
The two signatures were made by the same ring, therefore they are linkable
```

Chapter 5

Summary

I thoroughly enjoyed the time spent on this project. While at the start I faced a project which was unknown to me, it lead to deeper understanding of many key concepts in cryptography.

Firstly I could examine the mathematical side of cryptography. I could see how modular arithmetic is being used in real life projects, and how they are **basically** the basis of a security system. (Modular exponentiation, modular multiplication)

While having studied **RSA cryptosystems** beforehand, I was eager to learn about a new one: elliptic curve cryptography. ECC proved to be harder to crack, basically it means that for numbers of the same size, solving elliptic curve discrete logarithms is significantly harder than factoring, hence its higher security. [20]

Monero is a huge project, and it has many many chapters to explore. I shifted my focus on signatures, mainly the different versions of ring signatures. I was able to see in theory and in practice different algorithms to achieve the sender's anonimity.

My goal for the future is to stay connected to this project: to continue studying and implementing different cryptographical schemes used in **Monero**

Bibliography

[1] *Cryptocurrency investopedia*, en, May 2016. [Online]. Available: https://www.investopedia.com/terms/c/cryptocurrency.asp#:~:text=A%20cryptocurrency%20is%20a%20digital,a%20disparate%20network%20of%20computers..

[2] *Monero wikipedia*, en. [Online]. Available: https://en.wikipedia.org/wiki/Monero.

[3] *Khanacademy - what is modular arithmetic?* en. [Online]. Available: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic.

[4] *Modular arithmetic wikipedia*, en. [Online]. Available: https://en.wikipedia.org/wiki/Modular_arithmetic.

[5] *Brilliant - modular arithmetic*, en. [Online]. Available: https://brilliant.org/wiki/modular-arithmetic/.

[6] C. Dong, *Math in network security: A crash course*, en. [Online]. Available: https://www.doc.ic.ac.uk/~mrh/330tutor/ch03.html#:~:text=In%20modular%20arithmetic%2C%20the%20numbers,%2C%20subtraction%2C%20multiplication%20and%20division.&text=a%20is%20called%20the%20dividend,r%20is%20called%20the%20remainder.

[7] S. K. S. Rosy Sunuwar, *Encryption using elliptic curve cryptography*, en. [Online]. Available: https://cse.unl.edu/~ssamal/crypto/EEECC.php.

[8] *Elliptic curve cryptography wiki*, en. [Online]. Available: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography.

[9] K. M. Alonso, *Zero to monero*, en. [Online]. Available: https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf.

[10] N. van Saberhagen, *Cryptonote v 2.0*, en. [Online]. Available: https://bytecoin.org/old/whitepaper.pdf.

[11] B. Lutkevich, *Digital signatures*, en. [Online]. Available: https://searchsecurity.techtarget.com/definition/digital-signature.

[12] C. Schnorr, *Efficient identification and signatures for smart cards*, en. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F0-387-34805-0_22.pdf.

[13] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Advances in Cryptology — CRYPTO' 86*, pp. 186–194, DOI: 10.1007/3-540-47721-7_12.

[14] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," *Advances in Cryptology — ASIACRYPT 2001*, pp. 552–565, 2001. DOI: 10.1007/3-540-45682-1_32.

[15] M. O. Masayuki Abe1 and K. Suzuki1, *1-out-of-n signatures from a variety of keys*, en. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-36178-2_26.pdf.

[16] A. P. Gregory Maxwell, *Borromean ring signatures*, en. [Online]. Available: http://diyhpl.us/~bryan/papers2/bitcoin/Borromean%20ring%20signatures.pdf.

[17] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," *Information Security and Privacy*, pp. 325–335, 2004. DOI: 10.1007/978-3-540-27800-9_28.

[18] *Ec.py curves documentation*, en. [Online]. Available: https://ec-python.readthedocs.io/en/latest/_modules/ecpy/curves.html.

[19] *Ed25519 signing*, en. [Online]. Available: https://cryptography.io/en/latest/hazmat/primitives/asymmetric/ed25519/.

[20] *Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment?* en. [Online]. Available: https://eprint.iacr.org/2020/697.pdf.