Monero mining: CryptoNight Analysis

Orestis Konstantinidis
AL1180006

Examination committee:
Prof. Dr. Aggelos Kiayias, Department of Informatics and Telecommunications, NKUA.

Prof. Dr. Aris Pagourtzis, School of Electrical and Computer Engineering, NTUA.

Prof. Dr. Nikolaos Papaspyrou, School of Electrical and Computer Engineering, NTUA.

Supervisor:
Dr. Aggelos Kiayias, Associate Professor, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens.

Μεταπτυχιακό Πρόγραμμα «Αλγόριθμοι, Λογική και Διακριτά Μαθηματικά» - 2016

Η παρούσα Διπλωματική Εργασία
εκπονήθηκε στα πλαίσια των σπουδών
για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης
«Αλγόριθμοι, Λογική και Διακριτά Μαθηματικά»
που απονέμει το
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
του
Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την ..... από Εξεταστική Επιτροπή
αποτελούμενη από τους:

Ονοματεπώνυμο	**Βαθμίδα**	**Υπογραφή**
1.	.....	.....
2.	.....	.....
3.	.....	.....

ABSTRACT

Bitcoin has been a successful implementation of the concept of peer-to-peer electronic cash. Based on this technology several cryptocurrency projects have arisen, each one focusing on its purposes and goals. Monero is a decentralized cryptocurrency focusing on privacy and anonymity.

In a world of surveillance, Monero raises the alarm about one of the fundamental human rights, which is continuously violated: *Privacy*. In addition, Monero is built to achieve equality between miners. Corporations are taking over almost every successful cryptocurrency, by making mining participation harder and harder for the hobbyists and supporters. Monero tries to keep its community clean of unhealthy competition. This is achieved through egalitarianism, which is based on a cryptographic mining function.

This function is called CryptoNight and is part of the CryptoNote protocol, the heart of Monero's structure. The feature of this function that makes it egalitarian is a cryptographic property, named memory-hardness. CryptoNight is alleged to be memory-hard. But, still today, this is just a claim.

We put to the test this claim, trying to construct a formal mathematical proof, but we fail to do so. We discuss the reasons for our failure and try to use them to construct an attack on this feature. To our knowledge, we are the first to study this CryptoNight's property and the first to present graphically all the stages of CryptoNight's functionality.

Finally, we present the knowledge gained and wish for this document to be useful in the future to colleagues that want to contribute in this field. The aim of this work is to contribute to Monero's fight for privacy, anonymity and equality.

ΣΥΝΟΨΗ

Το κρυπτονόμισμα Bitcoin αποτελεί την πρώτη πετυχημένη εφαρμογή της ιδέας του ηλεκτρονικού χρήματος χωρίς την διαμεσολάβηση τρίτων. Στην πορεία, πολλά κρυπτονομίσματα βασίστηκαν στην συγκεκριμένη τεχνολογία, εστιάζοντας το καθένα στους δικούς του στόχους και σκοπούς. Το κρυπτονόμισμα Monero είναι ένα τέτοιο εγχείρημα, βασικός σκοπός του οποίου είναι η διασφάλιση της ιδιωτικότητας και της ανωνυμίας.

Σε έναν κόσμο όπου η παρακολούθηση εντείνεται, το εγχείρημα του Monero σημαίνει τον συναγερμό για την διαρκή καταπάτηση ενός εκ των θεμελιωδών ανθρώπινων δικαιωμάτων. Επιπλέον, καθώς οι επιχειρήσεις έχουν περιορίσει δραματικά τον υγιή ανταγωνισμό σχεδόν σε όλα τα διαδεδομένα κρυπτονομίσματα, το Monero προσπαθεί να τον διατηρήσει στην κοινότητά του. Ένα από τα δομικά στοιχεία του Monero είναι η διατήρηση της ισότητας μεταξύ των "ανθρακωρύχων" (miners), η οποία επιτυγχάνεται μέσω της ισονομίας (egalitarianism).

Η ισονομία είναι συνέπεια μιας ιδιότητας της κρυπτογραφικής συνάρτησης που χρησιμοποιείται για την "εξόρυξη" νομισμάτων. Η συνάρτηση που χρησιμοποιείται στο Monero για αυτόν τον σκοπό λέγεται CryptoNight και είναι μέρος του CryptoNote πρωτοκόλλου. Το στοιχείο της συνάρτησης που επιτυγχάνει την ισονομία είναι μια κρυπτογραφική ιδιότητα, η οποία ονομάζεται memory-hardness. Η CryptoNight συνάρτηση θεωρείται ότι διαθέτει αυτήν την ιδιότητα. Όμως, μέχρι σήμερα αυτό παραμένει ισχυρισμός. Απ' όσο γνωρίζουμε, δεν υπάρχει μαθηματική απόδειξη για αυτόν τον ισχυρισμό αλλά ούτε και κάποια επίθεση που να τον διαψεύδει.

Θέλοντας να ελέγξουμε την ορθότητα αυτού του ισχυρισμού, προσπαθήσαμε να κατασκευάσουμε μια μαθηματική απόδειξη. Αναφέρουμε τους λόγους για τους οποίους αποτυγχάνουμε να διατυπώσουμε μία τέτοια απόδειξη και προσπαθούμε να τους χρησιμοποιήσουμε για να καταρρίψουμε αυτόν τον ισχυρισμό. Απ' όσο γνωρίζουμε, η παρούσα εργασία είναι η πρώτη που μελετά αυτήν την ιδιότητα για την συνάρτηση CryptoNight και παρουσιάζεται για πρώτη φορά γραφικά η εσωτερική δομή της.

Τέλος, παρουσιάζουμε την γνώση που αποκτήσαμε και ελπίζουμε αυτή η εργασία να φανεί χρήσιμη μελλοντικά σε συναδέλφους που θέλουν να συμβάλλουν στην έρευνα στο ευρύτερο πεδίο. Στόχος αυτής της έρευνας είναι να συνεισφέρει στην προσπάθεια του εγχειρήματος Monero για την διασφάλιση της ιδιωτικότητας, της ανωνυμίας και της ισότητας.

ACKNOWLEDGEMENTS

I want to express my gratitude to Prof. Dr. Aggelos Kiayias for his supervision. I want to thank explicitly Dionysis Zindros, for his crucial assistance in the progress of this thesis, his support and his patience. I highlight the help I got from the thesis of Kostis Karantias and his work. Christos Nasikas, colleague and friend, has offered academic and personal guidance every time I needed it.

Along with Prof. Dr. Aggelos Kiayias, I am also thankful to the other two members of my three-member committee, Prof. Dr. Aris Pagourtzis and Prof. Dr. Nikolaos Papaspyrou, for the evaluation of my thesis.

During my master curriculum I had a huge offer of valuable courses and inspiring professors. This program has been an important factor of my academic and personal development.

My research was accompanied by a lot of interesting discussions and help from friends that took the time to listen about my thoughts and interests. Among them I should not neglect to name Yanna Papadodimitraki for her valuable contribution and Vassilis Agiotis for the illustration of my thesis. I thank for the support and patience my friends G.K., M.G., K.A., M.P. and C.V..

I also thank the Monero project and all of the people who decide to work in projects that have significant political and social impact. They have been an example for me.

For his invalueable contribution to my personality and culture development as an active member of society I have to thank my friend and my role model C.C.

Last, but most significantly, I want to salute all these people who do (or did) not have the chance to study and those who chose not to, in order to fight for freedom.

CONTENTS

CHAPTER 0

PREFACE

0.1 Why Monero?

I would like to take a moment here and share the experience of selecting a research topic. This was as important to me as the rest of my work. I am not talking about concepts of ”good topic” or a list of ”factors to consider”. I am definitely not an expert on this subject and you can find many valuable information about this process online. I would like to share with you the impact of an argument, about selecting this topic over other options, that was addressed to me by my colleague and friend, Dionysis Zindros. He said to me that in his opinion this topic would be ”beneficial for the Monero community”.

I strongly recommend before you select your topic of research to take an evening of your time and read a paper titled ”The Moral Character of Cryptographic Work”, written by Phillip Rogaway [64]. No matter how small this world makes you feel, when you select a topic you have responsibility. I was lucky to work with people who understood this and led me to take a moment and think about what cause I really wanted to contribute to.

In our days, cryptographic work is a political action. There is no doubt about that. Think through your intentions and the person you want to be before you select your path. Remember, you have responsibility.

0.2 An important thank you note

During my research I had a lot of help from forum answers and conversations and most of all, from the monero stack exchange users [56]. I would like to thank many users for their valuable share of knowledge. I had help from a lot of users and several stack exchange forums, who pointed me to the right direction or helped me clarify my misunderstanding of notions from time to time.

I want to thank them for the aid, but most important, I want to thank everyone who contributes to this knowledge sharing. It means a lot to me and gives me a perspective on my occupation. I don't see the work of a cryptographer as an 8-hour employment to make ends meet. I hope my interests and efforts to be much more than a day job. I feel part of a community that contributes to the real world and keeps me motivated and convinced that our work makes the world a better place to live in. I thank you all for your example.

0.3 Narrative

I will change the first person singular narrative. Apart from all the people I have already thanked, my personality has been shaped by my family, my friends, important acquaintances and stories about my heros. All of them are part of me, part of who I am.

As a result, I see this work as a collaboration of all of us and that completes the list of the reasons why I will keep a first person plural narrative from now on. I could not do that in the above paragraphs because this section, as you can understand, is extremely personal. I thank you all.

Another change will be the way I am addressing to you, my reader. It will be, from now on, in a third person point of view. This is just a personal aesthetic choice. I thank you for taking the time to read our work. I hope you find it enjoyable.

CHAPTER 1

INTRODUCTION

> If you don't believe it or don't get it, I don't have the time to try to convince you, sorry.
>
> *Satoshi Nakamoto*

1.1 Decentralization

Decentralized networks aim to eliminate the need for a central authority. Rather than to hand over control to a central party, decentralized networks are run by the participants themselves. This means that the entire system becomes more distributed.

As a result, any processes underpinned by decentralized technology become significantly harder to shut down. This is primarily due to the elimination of a single point of failure in the system. It is worth noting that decentralization brings with it a lot of positive effects. For one, it means that users will no longer need to put their trust in a single central authority to perform a process.

A decentralized network consists of a so-called peer-to-peer network. This means that data is distributed across numerous devices.

Ownership of users' data

This is a simplistic view of decentralization but it highlights one of its most fundamental strengths. The advent of cryptocurrencies and blockchains – which build on decentralization – can be said to give back ownership of data to users. This is because it does not rely on a single entity to handle users' information. In an age where *big data* is increasingly becoming a popular topic, decentralization can be seen as a solution to this problem.

Companies like Google or Facebook are often accused of infringing on users' privacy by collecting information regarding them. However, this is not something that is exclusive to corporations. Governments could also abuse their power over such information.

This highlights the main problem with centralized entities, as the risk of abusive behavior becomes more prevalent. Moreover, it is hard to stop this from taking place, as such centralized entities are often crucial to the system as a whole. Nonetheless, some might object to the notion of preserving privacy through sharing information more freely, across multiple devices. Although this approach might seem counterintuitive, it is actually quite sound. As blockchains employ cryptography, the information can be kept safe and private.

Egalitarian tool

It should be noted that the benefits of decentralization extend far beyond practical reasons. In fact, decentralization is increasingly being heralded as a powerful egalitarian tool.

As people around the world face certain oppressive regimes that attempt to seize control over the free press or shut down users' access to social media, decentralization might prove even more useful in countering censorship. A government could shut down a service like Twitter or Facebook comparatively easy. This could simply be achieved by denying traffic going to any of Twitter's or Facebook's central servers.

Decentralized networks – on the other hand – make use of peer-to-peer networks, which would be virtually impossible to censor. This is because a government would then have to block all of the undesired points of the peer-to-peer network, rather than just a central server.

Moreover, one of the most important draws of decentralized networks is that they are open. As a result, anyone with the expertise to do so could potentially develop their own services, processes, products or tools. In fact, decentralized networks can be seen as nothing more than an underlying platform. What this platform allows for are the benefits of decentralization.

This can be compared to the internet itself. The world wide web is nothing more than an underlying platform that allows for applications to be built on top. The modern notion of decentralized blockchain technology is less than a decade old. Decentralized technology presents a plethora of new opportunities.

Overview

To sum up, the benefits that decentralized systems offer are:

- Users don't have to put trust in a central authority
- It it less likely for a single point of failure to exist
- There is less censorship
- Decentralized networks are more likely to be open development platforms
- There is potential for network ownership alignment

The last point is the idea that the people who contribute value to a decentralized network receive ownership or economic stake in the network, that becomes more valuable as the network grows. This is one of the most exciting things that blockchain technology brings to decentralized networks, as it allows economics to be designed into the networks themselves, to create the right incentives for early participants to become value-contributing users.

1.2 Summary of Contribution

We study the claim that Monero's mining function, CryptoNight, is memory-hard. Our contributions in this thesis are as follows:

1. We represent graphically the functionality of CryptoNight
2. We introduce a mathematical model for CryptoNight function
3. We attempt to construct a formal mathematical proof of CryptoNight's memory-hardness property
4. We attempt to attack CryptoNight's memory-hardness property

Graphical representation

To our knowledge, we are the first to present graphically the whole process of the CryptoNight function. We believe that this will help any researcher who wants to analyze or understand the way this function handles its components and the relation between them.

Mathematical model

Then, we construct a mathematical model. We make intuitive but solid assumptions about:

- The nature of the AES encryption operation and its output
- The distribution of the input

During this process we note implementation details that may be theoretically problematic about the addition and multiplication operations used. However, we proceed in our analysis, assuming that these minor problems don't exist. The reason is twofold – one, these problems admit a relatively easy fix and, two, because of the nature of our results we care to highlight other characteristics that didn't allow us to succeed in our attempts.

Proof

We fail to construct a formal mathematical proof of CryptoNight's memory-hardness property. We discuss the reasons for this result and present our thoughts and effort. This analysis can be helpful for the person or team that wants to research this problem or a similar one.

Attack

An attack on CryptoNight's property seems improbable and we discuss the reasons behind this claim. We hope that this analysis is helpful too, for future research on CryptoNight.

1.3 Thesis structure

We tried to make this document readable. From this point of view, we tried to achieve completeness, in the sense that the reader should not refer to external resources to understand the basic arguments involved. Of course, a more keen reader can find references for a detailed study of the notions involved in our work, but that is something that we wanted to be as optional as possible.

In the first part, the reader will find an overview of the first complete implementation of a cryptocurrency. That was a paper published under the pseudonym Satoshi Nakamoto [58], introducing the Bitcoin project to the world. This overview is certainly not a detailed description of all aspects of Bitcoin, as this would be unproductive for the discussion in this work. However, we believe that we give a satisfactory description of the project and we hope that the reader will acquire an adequate understanding of the structure of this novelty.

We introduce the reader to the concept of mining, based on the knowledge gained through the study of the Bitcoin project. The purpose of this description is to introduce the notion of *egalitarianism* and more precisely, the notion of egalitarian mining. This notion was for a long time a folklore subject of dispute in the community, being a starter for many passionate conversations. It was formally defined recently, in the work of Dimitris Karakostas, Aggelos Kiayias, Christos Nasikas and Dionysis Zindros [45].

In the second chapter of the first part, we introduce another cryptocurrency, named Monero, and the purposes of its community along with a technical description of its features. The basic structure for Monero project is the CryptoNote protocol, described in a paper published under the pseudonym Nicolas van Saberhagen [65]. We describe its features as well.

Then, we get to the point where we can discuss the problem of interest of this thesis. This is Monero's mining function, CryptoNight. This function is part of the CryptoNote protocol and it is the reason that Monero claims to offer egalitarianism in its mining process. This feature's existence is due to a known property, called memory-hardness. This function is alleged to be memory-hard, although we found no formal research on this matter. CryptoNight's functionality and features are described in the first chapter of the second part.

In the next chapter, we try to construct a formal mathematical proof of CryptoNight's memory-hardness property and we discuss the reasons we failed to do so. Using these reasons, we attempt to construct an attack on this property.

Finally, we sum up our observations and try to highlight important knowledge gained through this journey.

In the first part, the reader can find formal mathematical definitions of the notions used or referred, in an effort for this thesis to be both readable and complete. That is presented in the Cryptographic Background section of the first chapter. Constructions and details about the Monero project, are included in the Monero chapter. However, this information is not needed for the reader to keep up with the flow of our work or to understand our remarks. Nevertheless, they were included for the sake of completeness and for the reader, as he/she might find this information beneficial.

Part I

From decentralization to re-centralization

CHAPTER 2

PRELIMINARIES

> Security is a binary state. A system cannot be secure against malicious attackers and insecure against other ”noble” parties. It is either secure or insecure. And if the prospect of an attack exists, then security collapses.
>
> *Edward Snowden*

2.1 Cryptography background

Here we will include the formal mathematical definition of any cryptographic primitive that the reader will need to study this thesis. Our goal is to make the citation and references in this document an optional read. We believe that we can make this project complete on its own and any available source will be needed only for the experienced reader who may want to get a deeper understanding of the mechanics discused.

A great help, in order to keep this section as concrete as possible, was the work of my collegue and friend Kostis Karantias. He had already defined some notions, which are helpful in this thesis too. Some of his work is reproduced here. I thank him for his help and appreciate his work. I definitely refer the reader to his thesis [46].

2.1.1 Hash function

We will define the syntax and the security model of the cryptographic hash function, as introduced in [47]. We will slightly change their definition, because we assume no key as input to the hash function. In our case, the only input is a message.

Definition 2.1 (Hash function - syntax)**.** A *hash function* is a probabilistic polynomial-time (p.p.t.) algorithm H satisfying the following:

- There exists a, polynomial in n, function l such that H is a (deterministic) p.t. algorithm that takes as input any string $x \in \{0,1\}^*$ and outputs a string:

$$H(x) \in \{0,1\}^{l(n)}$$

If for every n, H is defined only over inputs of length $l'(n)$ and $l'(n) > l(n)$, then we say that H is a *fixed-length hash function* with length parameter l'. An output of a hash function is called a *digest* of the function.

Notice that in the fixed-length case we require that l' be greater than l. This ensures that the function is a hash function in the classic sense in that it *compresses* the input. We remark that in the general case we have no requirement on l because the function takes for input all (finite) binary strings. Thus, by definition, it also compresses.

We will now define security for this model. We begin by defining a game for a hash function H, an adversary $\mathcal{A}$ and a security parameter n:

The collision-finding game Hash-coll$_{\mathcal{A},H}(n)$: [47]

1. The adversary $\mathcal{A}$ outputs a pair x and x'.
   Formally, $(x, x') \leftarrow \mathcal{A}(s)$.
2. The output of the experiment is 1 if and only if $x \neq x'$ and $H(x) = H(x')$. In such a case, we say that $\mathcal{A}$ has found a collision.

The definition of collision resistance for hash functions states that no efficient adversary can find a collision except with negligible probability.

Definition 2.2. [47] A hash function H is *collision resistant* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function **negl**$(\cdot)$ such that

$$Pr[\text{Hash-coll}_{\mathcal{A},H}(n) = 1] \leq \textbf{negl}\,(n) \tag{2.1}$$

2.1.2 Password Scramblers

Passwords[1] are user-memorizable secrets. Typical (user-chosen) passwords often suffer from low entropy and can be attacked by trying out all possible password candidates. If we let asside the case of a dedicated cryptographic protocol on an interactive session, the next best protection are password scramblers performing *key-stretching*. Christian Forler, Stefan Lucks and Jakob Wenzel [34], give three basic conditions a good password scrambler should satisfy at least:

1. Given a password pwd, computing $PS(pwd)$ should be "fast enough" for the user.
2. Computing $PS(pwd)$ should be "as slow as possible", without contradicting condition 1.
3. Given $y = PS(pwd)$, there must be no significantly faster way to test q password candidates $x_1, x_2, \ldots, x_q$ for $PS(x_i) = y$ than by actually computing $PS(x_i)$ for each x_i.

[1] Passphrases and personal identification numbers (PINs) are considered "passwords", in this context.

Tratidionally, most password scramblers satisfy condition 2 by iterating a cryptographic primitive (a block cipher or hash function) many times. However, an adversary with b computing units (*cores*) can try b different passwords in parallel. With today's availability of graphical processing units (GPUs), slowing down these kind of attacks becomes a pressing question.

2.1.3 Memory-Hard Functions

We will define the notion of the memory-hard function in the Parallel Random Oracle Model (pROM) of [2], as introduced in [3]. First, we will define the model along with the associated complexity notions.

The parallel Random Oracle Model. We consider an algorithm $\mathcal{A}$ executing in the pROM of [2]. Let this algorithm be repeated an arbitrary amount of times. After each invocation we make states. At invocation $i \in \{1, 2, \dots\}$ algorithm $\mathcal{A}$ keeps the state σ_{i-1} it produced. Next $\mathcal{A}$ can make calls $\mathbf{q}_i = (q_{1,i}, q_{2,i}, \dots)$ to the *fixed input length random oracle* H (ideal compression function). After it receives the digest of H, is allowed to perform arbitrary computation before producing its output (the next state σ_i). The state σ_0 contains the input to the computation and no other state is kept by $\mathcal{A}$. Now, we need some complexity notions to be defined.

The cumulative memory complexity (CMC) is defined to be

$$cmc(\mathcal{A}) = \mathop{\mathbb{E}}_{H}\left[\max_{x,r} \sum_{i} |\sigma_i|\right]$$

where $|\sigma|$ is the bit-length of state σ, the expectation is taken over the choice of H and $max_{x,r}$ denotes the maximum over all inputs and coins of $\mathcal{A}$.

Moreover, the *time complexity* (TC), $time(\mathcal{A})$ is the maximum running time of $\mathcal{A}$ in any execution. Similarly, the *space complexity* (SC) is the largest state it ever outputs in any execution.

Oracle function. Let f be a function over strings depending on the choice of H. We consider the scenario in which we want to compute f on $m \in \mathbb{N}^+$ arbitrary distinct inputs. Let $\mathbb{A}_{f,m,q}$ be the set of pROM algorithms that accomplish this, making at most q queries to H. Then,

(a) f is an oracle function

(b) The *amortized cumulative memory complexity* (aCMC) of f is defined to be

$$cmc_{m,q}(f) = \min\left\{\frac{cmc(\mathcal{A})}{n} : n \in [m], \mathcal{A} \in \mathbb{A}_{f,n,q}\right\}$$

This definition provides a good lower-bound on the *amortized time complexity* of a function [2].

For more detailed information about the above, we refer the reader to the appendix of [3]. The reader can find some technical details there that are beyond of the scope of this thesis. Now we are ready to define properly the notion of the memory-hard function.

Definition 2.3 (Memory-Hard Function)**.** Let $\{f_{\sigma,\tau}\}_{\sigma,\tau\in\mathbb{N}^+}$ be a family of (oracle) functions and $\mathcal{N}$ be a sequential pROM algorithm which, on input (σ,τ,x), outputs $f_{\sigma,\tau}(x)$ in time at most $\tau\sigma$ using space at most σ. Then $F=\left(\{f_{\sigma,\tau}\},\mathcal{N}\right)$ is an (h,g,t)-memory-hard function (for up to m instances and q queries) if it has memory-hardness at least $h(\cdot)$, memory-gap at most $g(\cdot)$ and throughput at least $t(\cdot)$ (all functions of σ and τ).

$$cmc_{m,q}(f_{\sigma,\tau})\geq h(\sigma,\tau)\quad \frac{\text{space}(\mathcal{N})*\text{time}(\mathcal{N})}{cmc_{m,q}(f_{\sigma,\tau})}\leq g(\sigma,\tau)\quad \frac{\text{space}(\mathcal{N})}{\text{time}(\mathcal{N})}\geq t(\sigma,\tau)$$

The above definition [3], although extremely rigid, is not that intuitive. In order to describe memory requirements, we will mention another definition given in [34], without causing any conflict with the above. Before we give the second definition, one should notice that for any parallelized attack, using b cores, the required memory per core is decreased by a factor of $\frac{1}{b}$, and vice versa.

Definition 2.4 (Memory-Hard Function - intuitive)**.** Let g denote the memory cost factor. For all $\alpha>0$, a memory-hard function f can be computed on a Random Access Machine using $\textbf{space}(g)$ space and $\textbf{time}(g)$ operations, where $\textbf{space}(g)\in\Omega(\textbf{time}(g)^{1-\alpha})$.

Thus, for $\textbf{space}(\cdot)$ $\textbf{time}(\cdot)=G^2$ with $G=2^g$, using b cores, we have

$$\left(\frac{1}{b}\cdot\textbf{space}(\cdot)\right)\left(b\cdot\textbf{time}(\cdot)\right)=G^2. \tag{2.2}$$

In their paper a formal generalization of this notion is given but it is beyond of the scope of this thesis. For more information about memory-hardness, the reader is refered to their work [34].

2.1.4 Pseudorandom Functions

In cryptography, a pseudorandom function family, abbreviated PRF, is a collection of efficiently-computable functions which emulate a random oracle in the following way: no efficient algorithm can distinguish (with significant advantage) between a function chosen randomly from the PRF family and a random oracle (a function whose outputs are fixed completely at random). With that in mind, we must first recall the definition of oracle indistinguishability and then proceed to define a pseudorandom function. Reproduced from [61]:

Definition 2.5 (Oracle Indistinguishability)**.** Let $\{O_n\}_{n\in\mathbb{N}}$ and $\{O'_n\}_n$ be ensembles where O_n, O'_n are probability distributions over functions $f:\{0,1\}^{l_1(n)}\rightarrow\{0,1\}^{l_2(n)}$ for some polynomials $l_1(\cdot), l_2(\cdot)$. We say that $\{O_n\}_n$ and $\{O'_n\}_n$ are *computationally indistinguishable* (denoted by $\{O'_n\}_n\approx\{O'_n\}_{n\in\mathbb{N}}$) if for all non-uniform p.p.t. oracles machines D, there exists a negligible function $\epsilon(\cdot)$ such that $\forall n\in\mathbb{N}$

$$\left|Pr\left[F\leftarrow O_n:D^{F(\cdot)}(1^n)=1\right]-Pr\left[F\leftarrow O'_n:D^{F(\cdot)}(1^n)=1\right]\right|<\epsilon(n). \tag{2.3}$$

Definition 2.6 (Pseudorandom Function)**.** A family of functions $\{f_s : \{0,1\}^{|s|} \rightarrow \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$ is *pseudorandom* if

- (Easy to compute): $f_s(x)$ can be computed by a p.p.t. algorithm that is given input s and x
- (Pseudorandom): $\{s \leftarrow \{0,1\}^n : f_s\}_n \approx \{F \leftarrow RF_n : F\}_n$.

where, RF_n is considered a family of functions that are random oracles.

The intuition in this section is that we can't construct a function that actually implements a random oracle. But, for practical purposes, we can construct a function or family of functions that are in distriguishable from random oracles. That is good enough for a security point of view and the next game is necessary in order to define the security model for this mathematical element.

Algorithm 1 The game algorithm for a pseudorandom function f_s (Adversary $\mathcal{A}$)

function $\mathsf{game}_{\mathcal{A}}^{f_s}(n)$
  $b \xleftarrow{\$} \{0,1\}$
  if $b = 0$ **then**
    $f' \leftarrow \{\{0,1\}^n \rightarrow \{0,1\}^n\}$
  else
    $f' \leftarrow f_s$
  end if
  $b^* \leftarrow \mathcal{A}^{f'}(1^n)$
  if $b = b^*$ **then**
    return 1
  end if
  return 0
end function

Based on the above algorithm we can now define the security model:

Definition 2.7 (Security model)**.** Let f_s be a pseudorandom function for some $s \in \{0,1\}^n$, where n is the security parameter. Then, $\forall$ p.p.t. adversarial algorithm $\mathcal{A}^{f'}$, with access to some random oracle function f', there exists a negligible function $\epsilon(\cdot)$ such that $\forall n \in \mathbb{N}$:

$$\left| \Pr_{s\xleftarrow{\$}\{0,1\}^n}\left[\mathcal{A}^{f_s}(1^n) = 0\right] - \Pr_{f'\xleftarrow{\$}\left\{\{0,1\}^n\rightarrow\{0,1\}^n\right\}}\left[\mathcal{A}^{f'}(1^n) = 0\right] \right| < \epsilon(n). \quad (2.4)$$

Notice that if someone knows s then it is easy to distinguish f_s from a random function. In order to consider this function indistinguishable from a random function, one should keep seed s secret.

2.1.5 Pebbling game

Hellman presented in [42] a possibility to trade memory/space S against time T in attacking cryptographic algorithms, i.e. he has introduced the idea of a time-memory trade-off (TMT) in terms of generic attacks. Hence, we can assume that an adversary with access to this algorithm and restricted resources is always looking for a sweet spot to optimize $S \cdot T$. For, studying the TMT, one needs to choose a certain model. But first we must introduce the reader to the notion of the *directed acyclic graph*. Reproduced from [34]:

Definition 2.8 (Directed Acyclic Graph)**.** Let $\Pi(\mathcal{V}, \mathcal{E})$ be a graph consisting of a set of vertices $\mathcal{V} = (v_0, v_1, \ldots, v_{n-1})$ and a set of edges $\mathcal{E} = (e_0, e_1, \ldots, e_{l-1})$, where $\mathcal{E} = \varnothing$ is a valid variant. $\Pi(\mathcal{V}, \mathcal{E})$ is a *directed acyclic graph*, if every edge in $\mathcal{E}$ consists of a starting vertex v_i and an ending vertex v_j, with $i \neq j$. A path through $\Pi(\mathcal{V}, \mathcal{E})$ beginning at vertex v_i must never reach v_i again (else, there would be a cycle). If there exists a path from a vertex v_i to a vertex v_j in the graph with $i \neq j$, we will write $v_i \leq v_j$.

In 1970, Hewitt and Paterson introduced a method for analyzing TMTs on directed acyclic graphs (DAG), called *pebbling game*. It has been occasionally used in cryptographic context, see e.g. [28] for a recent example. The pebble game model is restricted to DAGs with bounded in-degree and can be seen as a single-player game. The two following definitions are produced from [3]:

Definition 2.9 (Parallel/Sequential Graph Pebbling)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be a DAG and let $T \subseteq \mathcal{V}$ be a target set of nodes to be pebbled. A pebbling configuration (of G) is a subset $P_i \subseteq \mathcal{V}$. A legal parallel pebbling of T is a sequence $P = (P_0, \ldots, P_t)$ of pebbling configurations of G where $P_0 = \varnothing$ and which satisfies conditions 1 & 2 below. A sequential pebbling additionally must satisfy condition 3.

1. At some step every target node is pebbled (though not necessarily simultaneously).
$$\forall x \in T \;\; \exists z \leq t \quad : \quad x \in P_z. \tag{2.5}$$

2. Pebbles are added only when their predecessors already have a pebble at the end of the previous step.
$$\forall i \in [t] \quad : \quad x \in (P_i \setminus P_{i-1}) \;\Rightarrow\; \textbf{parents}(x) \subseteq P_{i-1}. \tag{2.6}$$

3. At most one pebble placed per step.
$$\forall i \in [t] \quad : \quad |P_i \setminus P_{i-1}| \leq 1. \tag{2.7}$$

We denote with $\mathcal{P}_{G,T}$ and $\mathcal{P}^{||}_{G,T}$ the set of all legal sequential and parallel pebblings of G with target set T, respectively. Note that $\mathcal{P}_{G,T} \subseteq \mathcal{P}^{||}_{G,T}$. In the case where $T = \textbf{sinks}(G)$, we will simply write $\mathcal{P}_G$ and $\mathcal{P}^{||}_G$.

Definition 2.10 (Time/Space/Cumulative Pebbling Complexity)**.** The *time*, *space*, *space-time* and *cumulative* complexity of a pebbling $P = \{P_0, \ldots, P_t\} \in \mathcal{P}^{||}_G$ are defined to be:

$$\Pi_t(P) = t \quad \Pi_s(P) = \max_{i\in[t]}|P_i| \quad \Pi_{st}(P) = \Pi_t(P)\cdot\Pi_s(P) \quad \Pi_{cc}(P) = \sum_{i\in[t]}|P_i|.$$

For $\alpha \in \{s, t, st, cc\}$ and a target set $T \subseteq \mathcal{V}$, the *sequential* and *parallel* pebbling complexities of G are defined as

$$\Pi_\alpha(G,T) = \min_{P\in\mathcal{P}_{G,T}} \Pi_\alpha(P) \qquad \text{and} \qquad \Pi^{||}_\alpha(G,T) = \min_{P\in\mathcal{P}^{||}_{G,T}} \Pi_\alpha(P).$$

When $T = \textbf{sinks}(G)$, we simplify notation and write $\Pi_\alpha(G)$ and $\Pi^{||}_\alpha(G)$.

We notice that the definition comes along with the intuition about these complexities. For $\alpha \in \{s, t, st, cc\}$ and any G, the parallel pebbling complexity is at most as high as the sequential, i.e., $\Pi_\alpha(G) \geq \Pi^{||}_\alpha(G)$, and cumulative complexity is at most as high as space-time complexity, i.e. $\Pi_{st}(G) \geq \Pi_{cc}(G)$ and $\Pi^{||}_{st}(G) \geq \Pi^{||}_{cc}(G)$.

2.2 Bitcoin

Bitcoin [58] is a decentralized digital currency that enables instant payments to anyone, anywhere in the world. Bitcoin uses peer-to-peer technology to operate with no central authority: transaction management and money issuance are carried out collectively by the network.

The original Bitcoin software by Satoshi Nakamoto was released under the MIT license. Most client software, derived or "from scratch", also use open source licensing.

Bitcoin is the first successful implementation of a distributed cryptocurrency, described in part in 1998 by Wei Dai on the cypherpunks mailing list. For the reader to understand what this list was, we reproduce from *cryptoanarchy.wiki* [21]:

> *The Cypherpunks mailing list was started in 1992, and by 1994 had 700 subscribers. At its peak, it was a very active forum with technical discussion ranging over mathematics, cryptography, computer science, political and philosophical discussion, personal arguments and attacks, etc., with some spam thrown in. An email from John Gilmore reports an average of 30 messages a day from December 1, 1996 to March 1, 1999, and suggests that the number was probably higher earlier. The number of subscribers is estimated to have reached 2000 in the year 1997.*

It is during this period that the community was energised by a battle with the U.S. intelligence establishment relating to the export of cryptography (which the U.S. Government had at the time classified as a munition).

This is a battle that the cypherpunk movement and broader civilian cryptography community largely won, though some variations of government proposals still pop up to this day. More about the cypherpunks mailing list and the archived conversations can be found in cryptoanarchy wiki [21].

Building upon the notion that money is any object, or any sort of record, accepted as payment for goods and services and repayment of debts in a given country or socio-economic context, Bitcoin is designed around the idea of using cryptography to control the creation and transfer of money rather than relying on central authorities.

Bitcoin is pseudonymous [46]: the identity of each user is only their *address* (a user can have multiple addresses), which corresponds to an ECDSA public key [31]. This address can be used to receive money from other users. Each user can spend money only if they have their corresponding private key. A set of ECDSA keypairs comprises a *wallet*.

As in fiat money, transfer of value in Bitcoin happens with transactions. A transaction has inputs and outputs (see sections 2.2.1, 2.2.2, 2.2.3). An output is where the value creation happens for the receiver. An output can be later redeemed by using its designated receiver's private key and turned into an input to be used for another transaction.

Bitcoins have all the desirable properties of a money-like good. They are portable, durable, divisible, recognizable, fungible, scarce and difficult to counterfeit.

2.2.1 Transactions

A *transaction* is a collection of inputs and outputs. It uses the sum of the inputs' values as credit to debit each output accordingly. As it makes sense, a transaction is only valid as long as all its outputs and inputs are valid. It should also be clear that the value of the outputs should not exceed the value of the inputs, otherwise we would be creating value out of thin air with new transactions. Specifically this is expressed as

$$\sum_{i \in inputs} i.value \geq \sum_{o \in outputs} o.value$$

This is sometimes called the *Law of Conservation*. In cases where

$$\sum_{i \in inputs} i.value > \sum_{o \in outputs} o.value$$

we call

$$\sum_{i \in inputs} i.value - \sum_{o \in outputs} o.value$$

the *transaction fee*.

The transaction id is the digest of the two times hashing operation (SHA256[2]) on the transaction data. In figure 2.1 the reader can see the notion described above in practice.

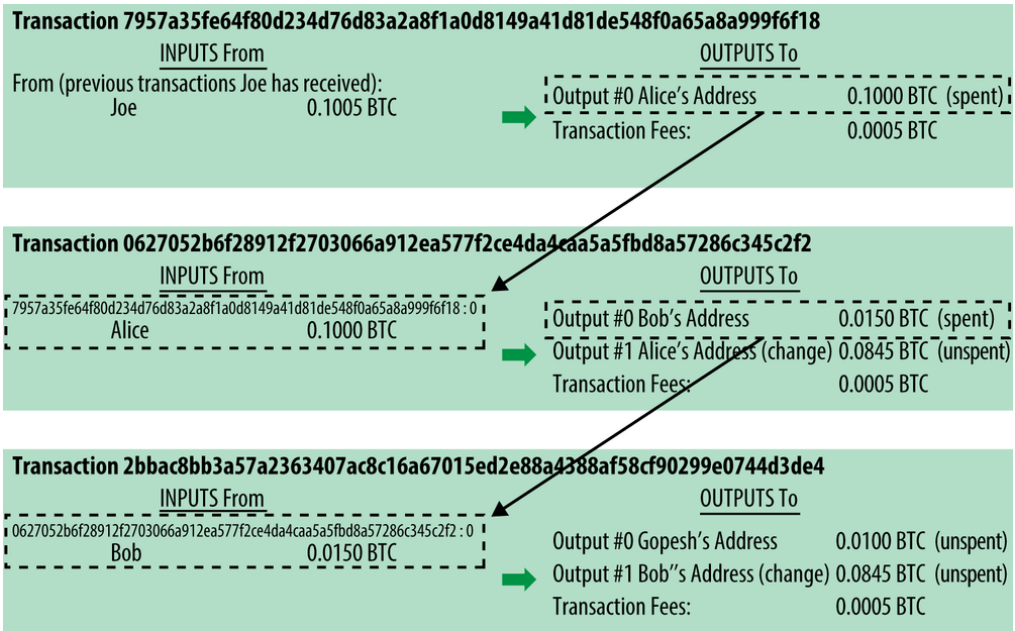

Figure 2.1: Transactions with their inputs and outputs [5]

2.2.2 Inputs

An input is the way an output is redeemed. Specifically, it contains three things:

- The hash of the transaction where the output of interest is contained.
- An index clarifying which output in the transaction this input is referring to.
- A signature used for the validation of the output script.

As a convention, when we talk about the value of an input we mean the value of the output it redeems.

2.2.3 Outputs

An *output* is a tuple (value, pubKeyScript). The value refers to an amount of Bitcoin in Satoshi (where 10^8 Satoshi $= 1$ ฿) and pubKeyScript is a boolean check in order for value to be transferable (see section 2.2.11).

2.2.4 Blocks

A block is a collection of transactions. A valid block satisfies the following:

- there are no double spends (all the inputs are unique)
- Each transaction is included once

The block id is the digest (SHA256) of the block data.

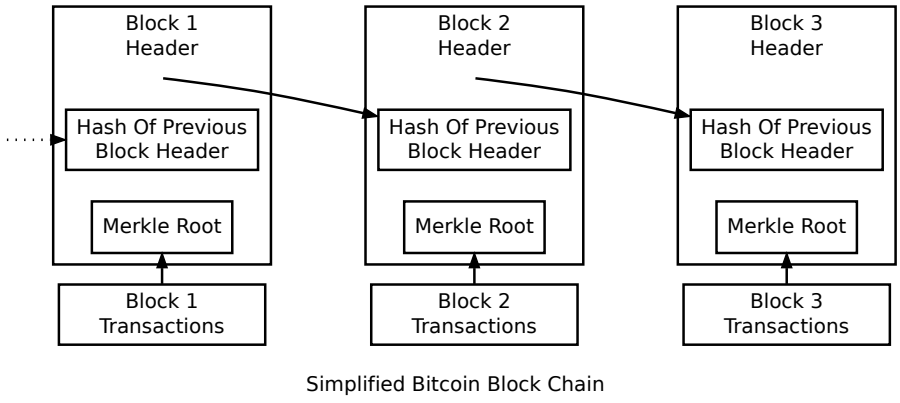

Figure 2.2: The block structure [58]

A block header contains mainly the hash of the previous block, a Merkle root hash (see section 2.2.5) to commit to a set of transactions, and a nonce. Blocks are always referenced by the hash of their block header. Once a transaction has been included in a valid block it's called *confirmed*.

2.2.5 Merkle Trees

In this section we will describe a data structure needed by the meticulous reader in order to understand the description of the Bitcoin protocol (see chapter 2.2). A Merkle tree [53] is a data structure which allows a party to commit to a set of items using only a single hash and prove the inclusion of any item in the committed set, by providing a logarithmic proof in terms of the cardinality of the set.

More specifically, the hashes of the items consist the leaves of the tree and the last level. The internal levels are defined recursively as follows: To create level $k-1$ each pair of level k, (A, B), is transformed as a node of value $H(A||B)$ which points to both A and B for some hash function $H(\cdot)$. If the number of nodes at level k is odd, the last node at that level is paired with itself[2].

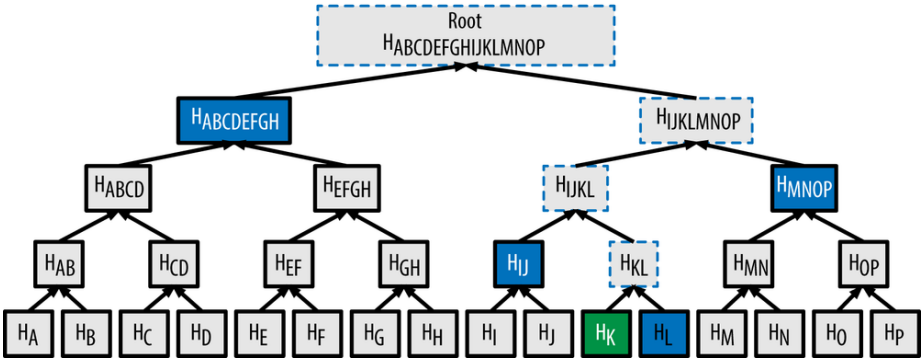

Figure 2.3: A Bitcoin Merkle tree. Source: [5]

Merkle trees are useful in Bitcoin in order to commit to a set of transactions that will be included in a block while keeping the block header of a constant size.

To provide proof of inclusion, all a prover has to do is provide a path of siblings up to the root siblings and a bit vector left indicating whether each sibling is on the left or the right. The verification process is shown in Algorithm 2 [46].

Algorithm 2 The Verify algorithm for a Merkle proof

```
1:  function Verify_root(leaf, siblings, left)
2:      currentHash ← leaf
3:      while left ≠ [] do
4:          siblingIsLeft ← left.shift()
5:          if siblingIsLeft then
6:              currentHash ← H(siblings.shift() || currentHash)
7:          else
8:              currentHash ← H(currentHash || siblings.shift())
9:          end if
10:     end while
11:     return currentHash = root
12: end function
```

[2]This specific construction is the one Bitcoin implements. There are various other constructions which are beyond the scope of our work.

2.2.6 Blockchain

Each block contains a SHA-256 cryptographic hash of the previous block [29], thus linking it to the previous block and giving the blockchain its name. Now, the reader can visualize the famous term *blockchain*. The blockchain is a chain of blocks. The blockchain is public and it holds the history of all valid transactions in a cryptocurrency's network. It holds the timeline of a cryptocurrency's life. It is easy to see that, by the definition of the blockchain, there can be no parallel chains. There is not such thing in economics as two valid transaction histories.

It's possible that there are contending chains of blocks. We then say, there is a *fork* on the chain. On figure 2.4, the chain has forked on blocks 3 and 6.

We call any valid blocks which are not part of our active chain *orphans*. In our example blocks 4b, 7a and 8a are orphans. As expected, orphan blocks, although typically valid, cannot be part of the transaction history. So, transactions that are included in an orphan block (and have not been included in another block yet) return back in the pool (become *unconfirmed*[3]) and are expected to be included in a block in the future.

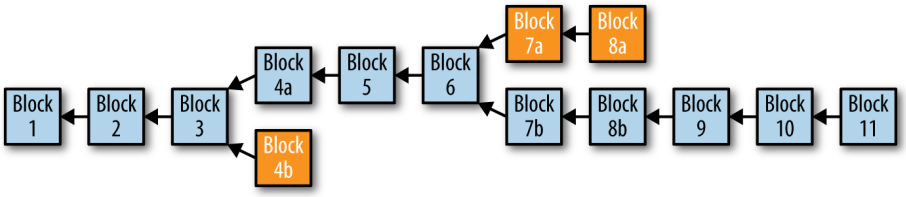

Figure 2.4: A blockchain (the orange blocks are orphans) [5]

2.2.7 Mining

We need to maintain a chronological sequence of transactions. In order to avoid several types of attacks we need everyone to agree in some common transaction history. This agreement is called *consensus*. Mining[4] is the processing of transactions in the digital currency system, in which the records of some cryptocurrency's current transactions, *blocks* (see section 2.2.4), are added to the record of past transactions, the *blockchain* (see section 2.2.6). Miners keep the blockchain consistent, complete, and unalterable by repeatedly grouping newly broadcast transactions into a block, which is then broadcast to the network and verified by recipient nodes [29].

A block contains a list of transactions, the first of which is called the *coinbase transaction* which is where value creation happens in Bitcoin. The miner crafts this transaction granting them some amount of Bitcoins and this transaction is going to be valid only if the block turns out valid. The amount of the *coinbase transaction* is fixed by the Bitcoin protocol. This is a way Bitcoin uses to incentivize miners. However, this doesn't mean that anyone can generate Bitcoin out of thin air: we'll see shortly how it actually comes at a cost with Proof of Work (PoW) (see section 2.2.8).

[3] Note: The reader may find this peculiar. However, we observe that as the blockchain grows, older blocks become "safer" (less probability to become orphans). That means that after a transaction becomes confirmed (included in a valid block), one should wait until this valid block is "safe enough".

[4] It is misleading to think that there is an analogy between gold mining and cryptocurrency mining. The fact is that gold miners are rewarded for producing gold, while some cryptocurrency's miners are not rewarded for producing cryptocoins; they are rewarded for their record-keeping services.

2.2.8 Proof of Work (PoW)

The key to making Bitcoin decentralized is a technique called Proof of Work (PoW). Proof of Work was first invented in 1992 by Dwork et al. [27] as a measure of limiting email spam and denial of service attacks and later explored by Back [8] as Hashcash.

We'll examine a simplified model of Hashcash in order to explore the idea. Suppose we want to send an email to someone. In order to prove we've done work, we include a header (like `X-Hashcash`), which includes the receiver's email address, and a nonce[5]. The nonce is picked so that the hash of the header $H(email||nonce)$ has its 20 most significant bits be all 0. The only feasible way to find this is by brute-forcing the nonce. Once the sender has found the nonce, it's included in the header and sent.

The receiver can then very easily check whether the header hashes to a valid value. If that's so, the email it contains belongs to the receiver and the header is not being reused. After this confirmation, the email can be considered not spam.

To reiterate, the idea is having a series of data to commit to and a hole for the nonce, which is brute-forced to satisfy a necessary predicate on the hash, specifically that its n most significant bits are all zeroes. This is exactly how Bitcoin implements Proof of Work. Instead of the hole being on an email header the hole is on the block header. For a block to be valid, its header has to satisfy a predicate like the above.

Bitcoin introduces a couple of differences. n varies according to the block generation rate. Specifically, to translate the previous predicate to Bitcoin terminology, the hash of each block header has to satisfy

$$H(\mathsf{blockHeader}) \leq T \tag{2.8}$$

where T is called the *target*. As the target goes up, the probability of being below it goes up and generating a valid block is easier. Conversely, if the target goes down it's harder to generate a valid block. To express this, in Bitcoin, the value $\frac{1}{T}$ is called the *difficulty*.

To account for the block generation rate, which Bitcoin tries to keep to 1 block per 10 minutes, every 2016 blocks the target (and subsequently the difficulty) is adjusted accordingly. The target is calculated inside the Bitcoin software and is only a function of the blocks previously seen (frequently called their *view*). So, as long as the Bitcoin nodes agree on the view, they'll agree on the target and all will consider the same set of incoming blocks as valid.

2.2.9 Simplified Payment Verification (SPV)

The size of the blockchain has reached 197GB by the beggining of January 2019, which makes it a very time consuming or even infeasible process to synchronise a full node. Fortunately, a solution was proposed in the original white paper [58], which allows the creation of so-called *lite nodes*.

Lite nodes only know the headers of the entire blockchain, which are constant-size for each block (80 bytes). At the time of writing of this thesis, the size of all block headers was ∼45MB. The lite node then asks the network for transactions concerning it (e.g. transactions concerning a specific public key). Full nodes of the network find such transactions and return them to the requester. For each transaction, the block header of the block it is included in, is returned along with a Merkle tree (see section 2.2.5) proof of inclusion which the lite node can then verify.

[5] Hashcash headers actually contain 7 different fields which have been omitted here for simplicity. The simplified version explained here is not making the same security guarantees as Hashcash.

This protocol is reliable, as long as an adversary does not control the network of a lite node.

2.2.10 Smart contracts

The idea of the smart contracts was first proposed by Nick Szabo [71]. The proposal was about a computerized transaction protocol that executes the terms of a contract. A set of promises, specified in digital form, including protocols within which the parties perform on these promises. On blockchain, this idea can expand in a general purpose computation.

The reader may be familiar with the notion of smart contracts because of the popular implementation in Ethereum blockchain. However, we should note here that the original smart contract language is Bitcoin!

In the next section we will analyze the basic use of the Bitcoin Scripts.

2.2.11 Scripts

Bitcoin offers much more than just moving currency around. It allows us to actually move currency conditionally, where the condition can be expressed as a *Bitcoin script*. Bitcoin script is a stack-based language. An example of a Bitcoin script can be seen on figure 2.5.

```
OP_HASH256
6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000
OP_EQUAL
```

Figure 2.5: A Bitcoin script [46]

This script introduces two kinds of formats. The first kind is commands prefixed with `OP_`. These operations are called *opcodes* and they perform calculations on values on the stack. The result is pushed again to the stack. The types of the calculations are intuitive, e.g. `OP_HASH256` calculates the SHA256 hash of the value on the top of the stack, `OP_EQUALS` compares the top 2 values on the stack and pushes 1 if they are indeed equal or 0 otherwise. The second kind is hex values. These values are simply pushed to the stack. Usually they will be used as input for some operation.

It is easy for the reader to see that the script of figure 2.5 checks if the value on the stack is the preimage of the given hash value and returns 1 (true) or 0 (false). More information about Bitcoin scripts and details about the stack operations can be found in [13]. In practice, this output confirms the success of the evaluation. Such a script is called a pubKeyScript. However, in our example we assume the preimage was on the stack. The way this is implemented in Bitcoin is running another script called scriptSig that passes the parameters to pubKeyScript. The combination of these two scripts is enough powerful and the calculations they perform can be used to make a transaction in the Bitcoin network.

2.2.12 P2PKH

Now, let's see the standard script for conventional fund transfer in Bitcoin, called *pay to public key hash* (P2PKH). Two types of payment are referred as P2PK (pay to public key) and P2PKH (pay to public key hash). Satoshi later decided to use P2PKH instead of P2PK for two reasons:

- Elliptic Curve Cryptography is vulnerable to a modified Shor's algorithm for solving the discrete logarithm problem on elliptic curves. That means, that in the future a quantum computer might be able to retrieve a private key from a public key. By publishing the public key only when coins are spent (and assuming that addresses are not reused), such an attack is rendered ineffective.
- With the hash being smaller (20 bytes) it is easier to print and easier to embed into small storage mediums like QR codes.

A Bitcoin address is only a hash, so the sender can't provide a full public key in pubKeyScript. When redeeming coins that have been sent to a Bitcoin address, the recipient provides both the signature and the public key. The script verifies that the provided public key does hash to the hash in pubKeyScript, and then it also checks the signature against the public key. The reader can see the process in detail in table 2.1.

This is the standard script for conventional fund transfer in Bitcoin. Let's say we want to make sure only Bob can satisfy this script. The pubKeyScript is the following:

```
OP_DUP OP_HASH160 <Bob's address> OP_EQUALVERIFY[6] OP_CHECKSIG
```

The scriptSig is then typically `<Bob's signature> <Bob's public key>`.

Bob's signature will be available. We will see that it can be found on the hash of the transaction containing the output. The script will then duplicate his public key, check that it matches the one on the pubKeyScript and if it does, it will check that he has provided a valid signature with that public key. If all these checks pass, the stack will end up with 1 on top and the execution will be valid.

2.2.13 Theoretical model

A deeper analysis and security results on the mathematical model of the blockchain technology can be found in two very important papers called "The Bitcoin backbone protocol: Analysis and Application" [37] and "The Bitcoin backbone protocol with chains of variable difficulty" [38]. We definitely refer the reader to the above work, if he/she wants to aquire a deep understanding of the blockchain structure.

[6] This operation is a lot like `OP_EQUAL` but instead of pushing 1 or 0 to the stack, it fails the script if the arguments are not equal or does nothing otherwise.

Stack	Script	Description
Empty	<sig><pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	scriptSig and scriptPubKey
<pubKey> <sig>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Constants added to stack.
<pubKey> <pubKey> <sig>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item duplicated.
<pubKeyHashA> <pubKey> <sig>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item hashed.
<pubKeyHash> <pubKeyHashA> <pubKey> <sig>	OP_EQUALVERIFY OP_CHECKSIG	Constant added.
<pubKey> <sig>	OP_CHECKSIG	Equality check between the top two stack items.
True	Empty.	Signature is checked for top two stack items.

Table 2.1: Bitcoin script process (successful) [13]

2.3 Egalitarian Mining

To be accepted by the rest of the network, a new block must contain a Proof of Work (PoW) (see section 2.2.8). The PoW requires miners to find a number called a nonce, such that when the block content is hashed along with the nonce, the result is numerically smaller than the network's difficulty target [58] and thus the PoW equation (see equation 2.8) is satisfied. This proof is easy for any node in the network to verify, but extremely time-consuming to generate, as for a secure cryptographic hash, miners must try many different nonce values before meeting the difficulty target.

The primary purpose of mining is to set the history of transactions in a way that is computationally impractical to modify by any one entity. By downloading and verifying the blockchain, nodes are able to reach consensus about the ordering of events in some proof of work cryptocurrency [13].

As we noted in section 2.2.8, every 2,016 blocks the difficulty target is adjusted based on the network's recent performance, with the aim of keeping the average time between new blocks at ten minutes. In this way, the system automatically adapts to the total amount of mining power on the network. Between 1 March 2014 and 1 March 2015, the average number of nonces, miners had to try before creating a new block, increased from 16.4 quintillion to 200.5 quintillion [15].

The proof of work system, alongside the chaining of blocks, makes modifications of the blockchain extremely hard, as an attacker must modify all subsequent blocks in order for the modifications of one block to be accepted. As new blocks are mined all the time, the difficulty of modifying a block increases as time passes and the number of subsequent blocks (also called confirmations of the given block) increases [29].

Mining is also the mechanism used to introduce coins into the system: Miners are paid any transaction fees as well as a "subsidy" of newly created cryptocoins. This both serves the purpose of disseminating new cryptocoins in a decentralized manner as well as motivating people to provide security for the system [13]. To elaborate on the second part of this purpose, one can think about the structure of the consensus on the network. The network agrees by majority. So an attacker who controls 51% of the mining power can successfully attack this structure. As more honest miners contribute to the network, the 51% attack becomes less feasible.

Originally, Bitcoin mining was conducted on the CPUs of individual computers, with more cores and greater speed resulting in more profitability. After that, the system became dominated by multi-graphics card systems, then field-programmable gate arrays (FPGAs) and finally application-specific integrated circuits (ASICs), in the attempt to find more hashes per hour with less electrical power usage (see figure 2.6).

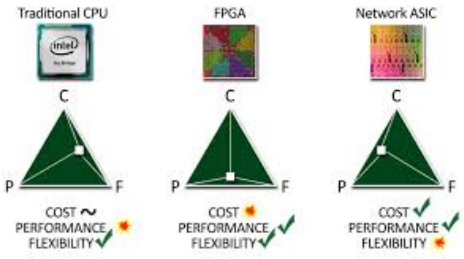

Figure 2.6: Mining options

Due to this constant escalation, it has become hard for prospective new miners to start. This adjustable difficulty is an intentional mechanism created to prevent inflation. To get around that problem, individuals often work in *mining pools*. Mining pools are groups of miners who join their collective computational power and share their profit according to the contribution of each party.

Bitcoin generally started with individuals and small organizations handling the mining. At that time, start-up could be enabled by a single high-end gaming system. However, nowadays larger mining organizations might spend tens of thousands on one high-performance, specialized, application-specific integrated circuit.

That creates a problem. In a system, which since its creation is supposed to distribute power among users, there has been a great power concentration in the hands of big companies, like Bitfury or 21, that develop ASICs to mine Bitcoin. Because of the extreme cost of ASICs and extreme hashrate, someone who uses a multi-graphics card system or a CPU is out of competition. As a result, independent miners have largely dried up.

2.3.1 Egalitarianism

Let's consider several contexts where an adversary has an upper hand over the defender, by using special hardware in an attack. These include password processing, hard-drive protection, cryptocurrency mining, resourse sharing, code obfuscation, etc. Memory-hard computing is a generic paradigm, which can protect the defender against attacks in the aforementioned contexts. Every task is amalgamated with a certain procedure requiring intensive access to RAM, both in terms of size and bandwidth, so that transferring the computation to GPU, FPGA, and even ASIC brings little to no cost reduction.

Cryptographic schemes that run in this framework become *egalitarian* in the sense that both users and attackers are equal in the price-performance ratio conditions. When the cryptographic scheme is a hash function used for cryptocurrency mining, we refer to this notion as *egalitarian mining*.

But let's step back a little and think about the need for such a notion. Do we actually need it? Is egalitarian mining a way to destroy competition? Is it unfair? Shouldn't a miner be rewarded for the extra money he invested?

Many questions like the above have been asked and usually the answer is not descriptive enough of what really memory-hardness introduces to the world. We will try here to demonstrate concretely what it means for a cryptocurrency to offer egalitarian mining.

Egalitarian mining does not destroy competition. The miner who invests more in hardware is rewarded more. Each individual miner is rewarded according to the computation power he offers to the community. The real difference is that it is really easy for people to start mining with a single high-end gaming system. Hobbyists, who want to support the community are welcome to mine. In Bitcoin system, this option is not available. In order to support the community by mining, you have to invest a lot of money on ASICs to be competitive. This means that, in general, the mining to support the Bitcoin project or for fun is dead.

This is hurtful for a system, which by design is supposed to bring decentralization in the financial market. Because, without hobbyists, we are actually left with big companies handling almost all of the mining. Companies will comply with regulations that the government of each country enforces and cannot be expected to react and inspire political movements. Since countries can and they have, historically, collaborated against threats, a union of countries who can enforce regulations to companies that control more than 51% of the hashing power, can bring a cryptocurrency to its knees, if seen as a threat. That scenario does not fit in most definitions of security.

One of the reasons that cryptocurrencies have a bootstrapping period is because they need a big support community to distribute mining in order to guarantee security. When the total hashing power is a few high-end gaming systems, acquiring 51% of the hashing power is feasible. As the support expands, the security is satisfied for all practical purposes. But when mining is dominated by companies, then a totally trustless system gives birth to a trusted party. That's against the motivation for the inception of a cryptocurrency and it raises questions like "*Why should I trust the mining companies and support this cryptocoin? Do I trust my bank more? After all, my bank is just another company...*"

Formal definition

Now the reader should have a good understanding about the notion of *egalitarianism*. However, the claims for egalitarianism in several cryptocurrencies have been hand wavy and no such claim is accompanied by exact data. Egalitarianism was a vague and undefined term until quite recently. In 2019, Dimitris Karakostas, Aggelos Kiayias, Christos Nasikas and Dionysis Zindros published a paper [45] aiming to end this era of ambiguity. They presented a quantitative definition for this term and set the basis for future work in this direction.

As a means towards establishing their definition, they define the *egalitarian curve* f of a cryptocurrency. Reproduced from [45]:

> *The horizontal axis of this curve plots the financial capital which is available for investment denominated in a fiat currency*[7]*, USD. The vertical axis plots the Return On Investment (ROI), which measures the cryptocurrency amount that is freshly generated in the investment period and remains unspent at the end of the investment period, given an optimal allocation of the initial capital.*

They continue with the necessary definition of the *egalitarian curve* in order to prepare a concrete and sound definition of the term *egalitarianism*. We reproduce here the two definitions. Again, from the [45]:

Definition 2.11 (Egalitarian curve)**.** Given a cryptocurrency c, an investment period interval d, the set of all possible investment strategies $\mathcal{B}$, we define the *egalitarian curve* $f_{c,d} : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ of c for investment period d as:

$$f_{c,d}(v) = \frac{\max_{B \in \mathcal{B}} \mathbb{E}[B(v)] - v}{v}$$

The value $\max_{B \in \mathcal{B}} \mathbb{E}[B(v)]$ identifies the maximum expectation of returns across all investment strategies $\mathcal{B}$, i.e., the amount of returns which the *optimal* strategy ensures for a given initial capital v.

[7] Fiat currency is legal tender whose value is backed by the government that issued it. The U.S. dollar is fiat money, as are the euro and many other major world currencies. A fiat currency's value is underpinned by the strength of the government that issues it, not its worth in gold or silver.

Now, we are ready to reproduce their definition of the notion of interest:

Definition 2.12 (Egalitarianism)**.** Given a cryptocurrency c, an investment period duration d and an initial capital distribution $\mathcal{D}$, we define the *egalitarianism* e of c for investment duration d under initial capital distribution $\mathcal{D}$ as follows:

$$e_{c,d,\mathcal{D}} = -\mathsf{Var}_{v \leftarrow \mathcal{D}}[f_{c,d}(v)]$$

where f is the egalitarian curve of c.

As they remark in their paper, the intuition behind this definition is that, to have egalitarianism, the ROI must remain the same across different capital investments. As such, any deviation from the mean is non-egalitarian. For further reading the reader is refered to their work [45].

We are focusing on mining, but one should think about the possibilities of a Proof of Work (PoW) (see section 2.2.8) mechanism in order to understand the contribution of a memory-hard hash function. The PoW mechanism is actually a voting system. Users vote for the right order of the transactions, for enabling new features in the protocol and for the honest money supply distribution. Therefore, it is important that during the voting process all participants have equal voting rights.

To sum up, for security and decentralization arguments, it is healthy for some cryptocurrency's mining power to be distributed among users. Memory-hardness sustains the competition, but it makes it less harsh and keeps the door open for hobbyists to support the community. It is extremely difficult for the corporate mining to acquire tremendous power for two reasons and that is essential in a trustless system that aims to remain trustless. The reasons are:

(α) It is not that lucrative for companies. If someone makes a big investment, he will get big rewards but not insanely huge rewards leaving every hobbyist out of the mining community.

(β) Even if a lot of companies decide to participate, it is difficult for them to acquire a combined 51% of the total hashing power (It is no more about what devices you buy, just how many).

Memory-hardness defends a system against the aforementioned prospect and thus strengthens the notion of any PoW cryptocurrency's security.

CHAPTER 3

MONERO

> They who can give up essential liberty
> to obtain a little temporary safety
> deserve neither liberty nor safety.
>
> *Benjamin Franklin*

3.1 Introduction

Monero (XMR) is a decentralised open-source cryptocurrency. The project's fundamental feature is privacy - it aims to be a digital medium of exchange with untraceable payments, unlinkable transactions and resistance to blockchain analysis. The parties behind a Monero transaction are not known; this results in considerable increase of privacy compared to Bitcoin and its forks [73].

3.2 History

First, the construction was outlined in an October 2013 white paper by the pseudonymous figure Nicolas van Saberhagen and called CryptoNote protocol [65]. Later, in 2014, Bitcointalk forum user known as `thankful_for_today` forked the codebase of *Bytecoin* (CryptoNote's reference implementation) into the name *BitMonero*, which is a compound of bit and monero (literally meaning *coin* in Esperanto [32]).

The release of BitMonero was very poorly received by the community that initially backed it. Plans to fix and improve Bytecoin with changes to block time, tail emission and block reward had all been ignored, and `thankful_for_today` simply disappeared from the development scene. A group of users led by `Johnny Mnemonic`[1] decided that the community should take over the project and five days later they did, while also changing the name to *Monero*.

[1] Fun fact: reference to a 90's cult film character, incarnated by Keanu Reeves, who could store data into his mind and worked as a data courier.

Due to its privacy features, Monero experienced rapid growth in market capitalization and transaction volume during the year 2016, faster and bigger than any other cryptocurrency that year. This growth was driven by its uptake in the darknet market. From the beginning, Monero has been used by people holding other cryptocurrencies like Bitcoin to break the link between transactions, with the other cryptocoins first converted to Monero, then after some delay converted back and sent to an address unrelated to those used before.

On January 10, 2017, the privacy of Monero transactions was further strengthened by the adoption of Bitcoin Core developer Gregory Maxwell's algorithm *Confidential Transactions* [59], hiding the amounts being transacted, in combination with an improved version of *Ring Signatures*.

In late 2017, malware and antivirus service providers blocked a JavaScript implementation of Monero miner *Coinhive* [20] that was embedded in websites and apps. Coinhive generated the script as an alternative to advertisements; a website or app could embed it and use website visitor's CPU to mine the cryptocurrency, while the visitor was consuming the content of the webpage.

However, some websites and apps did this without informing visitors and some hackers implemented it in a way that drained visitors' CPUs. As a result, the script was blocked by companies that offer ad blocking subscription lists, antivirus services and antimalware services.

Monero is actively encouraged to those seeking financial privacy, since payments and account balances remain entirely hidden, which is not the standard for most cryptocurrencies.

3.3 Specifications

Monero is [73]:

Untraceable Monero uses a digital signature scheme called *ring signatures* [59], which shuffles users' public keys in order to eliminate the possibility to identify a particular user.

Unlinkable Monero employs a specific protocol which generates multiple unique one-time addresses that can only be linked by the payment receiver and are unfeasable to be revealed through blockhain analysis.

Secure Monero is cryptographically secured. Moreover, the design of the algorithm used consists in tremendous computational and electric capabilities, that an adversary would need to even try to steal funds.

Private Privacy is basically provided by the idea of anonymous transactions without any obligations to cooperate with third parties.

Analysis Resistant Monero's blockchain analysis resistance results from unlinkability, which is achieved by using a modified version of the Diffie-Hellman exchange protocol [26] that generates multiple one-time public addresses that can only be simply gathered by the message receiver, but hardly analyzed by confused foreigners inside the block explorer.

3.3.1 Account

In Monero, a wallet is called an account and it is a private account owned and operated by a Monero user. An account contains all of the Monero transactions a user has sent and received. Some user's account balance is a sum of all the Monero received, less the Monero sent.

A Monero account has two balances, a locked and an unlocked balance. The unlocked balance contains funds that can be spent immediately, and the locked balance contains funds that can't be spent right away. A Monero user may receive a transaction that has an unlock time set or he/she may have sent some Monero and is waiting for the change to come back to his/her wallet, both of which situations could lead to those funds being locked for a time.

An account resides only under user's control, normally on his/her computer, and cannot be accessed by anyone else if he/she practices good security [40].

3.3.2 Keys

A Monero account is based on two keys. They are called *spend key* and *view key*. The *spend key* is special in that it is the single key required to spend your Monero funds, whereas the *view key* allows you to reveal your transactions to a third party. That makes sense in case of auditing or accounting purposes.

The spine of the Monero project is the CryptoNote protocol. All the above specifications are based on ideas that exist in the CryptoNote white paper [65]. Monero is the most successful implementation of this protocol, among numerous efforts (CryptoNoteCoin, Bytecoin, AEON, etc. [22]). Describing every implementation is impractical and beyond the scope of this thesis.

However, it would be an inexcusable omission not to describe the features and specifications of the CryptoNote protocol itself. For our purposes, we will illustrate the above with Monero project in mind and especially one specific element, the CryptoNight function (see chapter 4), which is the feature of interest in this thesis.

Unlike many cryptocurrencies that are derivatives of Bitcoin, Monero uses a proof of work mechanism to issue new coins and incentivize miners to secure the network and validate transactions. One key part, for Monero project to offer the above, is a proof-of-work algorithm called CryptoNight, developed by the CryptoNote project [65]. On top of typical security attributes, this algorithm is also suspected to be memory-hard. The aim of this work is to study the memory-hardness property (see section 2.1.3) of this algorithm.

3.4 CryptoNote

The *CryptoNote Technology* is designed to provide some of the most innovative privacy features predicated on advanced cryptography, an egalitarian approach towards decentralization and censorship-resistance. CryptoNote, as described in the Bitcoin forum [12], is the technology that allows creation of privacy-centric cryptocurrencies. The level of anonymity provided by CryptoNote isn't possible with Bitcoin code base by design. *Bytecoin (BCN)* was the CryptoNote reference implementation, and *Monero (XMR)* is based on BCN's code.

The CryptoNote protocol possesses significant algorithmic differences relating to blockchain obfuscation. One of the main features of CryptoNote, are *ring signatures* [65] that mask sender identities by mixing them and one-time keys that make transactions unlinkable. Their combined effect gives a high degree of anonymity without any extra effort on the part of the user.

Unlike Bitcoin, a user's funds are not held in the address he/she gives out to others. Instead, every time he/she receives a payment it goes to an unlinkable address generated with random numbers. When he/she decides to spend the funds in that one-time address, the amount will be broken down and the components will be indistinguishable from identical outputs in the blockchain.

For example if 556.44 XMR are sent, the protocol will break it down into 500 + 50 + 6 + 0.4 + 0.04 and a ring signature will be performed with other 500's, 50's, 6's, 0.4's, and 0.04's in the blockchain. Unlike the *CoinJoin* mixing method [12], CryptoNote mixes outputs not transactions. This means no other senders need to be participating with some user at the same time or with the same amounts. Any arbitrary amount sent at any time can always be rendered fundamentally indistinguishable (a mathematical proof is given in the white paper [65]).

The degree of anonymity is also a choice rather than decided by the protocol: do you want to be hidden as one among five or one among fifty? The size of the signature grows linearly as $\mathcal{O}(n+1)$ with the ambiguity, so greater anonymity is paid for, with higher fees to miners.

3.4.1 Untraceable transactions

CryptoNote cryptographic scheme relies on the cryptographic primitive called a *group signature*. First presented by D. Chaum and E. van Heyst [18], it allows a user to sign his message on behalf of the group. The idea is actually simple. After signing the message the sender provides the keys of all the users of his group. A verifier is convinced that the real signer is a member of the group but cannot be exclusively identified.

However, this primitive required a trusted third party (Group Manager) who could trace the signer. The *ring signature* was introduced by Rivest et al. [63] and it was an autonomous scheme without anonymity revocation. Based on this work various modifications arose like *linkable ring signature* [51, 50, 6], a scheme that allowed to determine if two signatures were produced by the same group member, *traceable ring signature* [35, 36], a scheme that limited anonymity (it is possible to trace the signer of two messages) and *ad-hoc group signature* [1, 78]. The last scheme focuses on the arbitrary group formation. The other schemes rather imply a fixed set of members.

Based on [36] with a few modifications, in CryptoNote white paper [65] is presented the *one-time ring signature*. They weakened the traceability property and kept the linkability. That is needed because they wanted some user's public key to appear in many foreign verifying sets and from the private key to generate a unique anonymous signature. In case of a double spend attempt, these two signatures will be linked together. However, revealing the signer is not necessary.

Ring signatures are explained below. We will start with a normal signature scheme shown in figure 3.1. Reproduced from CryptoNote [23]:

Figure 3.1: Normal signature: One participant, which allows one-to-one mapping. [23]

In figure 3.2 we show the ring signature concept.

Figure 3.2: Ring signature: Only proves that a signer belongs to a group. [23]

The result is shown in figure 3.3. The reader can think of it as decentralized and trustless mixing.

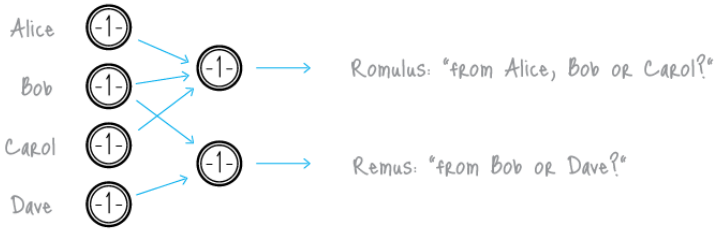

Figure 3.3: High level of anonymity in cryptocurrency transactions. [23]

For an example of a complete CryptoNote transaction the reader is refered to section 3.4.6. Due to figure's size and complexity, it was improbable for us to manage to describe this example here, without compromising readability.

3.4.2 Unlinkable transactions

First, we should clarify the problem which is solved with unlinkability. Even if a transaction is untraceable, when the receiver posts his/her public address anyone can check all his/her incoming transactions (see figure 3.4). A naive solution is to create a bunch of keys and addresses that can be sent privately to the payers (one distinct key per payer). This approach is highly problematical since it:

a) Deprives the receiver of the convenience of having a single public address

b) Implies that the default use of the structure *does not* create unlinkable transactions

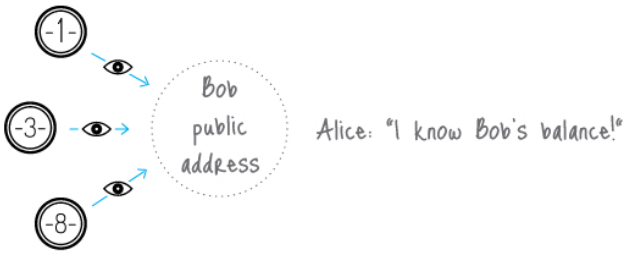

Figure 3.4: Linkable transactions. [23]

CryptoNote solves this problem. It creates automatically and by default multiple unique one-time keys, derived from the single public key, for each peer-to-peer payment. The solution lies in a clever modification of the Diffie-Hellman exchange protocol [26]. Originally, it allows two parties to produce a common secret key derived from their public keys. In CryptoNote protocol the sender uses the receiver's public address and his own random data to compute a one-time key for the payment.

The sender can produce only the public part of the key, whereas only the receiver can compute the private part; hence the receiver is the only one who can release the funds after the transaction is committed. He/she only needs to perform a single-formula check on each transaction to establish if it belongs to him/her. This process involves his/her private key, therefore no third party can perform this check and discover the link between the one-time key generated by the sender and the receiver's unique public address.

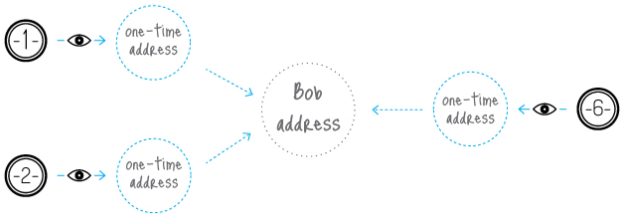

Figure 3.5: Unlinkable transactions. [23]

```
vJmsp9MxWMj6jiUg8Rejh23pqRCthWQhwtUKvmLw2kcE83AHer1MchTN4DVacHt
43r8hSKBQpjPuqYDKuKgyVBkGkUdcsNAdnk2aZW
```

Figure 3.6: Bitcoin stealth address.

An important part of CryptoNote is the use of random data by the sender. It always results in a different one-time key, even if the sender and the receiver both remain the same for all transactions (that is why the key is called ”one-time”). Moreover, even if they are both the same person, all the one-time keys will also be absolutely unique.

Stealth addresses

The structure of the above concept is inherited in all CryptoNote projects. But the details of each implementation may differ. What we will present here is the implementation details of unlinkability, as found in the Monero project. The reader can find all the information of this section and more in CryptoNote paper [65] and Monero project's code [55]. An additional valuable source of information and interactive conversation venue that the reader is refered to is Monero stack exchange forum [56]. The implementation of CryptoNote's unlinkable transactions in Monero project is mentioned as *stealth addresses* or *subaddresses*.

Stealth address technology originated from CryptoNote technology, but Bitcoin (e.g. *libbitcoin*) and its altcoins can also implement stealth addresses. For Bitcoin and its altcoins, stealth addresses must be explicitly supported by the sender's and recipient's wallets, but such support is implicit to CryptoNote wallets.

For Bitcoin, stealth addresses are a bit longer than normal Bitcoin addresses (see figure 3.6). However, the transactions associated with a stealth address looks no different than normal transactions on the Bitcoin blockchain. Stealth addresses contain one public *view* (in CryptoNote vernacular) or *scan* (in Bitcoin vernacular) key, and one or more *spend* public keys. These keys are always encoded in a stealth address to support the first portion of Diffie-Hellman key exchange [26]. Bitcoin's public/private key pairs are derived from the *secp256k1* elliptic curve [70], while CryptoNote uses *Ed25519* [9] (see section 3.4.6) derived public/private key pairs.

One can publish their stealth address on a business card, and sustain their privacy when funds are sent to a dynamically computed destination address by the sender of funds. Stealth addresses essentially put the onus of dynamic address calculation, typically associated with a recipient's hierarchical deterministic (HD) wallet, on the sender's wallet. One stealth address is functionally akin to an HD wallet account, and can thus be used over and over for many fund transfers. Stealth addresses provide confidentiality for the recipient of transaction pairs that utilize information from a stealth address.

So, simplifying a bit, in Bitcoin if there is one Bitcoin associated to the public key P and if Bob knows the corresponding private key x such that $P = xG$[2], then he can spend the Bitcoin by submitting a message (transaction) to the network signed with x.

[2] G is the generator for the algebraic ring that is the base of the key construction. For our purpose, it suffices to think G as a parameter of the user's existence in the Monero network, which is public and thus available to anyone.

There is one privacy issue, though: if Bob keeps using the same P to receive Bitcoins, then any observer will be able to see all payments were made to the same entity that controls P (Bob). This is the problem that stealth addresses solve.

Stealth addresses innately perform the first half of a Diffie-Hellman key exchange [26] when a sender of funds receives a stealth address. In Bitcoin, two blockchain transactions are required to complete the sending of funds to a stealth address that belongs to the recipient of funds.

Since it is operationally improbable for two users' wallets to communicate directly to each other, the first transaction is a persistent `OP_RETURN` transaction that is used to complete the second half of a Diffie-Hellman key exchange. The second transaction is the actual sending of funds to a dynamically calculated destination address that is strongly based upon an ephemeral random number generator in the sender's wallet.

The second half of the Diffie-Hellman key exchange, the `OP_RETURN`, allows the recipient's wallet of a stealth transaction to dynamically calculate the private redemption key associated with a particular transaction to redeem the funds at a later date.

Stealth addresses can be extended to support multisig. This is a multisig capability that is more inherent to Bitcoin than CryptoNote.

In the context of stealth addresses, addresses are now composed of two public keys, and the coins sent to Bob will not be sent to his stealth address on the blockchain, rather the stealth address will be used by the sender to produce fresh new Bitcoin addresses for every new transaction. These new addresses, even though generated by the sender (Alice) and unknown to Bob until the transaction is made, will nonetheless be controlled by Bob! Here is how it works:

Bob creates two pairs of private and public keys. Let's denote them by (a, A) and (b, B), where by definition

$$A = aG \quad \text{and} \quad B = bG \tag{3.1}$$

Bob makes the pair of public keys (A, B) available to the network; this will be his stealth address.

Alice wishes to send one Bitcoin to Bob. She wants to assign one Bitcoin to a public key P such that Bob knows x and $P = xG$. She will construct such P using Bob's stealth address by using a hashing function $\mathcal{H}$, choosing a random big number r, and setting

$$P = \mathcal{H}(rA)G + B \tag{3.2}$$

Then, Alice sends the Bitcoin to P and the transaction is broadcast along with $R = rG$ (but not r, which can't be recovered from R).

In order to get the money, Bob has to keep listening to the network for all new transactions and check whether one or more of these transactions are money that he should receive. When he sees Alice's transaction, he checks if $x := \mathcal{H}(aR) + b$ and realizes that:

$$\begin{aligned} xG &= (\mathcal{H}(aR) + b)G \\ &= \mathcal{H}(aR)G + bG \\ &= \mathcal{H}(arG)G + B \\ &= \mathcal{H}(raG)G + B \\ &= \mathcal{H}(rA)G + B \\ &= P \end{aligned}$$

Bob can reconstruct x such that $P = xG$ and is therefore the owner of the Bitcoin! Notice that neither Alice nor any observer has the ability to derive x (because they don't know a and b), and that besides Alice and Bob no one knows that (x, P) was generated from Bob's stealth address (because they don't know r).

Ignoring middleman snooping on IP addresses associated with stealth address transaction pairs, only the two core parties involved in a transaction pair will know any identity details associated with sending funds to a stealth address. Hence, the need for Kovri I2P technology (see section 3.5.3). So, stealth transactions by themselves don't provide 100% anonymity protection. Also Confidential Transactions (CT) technology is needed by Bitcoin to mask details about the amount transferred by a transaction.

Note that, as mentioned, this protects Bob's privacy, but it is still visible to the network that Alice, the entity that used to control that Bitcoin, made a transaction. In order to obfuscate that action, Monero implements the use of Ring Signatures [59], which will allow Alice to, instead of directly signing the transaction, produce a proof that her or several other people, did send a coin to Bob.

To sum up, in Monero, coins are received to a unique one-time stealth address. The formula for stealth addresses, is as follows:

$$P = \mathcal{H}(rA)G + B \tag{3.3}$$

Where:

G The standard *Ed25519* base point

A Bob's public view key

B Bob's public spend key

r The new random scalar Alice chose for this transaction

$\mathcal{H}$ A hashing algorithm that returns a scalar (i.e., the hash output is interpreted as an integer and reduced modulo l)

P The final stealth address (one-time output key, the destination where funds will actually be sent)

So, in a nutshell:

- *Stealth addresses* take care of **recipient's** privacy.
- *Ring Signatures* take care of **sender's** privacy.

An example is presented in the next section. There, the reader can find a real world construction of a Monero stealth address.

3.4.3 Stealth address construction

Here is a functional example for deriving a Monero stealth address. Here we will examine the developer mechanics, not cryptographic theory. Results below duplicate functionality that is part of *Crypto Note Test Address* [24].

It is worth noting custom `bytes_to_words`, `sc_reduce32`, and `secret_key_to_public_key` executables (coded in C or C++) below were named after Monero's functions that yielded output results. C++ coding insights came from `main.cpp`. The `bx` command line is bitcoin-explorer, see [11].

Monero's `secret_key_to_public_key()` functionality is using *Ed25519* (see section 3.4.6) technology but not in an inclusive way. Only the necessary computations for the production of stealth addresses are implemented. Results are different from *Tor* [74] test vectors results that custom executables utilizing *libsodium* and *ed25519-donna* yield, but Monero C/C++ code results match that from *Crypto Note Test Address*. Let us see the components and the calculations that take place in order to construct a stealth address.

The example that is presented here was posted by user `skaht` on Monero stack exchange forum [56]. The calculations were checked and confirmed.

- 256-bit hexadecimal-encoded seed is assumed to be:

```
198584347013dd91832be3d82529437db7cc8e1850e559cdd3872b29
ca819601
```

- Electrum mnemonic words[3] corresponding to seed
  (`./bytes_to_words <above seed>`)

```
$ ./bytes_to_words 198584347013dd91832be3d82529437db7cc8
e1850e559cdd3872b29ca819601
```

Output:

wallets drinks insult popular fall textbook scoop apology unsafe fifteen cuffs pimple roster nerves pixels upstairs academy sprig eclipse leopard peeled faxed gutter happens roster

- Private spend key calculation
  (`./sc_reduce32 <private spend key>`)

```
$ ./sc_reduce32 198584347013dd91832be3d82529437db7cc8e185
0e559cdd3872b29ca819601
```

Output:

198584347013dd91832be3d82529437db7cc8e1850e559cdd3872b29ca819601

[3] These are the words that a wallet owner should remember in order to restore his wallet, if he forgets his password.

- Private view key calculation
  (`./keccak`[4]`<private spend key> | ./sc_reduce`)

```
$ ./keccak 198584347013dd91832be3d82529437db7cc8e1850e559c
dd3872b29ca819601
$ ./sc_reduce32 <the keccak output>
```

Output:

889DA12A88D36BCE0966AB1A79125779DD1F2FC6F1145DE131FD52A5B468796D

faa5defce980fdbd03b9dd4841371dfcdc1f2fc6f1145de131fd52a5b468790d

- Public spend key calculation
  (`./secret_key_to_public_key <private spend key>`)

```
$ ./secret_key_to_public_key 198584347013dd91832be3d82529437
db7cc8e1850e559cdd3872b29ca819601
```

Output:

b66991d7d7c68513533d0560f820d75adfb0911487ba62274b759f7b3ccd4a90

- Public view key calculation
  (`./secret_key_to_public_key <private view key>`):

```
$ ./secret_key_to_public_key faa5defce980fdbd03b9dd4841371
dfcdc1f2fc6f1145de131fd52a5b468790d
```

Output:

3c450f27cd6849d9130addb2c566d910c5ef9bf4cecaed547004496fda52a4ff

Note that the calculation of the stealth address (hexadecimal format) is:

```
prefix + public_spend_key + view_public_key +
                                    keccak_checksum_postfix
```

[4] Hash function.

The `prefix` in Monero addresses is always 12 and is a marking of a Monero address. In this context the character + is used to mark concatenation of strings. The `keccak_checksum_postfix` computation is:

- Stealth address checksum calculation (`./keccak <almost an address>`):

```
$ ./keccak 12 +
b66991d7d7c68513533d0560f820d75adfb0911487ba62274b759f7b3
ccd4a90 +
3c450f27cd6849d9130addb2c566d910c5ef9bf4cecaed547004496fd
a52a4ff
```

Output:

ADD568169DBF2C6D3F595EE8610A189955BECD1EDF150627CBF2F2C49B0AEA71

- So, the hexadecimal format of a Monero stealth address is:

```
12b66991d7d7c68513533d0560f820d75adfb0911487ba62274b759f7
b3ccd4a903c450f27cd6849d9130addb2c566d910c5ef9bf4cecaed54
7004496fda52a4ffADD56816
```

In order to convert a hexadecimal representation of a stealth address in base58 format we calculate the base58 format of each 8 bytes and concatenate the results (presented between brackets). For the conversion, we used `bx` (bitcoin explorer [11]):

1. `$ bx base58-encode 12b66991d7d7c685` (→ `48Y3H2eSZ6C`)
2. `$ bx base58-encode 13533d0560f820d7` (→ `4EUjY1B5viS`)
3. `$ bx base58-encode 5adfb0911487ba62` (→ `GCbCLPcmMiy`)
4. `$ bx base58-encode 274b759f7b3ccd4a` (→ `7aD69yqUsaH`)
5. `$ bx base58-encode 903c450f27cd6849` (→ `R8GLE3rvSwr`)
6. `$ bx base58-encode d9130addb2c566d9` (→ `dJtpZYG1peC`)
7. `$ bx base58-encode 10c5ef9bf4cecaed` (→ `3oipCqfUvCc`)
8. `$ bx base58-encode 547004496fda52a4` (→ `F89i86kuEjV`)
9. `$ bx base58-encode ffADD56816` (→ `Vr5GCdj`)

Finally, we get the Monero stealth address:

48Y3H2eSZ6C4EUjY1B5viSGCbCLPcmMiy7aD69yqUsaHR8GLE3rvSwrdJtpZYG
1peC3oipCqfUvCcF89i86kuEjVVr5GCdj

3.4.4 Double-spending proof

Fully anonymous signatures would allow spending the same funds many times which, of course, is incompatible with any payment system's principles. The problem can be fixed and here we reproduce the description of this solution, as presented in [23].

A ring signature is actually a class of crypto-algorithms with different features. The one CryptoNote uses is the modified version of the *traceable ring signature* [36]. In fact, they transformed traceability into linkability. This property restricts a signer's anonymity as follows: if he/she creates more than one ring signature using the same private key (the set of foreign public keys is irrelevant), these signatures will be linked together which indicates a double-spending attempt.

To support linkability CryptoNote introduced a special marker being created by a user while signing, which they called a *key image*. It is the value of a cryptographic one-way function of the secret key, so in mathematical terms it is actually an image of this key. One-wayness means that, given only the key image, it is impossible to recover the private key.

On the other hand, it is computationally impropable to find a collision (two different private keys, which have the same image). Using any formula, except for the specified one, will result in an unverifiable signature. All things considered, the key image is unavoidable, unambiguous and yet an anonymous marker of the private key.

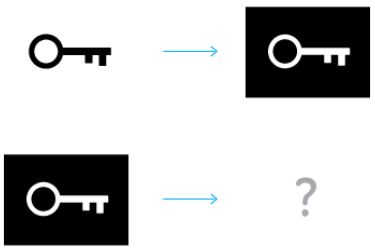

Figure 3.7: Key image via one-way function. [23]

All users keep the list of the used key images (compared to the history of all valid transactions, it requires an insignificant amount of storage) and immediately reject any new ring signature with a duplicate key image. It will not identify the misbehaving user, but it does prevent any double-spending attempts, caused by malicious intentions or software errors.

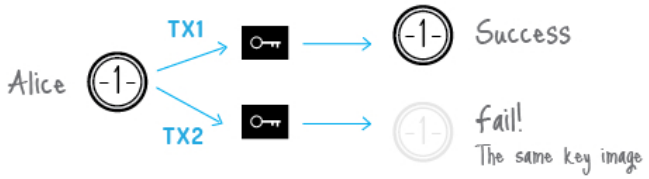

Figure 3.8: Double-spending check. [23]

3.4.5 Blockchain analysis resistance

We reproduce from [23] the reasons, why blockchain analysis in Monero project is not something that can be achieved.

There are many academic papers dedicated to the analysis of the Bitcoin's blockchain. Their authors trace the money flow, identify the owners of coins, determine wallet balances and so on. The ability to make such analysis is due to the fact that all the transfers between addresses are transparent: every input in a transaction refers to a unique output. Moreover, users often re-use their old addresses, receiving and sending coins from them many times, which simplifies the analyst's work. It happens unintentionally: if one has a public address (for example, for donations), one is sure to use this address in many inputs and transactions.

CryptoNote is designed to mitigate the risks associated with key re-use and one-input-to-one-output tracing. Every address for a payment is a unique one-time key, derived from both the sender's and the recipient's data. It can appear twice with a probability of a 256-bit hash collision. As soon as you use a ring signature in your input, it entails the uncertainty: which output has just been spent?

Trying to draw a graph with addresses in the vertices and transactions on the edges, one will get a tree: a graph without any cycles (because no key/address was used twice). Moreover, there are billions of possible graphs, since every ring signature produces ambiguity. Thus, you can't be certain from which possible sender the transaction-edge comes to the address-vertice. Depending on the size of the ring you will guess from "one out of two" to "one out of a thousand". Every next transaction increases the entropy and creates additional obstacles for an analyst.

3.4.6 More about CryptoNote

There are several noteworthy details about CryptoNote and several implementation details of this protocol in the Monero project. However, it would be unproductive and it would harm the readability of this thesis to describe every aspect of this protocol. We believe that the reader has now a good understanding of the backbone of Monero's privacy and anonymity features. Nevertheless, before we start the CryptoNight description (see section 4) we will elaborate on some additional details about CryptoNote. Reproduced from [23]:

Adaptive limits

A decentralized payment system must not depend on a single person's decisions, even if this person is a core developer. Hard constants and magic numbers in the code deter the system's evolution and therefore should be eliminated (or at least be cut down to the minimum).

Every crucial limit (like max block size or min fee amount) should be re-calculated based on the system's previous state. Therefore, it always changes adaptively and independently, allowing the network to develop on it's own. CryptoNote has the following parameters which adjust automatically for each new block:

Difficulty The general idea of our algorithm is to sum all the work that nodes have performed during the last 720 blocks and divide it by the time they have spent to accomplish it. The measure of the work is the corresponding difficulty value for each of the blocks. The time is calculated as follows: sort all the 720 timestamps and cut-off 20% of the outliers. The range of the rest 600 values is the time which was spent for 80% of the corresponding blocks.

Maximum block size Let MN be the median value of the last N blocks sizes. Then the *hard-limit* for the size of accepting blocks is $2 \cdot MN$. It averts blockchain bloating but still allows the limit to slowly grow with the time, if necessary. Transaction size does not need to be limited explicitly. It is bounded by the size of the block.

Smooth emission

In the CryptoNote description [23] one can find the following; the upper bound for the overall amount of all digital coins is also digital:

$$\text{MSupply } = 264{-}1 \text{ atomic units} \tag{3.4}$$

This is a natural restriction based only on implementation limits, not on intuition like "N coins ought to be enough for everybody". To make the emission process smoother, CryptoNote uses the following formula for block rewards:

$$\text{BaseReward } = (\text{MSupply } - A) >> 18 \tag{3.5}$$

where A is the amount of previously generated coins. It gives a predictable growth of the money supply without any breakpoints.

During our research, we found the above description peculiar and confusing. After a while and some forum conversations, we understood that the actual implementation restriction is that a single output cannot have an amount greater than $2^{64}{-}1$ atomic units (which is $1.84 \cdot 10^{31}$ XMR).

Therefore the restriction that the above feature is referring to, is about individual outputs and not the total sum of all outputs that can exist on the blockchain.

This is currently an implementation limitation related to Monero's *bulletproofs* [40] (see section 3.5.2) which proves that amounts are not negative, by proving they are less than 2^{64}. Because Monero units are expressed as positive integers which are subject to modular arithmetic, a very high number can be equivalent to a negative number when added to another Monero amount, which is why this "less than" check proves the number is not effectively negative.

It's easy for this limit to be increased, if necessary, in the future (it's extremely unlikely to be necessary). Note that this observation is specifically related to theoretical limitations. It may be possible that certain Monero wallet implementations also store amounts using a data storage technique that would prevent numbers larger than 2^{64} from being stored.

CryptoNote Transaction

Here we will present a complete CryptoNote transaction from Bob to Carol. Again reproduced from CryptoNote [23], we will subjoin an example and a figure showing the details. The example below is illustrated in figure 3.9, in the next page.

Bob decides to spend an output, which was sent to the one-time public key. He needs Extra **(1)**, TxOutNumber **(2)**, and his Account private key **(3)** to recover his one-time private key **(4)**.

When sending a transaction to Carol, Bob generates its Extra value by random **(5)**. He uses Extra **(6)**, TxOutNumber **(7)** and Carol's Account public key **(8)** to get her Output public key **(9)**.

In the input Bob hides the link to his output among the foreign keys **(10)**. To prevent double-spending he also packs the Key image, derived from his One-time private key **(11)**.

Finally, Bob signs the transaction, using his One-time private key **(12)**, all the public keys **(13)** and Key Image **(14)**. He appends the resulting Ring Signature to the end of the transaction **(15)**.

CryptoNote elliptic curve

The elliptic curve *ed25519* is both a signature scheme and a use case for Edwards form Curve25519 [10]. EdDSA (*Edwards-curve Digital Signature Algorithm*) generalises this signature scheme to any curve in Edwards form.

Curve25519 first arrived in 2006 [9], a few years before the Edwards normal form papers on elliptic curves. Montgomery curves, the form of curve used for Curve25519, was originally used to speed up elliptic curve factorisation [57]. The original proposal for Curve25519 was for use as a Diffie-Hellman (key exchange) protocol [26]. This is still its use and is now often called *X25519*.

Later, Edwards came up with his own form of elliptic curve [30]. Daniel J. Bernstein, Tanja Lange et al. researched these forms and realised they too were fast, especially for signatures and we got Ed25519 [10] using the Edwards form of Curve25519. So far, we have the following nomenclature:

Curve25519, Curve41417, Ed448-Goldilocks generally the name of the curve itself.

X25519, X448, X41417 Diffie-Hellman key exchange schemes using the above curve.

EdDSA, Ed25519, Ed448 The first being the generic Edwards variant of DSA, plus other fixes, the others being specific instances matched to their curve names.

Confusingly, Open Whisper Systems came up with *XEdDSA* [60]. To quote them:

> *XEdDSA enables use of a single key pair format for both elliptic curve Diffie-Hellman and signatures.*

Hence, *X EdDSA* is taken to mean "exchange and EdDSA" of the given curve. In this instance, the key exchange part still happens using the montgomery form of the curve, but the signature part (EdDSA) uses the same curve in Edwards form.

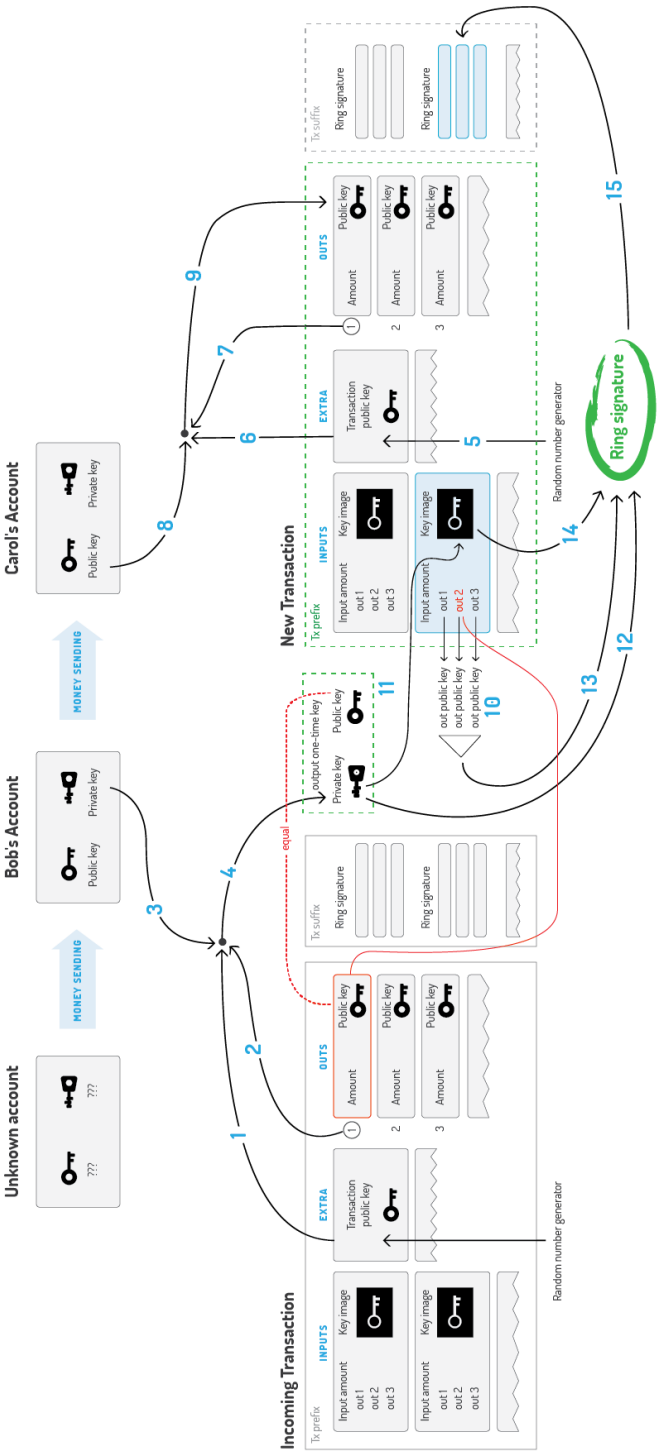

Figure 3.9: A sample transaction from Bob to Carol. [23]

3.5 Monero vs CryptoNote

The reader understands now that the backbone for Monero's anonymity and privacy features is CryptoNote protocol. However, there are some differences. One that was already pointed out is the *stealth address* feature. We have noted in section 3.4.2 that the Monero stealth address implementation is unique. Other CryptoNote projects do not share exactly the same calculations.

Moreover, Monero is actively developed as it is one of the most successful cryptocoin projects and several additions have been made since its first introduction to the world. Its backbone is still the CryptoNote protocol but in this section we will mention additional features that strengthen the anonymity and privacy goals of the project.

3.5.1 RingCT

In the Section 3.4.1 we mentioned *one-time ring signatures* that were presented in CryptoNote white paper.

In 2015, Shen Noether wrote a paper using a technique, introduced by Bitcoin Core developer Gregory Maxwell in [52], of using a commitment scheme to hide the amount of a transaction. The paper introduced *RingCT (Ring Confidential Transactions)* [59]. This signature scheme is called *A Multi-layered Linkable Spontaneous Anonymous Group signature* and that is how transaction amounts are hidden in Monero. Reproducing from [40]:

> *RingCT was implemented in block 1220516 in January 2017. After September 2017, this feature became mandatory for all transactions on the network.*

For further information the reader is refered to Shen Noether's paper [59].

The transaction structure remains similar to the structure in Bitcoin: every user can choose several independent incoming payments (transactions outputs), sign them with the corresponding private keys and send them to different destinations. Contrary to Bitcoin's model, where a user possesses unique private and public keys, in the proposed model a sender generates a one-time public key based on the recipient's address and some random data.

In this sense, an incoming transaction for the same recipient is sent to a one-time public key (not directly to a unique address) and only the recipient can recover the corresponding private part to redeem his funds (using his unique private key). The recipient can spend the funds using a ring signature, keeping his ownership and actual spending anonymous.

3.5.2 Bulletproofs

Starting in 2018, Monero began testing yet another highly sophisticated piece of cryptographic magic: *bulletproofs* [17]. This technology is intended to address one of the main drawbacks of *RingCT* [59]: the size of the *zero-knowledge range proofs* this scheme produces. *Bulletproofs* are a big deal, as they can increase the privacy of digital currency transactions and at the same time dramatically decrease their size. The scalability of confidential transactions have been a significant hurdle for the $1 billion blockchain, with users long suffering high transaction fees as well as an ever-increasing cost of storage for running a full node.

History

After working on the *Confidential Transactions* scheme [59], Greg Maxwell, Andrew Poelstra and Pieter Wuille teamed up with researchers from the Stanford Applied Cryptography Group to make it more efficient. Their research focused on applying a *non-interactive zero knowledge proof (NIZKP)* system [41] to aggregate all the range proofs of a Confidential Transaction and collectively prove their validity.

For context, the basic concept behind a zero-knowledge proof is to cryptographically prove that something exists, without knowing what that something is. This is achieved through a set of challenges that, if completed successfully, can statically prove that a party has a secret, without knowing what that secret is. This is the technology employed by Zcash [79] to entirely shield senders, receivers and the amount of ZEC (Zcash cryptocoin) sent in a transaction.

Zero-Knowledge proofs are an amazing and counter-intuitive cryptographic concept, first proposed by Goldwasser, Micali and Rackoff [41] in a paper that introduced the idea of *interactive proof* systems. The literature is extensive and if the reader wants to learn about the more modern and practical references and implementations, he/she can find the ZKP Science website [80] very useful.

The NIZKP system proposed by the bulletproof white paper [17] has both benefits and drawbacks. On one hand, the use of *NIZKP bulletproofs* does not require a trusted setup for parameter generation, like Zcash's Powers of Tau ceremony [16]. On the other hand, the verification of a bulletproof is more time consuming.

Details

In order to understand bulletproofs, the reader needs to understand what a *range proof* is. According to [40]:

> *A range proof allows anyone to verify that a commitment represents an amount within a specified range, without revealing anything else about its value.*

Monero uses a range proof in RingCT [59] to secure the amount being sent in a transaction. Without range proofs, the amount sent could be hidden, but a sender could cheat by making coins out of thin air. Range proofs prevent this from happening. Bulletproofs achieve this goal more efficiently.

From the whitepaper [17]:

> *A new non-interactive zero-knowledge proof protocol with very short proofs and without a trusted setup; the proof size is only logarithmic in the witness size. Bulletproofs are especially well suited for efficient range proofs on committed values [...]*

Beyond improving the privacy assumptions within confidential transactions [59], bulletproofs have a much lower fingerprint (or size) relative to the proof systems used in blockchain networks today. In fact, much like *SegWit* [66], bulletproofs can be seen as an approach to vertical scalability as they can greatly decrease the size of a cryptographic proof from over 10kB to less than 1kB.

The bulletproof white paper [17] focused on applying NIZKPs to the Bitcoin blockchain and stated that, if implemented, total size of Bitcoin's UTXO[5] set would be only 17 GB (compared to 160 GB) if confidential transactions were to be implemented.

It's worth noting that bulletproofs don't actually contribute to privacy itself. Rather, they simply ensure that the information stored within a confidential transaction doesn't contain any false information. Pseudonymous Monero cryptographer Sarang Noether, who assisted with the bulletproofs' integration, told CoinDesk [19]:

> *They're not about anonymity; they are about assuring that the other stuff we do for anonymity works correctly.*

Under the previous range proof format, the size of XMR transactions scale mostly linearly depending on the number of outputs (1 output = 7kB, 2 outputs = 13kB). Under bulletproofs, transaction sizes scale logarithmically instead (1 output = 2kB, 2 outputs = 2.5kB). The size of a bulletproof increases only logarithmically with both the size of the range and the number of outputs. Reproducing from [40]:

> *This gives us two related types of bulletproofs: single-output and multiple-output. A transaction with multiple outputs can either include several single-output proofs or one multiple-output proof (which is smaller than the separate proofs).*

Therefore, this technology has the potential to greatly contribute to Monero's scalability. However, one problem arose. Again, reproduced from [40]:

> *An attacker could pack a transaction with many outputs; this tiny transaction would require low fees but would be computationally expensive to verify, opening the door to denial-of-service attacks. Because of this, we will need to adjust the fee structure away from transaction size and take into account the verification scaling.*

They explain that this means that the fees will scale properly and in a safe way. It does not mean that fees go up.

[5] Unspent Transaction Output. UTXOs are processed continuously and are responsible for beginning and ending each transaction. Confirmation of transaction results lies in the removal of spent coins from the UTXO database. But a record of the spent coins still exists on the ledger. [43]

The space savings granted by bulletproofs may also enable the implementation of additional obfuscation mechanisms. It is noteworthy that increasing the mandatory number of outputs in a transaction can make it significantly harder to trace balances by analyzing the blockchain. Decoys are used in ring signature inputs, but not in a transaction's outputs. Implementing a system of decoy outputs will certainly increase the size of a transaction, but this increase may be trivial post bulletproof activation.

The impact

Transaction fees on Monero, the 10th largest cryptocurrency network, have fallen sharply. Bulletproofs' technology made the Monero network's privacy features more scalable by restructuring how its confidential transactions are verified.

According to data published by *BitInfoCharts* [54], average Monero fees fell from about $0.54 cents to roughly $0.021 cents in two days, a 96% drop. Monero's average transaction size is now 3kb versus a pre-fork average of 18.5kb. In figure 3.10 we can see the transaction size decrease since the implementation of bulletproofs.

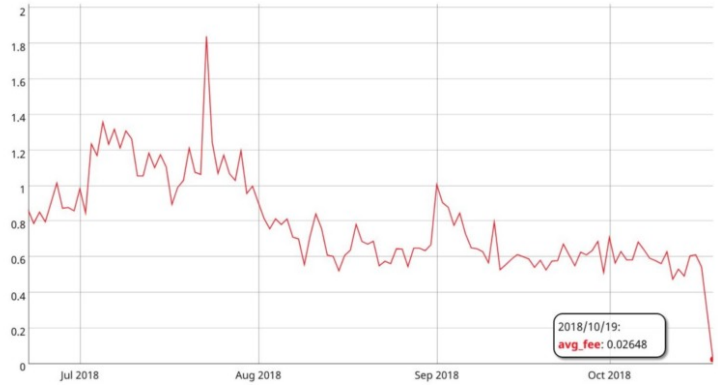

Figure 3.10: July 2018 - October 2018. [54]

If we see this change in a broader time interval, the result is even more impressive:

Figure 3.11: July 2018 - October 2018. [54]

There were predictions that the drop of fees might open the door to additional uses for XMR, the cryptocurrency that powers the Monero blockchain. Core developer `hyc` said that the upgrade,

definitely [makes] the notion of micropayments more palatable again

3.5.3 Kovri I2P Network

Up to now, we have covered how Monero obfuscates information stored on the blockchain. Ring signatures obscure the sender. Stealth addresses prevent outside observers from knowing the receiving address. Finally, confidential transactions hide the amount of Monero transmitted. However, some personally identifiable information may be leaked at the network level when making a transaction. This privacy leak is addressed with Kovri [49].

Kovri is a free, decentralized, anonymity technology based on I2P's open specifications [44]. Kovri uses both encryption and sophisticated routing techniques to create a private overlay-network across the Internet. This protected overlay allows users to hide their geographical location and IP address.

Examples

In the presentation of the project in the Gitlab page [48] some examples that Kovri's use protects a user's privacy are described. These examples will help the reader to understand the significance of Kovri's contribution to the anonymity level that Monero project aims to offer. Let's reproduce here these examples.

Suppose Alice wants to send Monero to Bob. Alice's wallet creates a transaction and then broadcasts it to the Monero network. The Monero network is made up of nodes that communicate with each other by directing messages using IP addresses. This means that it might be possible to geographically trace data as it travels over the open Internet, from start to finish and everywhere in between. Even though the sender's and recipient's wallet addresses - as well as the amount of Monero sent - remain private, Alice is taking a risk in broadcasting her transaction as some nodes may be logging IP addresses. An adversary with enough resources could attempt to associate transactions with IP addresses to determine from where transactions originate. This could potentially lead to an adversary not relaying her transactions to the rest of the network; or arriving at her front door!

Now let's imagine a different scenario. Suppose Charlie wants to support the Monero network by running a full node at his home. After a few weeks, he receives a cease and desist letter from his Internet Service Provider claiming that running a node is a violation of the terms of service.

Or consider this, suppose Dave is an operator of a mining pool that donates a portion of block rewards to a political party or controversial cause. Other nodes could purposefully reject his solved blocks to express their disagreement with his political or social views.

Alice, Bob, Charlie, and Dave all have at least one thing in common: their IP addresses were exposed. Users could try to hide IP addresses with the Tor [74] or a VPN; however both of these strategies have serious weaknesses. The Tor network has ”semi-trusted” Directory Authorities which give a handful of Tor node operators overreaching influence into network consensus. Network consensus ultimately determines who is allowed to relay traffic on the Tor network based on the views of the Directory Authorities. Furthermore, correlation attacks are easily possible with trusted VPNs, making it easy for large attackers to de-anonymize a user's traffic.

If Alice, Bob, Charlie, and Dave exclusively use Kovri to connect to the Monero network, no one will know their IP address, making passive surveillance impractical, while substantially improving Monero's censorship resistance.

Technical Attributes

Kovri tunnels traffic through the I2P network utilizing *garlic encryption* and *garlic routing*. The reader can find more about this technology in the I2P Network description [44]. Information travels within a private overlay-network by way of messages, which are encrypted in layers each time the message is passed along to peers in the network, similar to a Matryoshka doll.

For each inner doll there is a lock and public key to the next doll. Peers in the network are not able to read the contents of the message being relayed, so information sent from the sender to its destination (and vice-versa) are secured. The only information visible to peers is the instruction for sending messages to the next peer. To achieve greater privacy at a slight cost to performance, users are able to connect to several peers.

Essentially, Kovri covers an application's Internet traffic to make it anonymous within the network. Given this characteristic, Kovri is a great solution for anonymously communicating over IRC, email, or accessing hidden services. As Kovri is an open source project, the reader can find its full details in the project's Gitlab page [48].

Part II

Back to decentralization

CHAPTER 4

OUR MODEL

> Democracy must be something more than two wolves and a sheep voting on what to have for dinner.
>
> *James Bovard*

4.1 CryptoNight Description

In this section we will describe in detail the proposed implementation of the CryptoNight hash function. This function is used in the Monero project in order to achieve *egalitarian mining*. It is easy to understand why we characterize this implementation as *proposed*, since each miner is free to use any implementation he/she can think of, as long as it produces the right result.

The first step we took, to understand how CryptoNight actually works, was reading Monero project's reviews. However, as Monero core developer `smooth_xmr` has posted on reddit [62] when asked specifically about Cryptonight function reviews:

> *CryptoNight was extensively reviewed, though not as part of a "formal" review process, by Professor David Andersen who also wrote the current implementation of the hashing code.*
> *[...] stating that it would likely achieve its goals of resisting extreme optimizations and narrowing the performance gap between CPUs, GPUs, and ASICs.*

We tried to find and read this review, as it would be a great start for our work. Unfortunately, as we discovered, the review was really informal and the best we could find was a post in professor David Andersen's personal blog [4]. There, one can find the *first*, to our knowledge, graphical representation of the second stage of the CryptoNight function (see section 4.2.2).

Again to the best of our knowledge, this thesis is the close second.

In the proposed implementation, a scratchpad[1] is used (2MB) to ensure that the memory needed fits the size of L3 cache (per core) in modern processors. In practice, the miner should measure mining power and calculate efficiency.

In this chapter we will just show the proposed implementation of CryptoNight with a minor analysis in the last section. We will demonstrate the three stages of the computation and the role for each element. A really quick overview of these stages would be something like this:

1. Initialize the scratchpad in a pseudo-random manner.
2. Read/write operations at pseudo-random addresses. (*memory-hard* part)
3. Use all the computations' results to produce the output.

4.2 The three stages

Enough with the overview of the function and its history! Let's dive into it and see in detail its components as it is described in [67].

The input of this algorithm is a block and if the value of the Cryptonight function satisfies the target (see equation 4.1), it is possible that this block is the next block in the blockchain.

$$\text{Cryptonight}(\text{block}) \leq \text{Target} \tag{4.1}$$

So, the input of the function is a block of transactions along with the necessary fields, which are specified by the Monero protocol. For our purposes, it is enough for the reader to think of the procedure as simple as it gets. We accept that the only way to meet the target is by bruteforcing. So, the miner tries many blocks as "candidates" and hopes for the best. Every time he/she "tries", he/she actually computes the Cryptonight digest for some random block and checks whether the equation 4.1 is satisfied.

In this section, we will present CryptoNight's inner computations graphically. The illustration of this section is the work of the UI and graphic designer, Vasilis Agiotis [76]. His help is valuable, as the visual representation is needed. CryptoNight has a relatively complex operation sequence and our analysis requires focus on details.

4.2.1 The first stage

The first stage of the algorithm sets the initial value of the scratchpad. In order to prevent several attack schemes, the scratchpad must be initialized with data chosen in a way, which is indistiguishable from the uniform distribution. This is the goal.

We will describe the first stage in several parts and discuss the role of each part and its contribution regarding the properties of function's output. The first stage is presented graphically in figure 4.1. We recommend the reader to refer to the graphical representation for clarity.

[1] A large area of memory used to store intermediate values during the evaluation of a memory-hard function.

Description of the first stage

To begin, let's prepare the tools:

1. Hash the input using Keccak [72] ($b = 1600$, $c = 512$).
2. Choose the first 32 bytes of the final state.
3. Interpret them as an AES-256 key.
4. Expand them to 10 round keys.

Keccak is the versatile cryptographic function that is most known as SHA-3. The parameter analysis and the description of their part is beyond the scope of this thesis. The reader is referred to their work.

We will consider Keccak a collision-free hash function. The next three steps produce random keys for encryption. We consider these keys random enough for the purpose of their use. They are interpreted as keys and expanded according to [69]. Create the scratchpad:

5. Allocate 2097152 bytes (2MiB).

The encryption part:

6. Split the bytes 64 to 191 into 8 blocks of 16 bytes each.
7. Encrypt the blocks as follows:

```
for i = 0..9 do:
    block = aes_round(block, round_keys[i])
```

8. Fill 128 bytes of the scratchpad with the resulting blocks.

Repeat:

9. With the resulting blocks run step 7 again.

Each time 128 bytes are written, they represent the result of the encryption of the previously written 128 bytes. The process is repeated until the scratchpad is fully initialized.

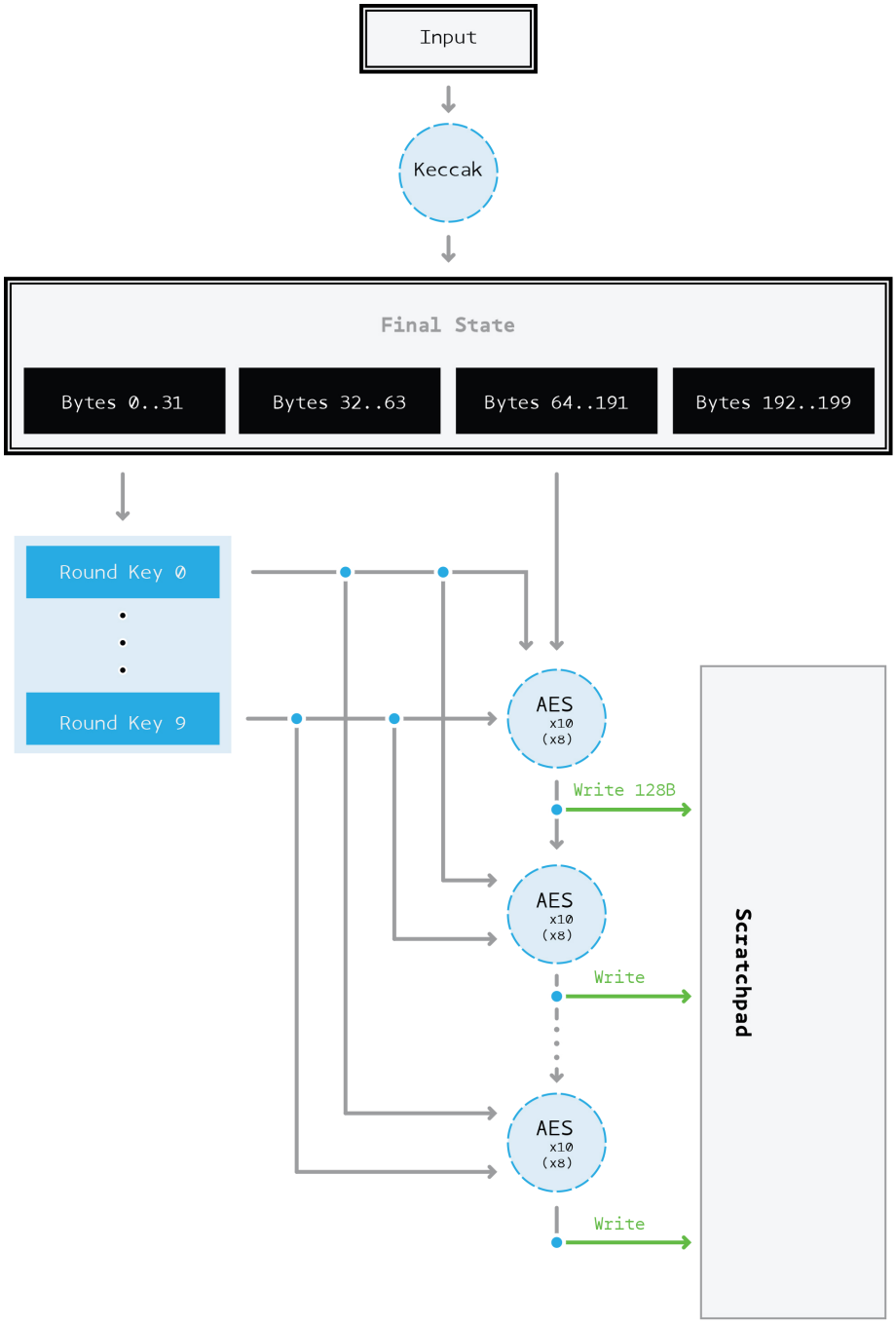

Figure 4.1: Scratchpad initialization. [76]

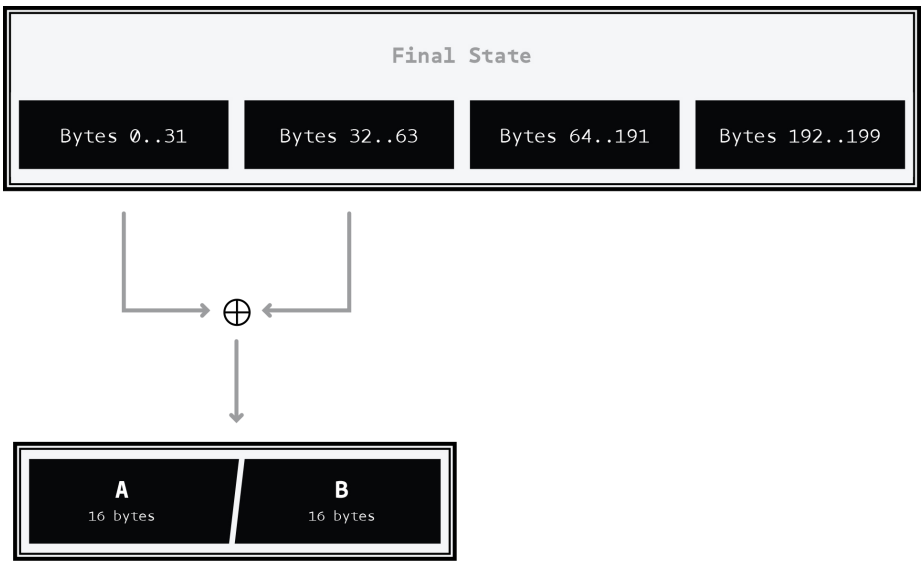

Figure 4.2: Extracting a and b values. [76]

4.2.2 The second stage (memory-hardness)

The second stage of the algorithm uses the initialized scratchpad and two values that are computed from the hashed input of the function. Its goal is to perform computations on the scratchpad values (on all of them with high probability) and produce a final scratchpad structure that can't be computed otherwise or in stages (without huge time complexity). The memory-hardness property is satisfied if and only if there is no other way to compute the final state of the scratchpad using less memory than the size of the scratchpad. That is the overview. Let's see the details.

(The preparation part) The core structure of this stage is a loop. However, before illustrating the computations that take place inside the loop, there are some computations needed for preparation and two technical clarifications.

1. Compute the values of a and b.

Elements a and b are the two values which, along with the scratchpad, are given as input to the loop. More specifically, the first 64 bytes of the hashed input (the Keccak state) are split in two parts (32 bytes each part) and XOR-ed ($\oplus$), and the resulting 32 bytes are used to initialize variables a and b, 16 bytes each.

In figure 4.2 the reader can see the visualization of this preparation part.

Clarification 1:

The reader may notice in figure 4.3 that the function uses a 16-byte value as an address in the scratchpad. Actually, the value is interpreted as a little-endian integer. The 21 low-order bits are used as a byte index. To ensure the 16-byte alignment, the four low-order bits of the index are cleared. This alignment is essential, as the data is read from and written to the scratchpad in 16-byte blocks.

Clarification 2:

The main loop is iterated $524,288 = 2^{19}$ times. Every time, two blocks of the scratchpad are written, so with high probability, the whole scratchpad will be overwritten. In every iteration, along with the two blocks of the scratchpad, values a' and b' are computed, which are used as input to the next iteration.

Now we are ready to describe the inner computations of the loop. We will divide this stage into parts, as it will help us later in the analysis. The number of parts are determined based on some intermediate values. We refer the reader to the graphical representation of this stage, presented in figure 4.3.

Here, we note that the intermediate values are not memory requirements, as we can implement the computations with only 32 bytes of memory (for a and b) plus the memory needed for the scratchpad. But during theoretical analysis and understanding of the function's computations, these intermediate values seem natural stops of the train of thought. So, after the step 1:

2. Interpret the value of a as a scratchpad address.
3. Read from this address.
4. Evaluate the AES function with data from step 3 and key the value of a.

Let's call this intermediate value c. And let's add a final step to this part:

5. Calculate $c \oplus b$ and write the result to the address of step 2.

The value c is passed as b', part of the input of the next iteration. The second part involves another read from the scratchpad:

6. Interpret the value of c as a scratchpad address.
7. Read from this address. (We will refer to this intermediate value as d.)
8. Multiply[2] c, d and add the value of a to the result.
9. Write the result of step 8 to the address of step 6.

This concludes the second part. The scratchpad is written twice per iteration. The only thing that is left to conclude the description of the second stage, is the computation of a', part of the input of the next iteration. This is computed as follows:

10. Compute $d \oplus$ (<the result of step 8>) to compute a'.

[2]The multiplication uses only the first 8 bytes of each argument, which are interpreted as unsigned 64-bit little-endian integers and multiplied together. The result is converted into 16 bytes, and finally the two 8-byte halves of the result are swapped [67].

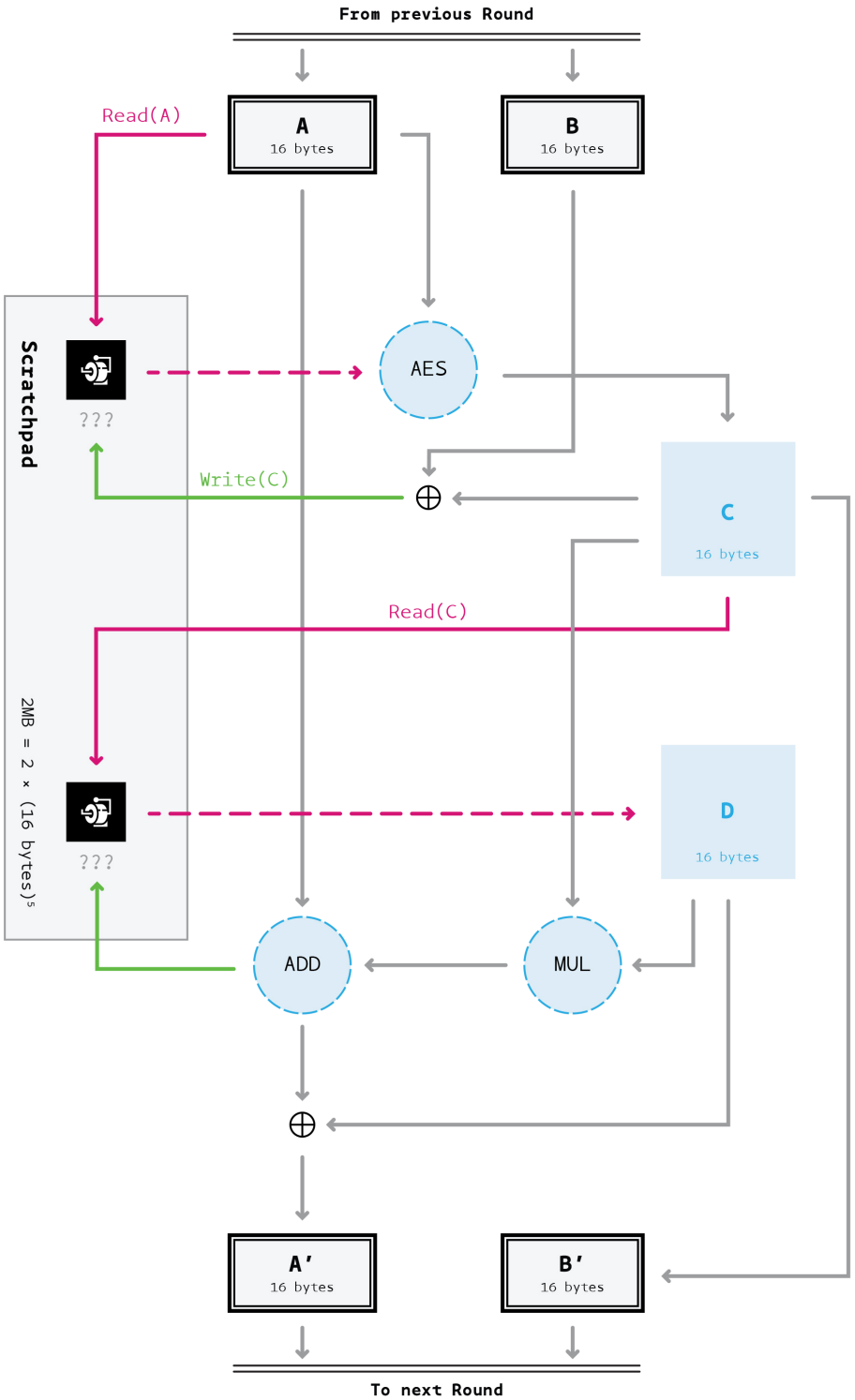

Figure 4.3: The memory-hard part. [76]

4.2.3 The third stage

The third stage of the algorithm uses the final state of the scratchpad to produce the output. During this stage AES operation is used. At first, the function extracts 10 key values from 32 bytes of the hashed input of the function, similar to the step 1 of the first stage. Extracting keys:

1. Choose the bytes $[32...63]$ of the final Keccak state.
2. Interpret them as an AES-256 key.
3. Expand them to 10 round keys.

After this, the function needs a starting value. It applies XOR-operation($\oplus$) on bytes $[64...191]$ of the hashed input and the first 128 bytes of the scratchpad. Let's call these values *input* and *scratchpad[0]*.

4. *input* $\oplus$ *scratchpad[0]*.

Now, using the first key of step 3 as key to the AES operation:

5. Encrypt the result of step 4.

Repeat the last two steps as follows:

- Take the encrypted result of the last step as input.
- Take the next 128 bytes of the scratchpad (*scratchpad[1]*).
- Use, as AES key, the next extracted key.
- Execute step 4 and step 5.

until the last bytes of the scratchpad are used to the aforementioned operations. After the last bytes of the scratchpad are XOR-ed and encrypted,

6. Use the result to replace the bytes $[64...191]$ of the hashed input.

We call the state of the hashed input after the above step as the *modified Keccak state*. To produce the final result the function performs the next steps (see figure 4.4b).

6. Pass the *modified Keccak state* through Keccak-f (the Keccak permutation [72]).
7. Choose the 2 low-order bits of the first byte of the *modified Keccak state*.
8. Based on these bits choose a hash function:

case 00: BLAKE-256	case 01: Groestl-256
case 10: JH-256	case 11: Skein-256

9. Apply the chosen function to the *modified Keccak state*.

The result of step 9 is the output of the CryptoNight function. For more information about these functions, the reader is refered to the respective articles [7, 39, 77, 33].

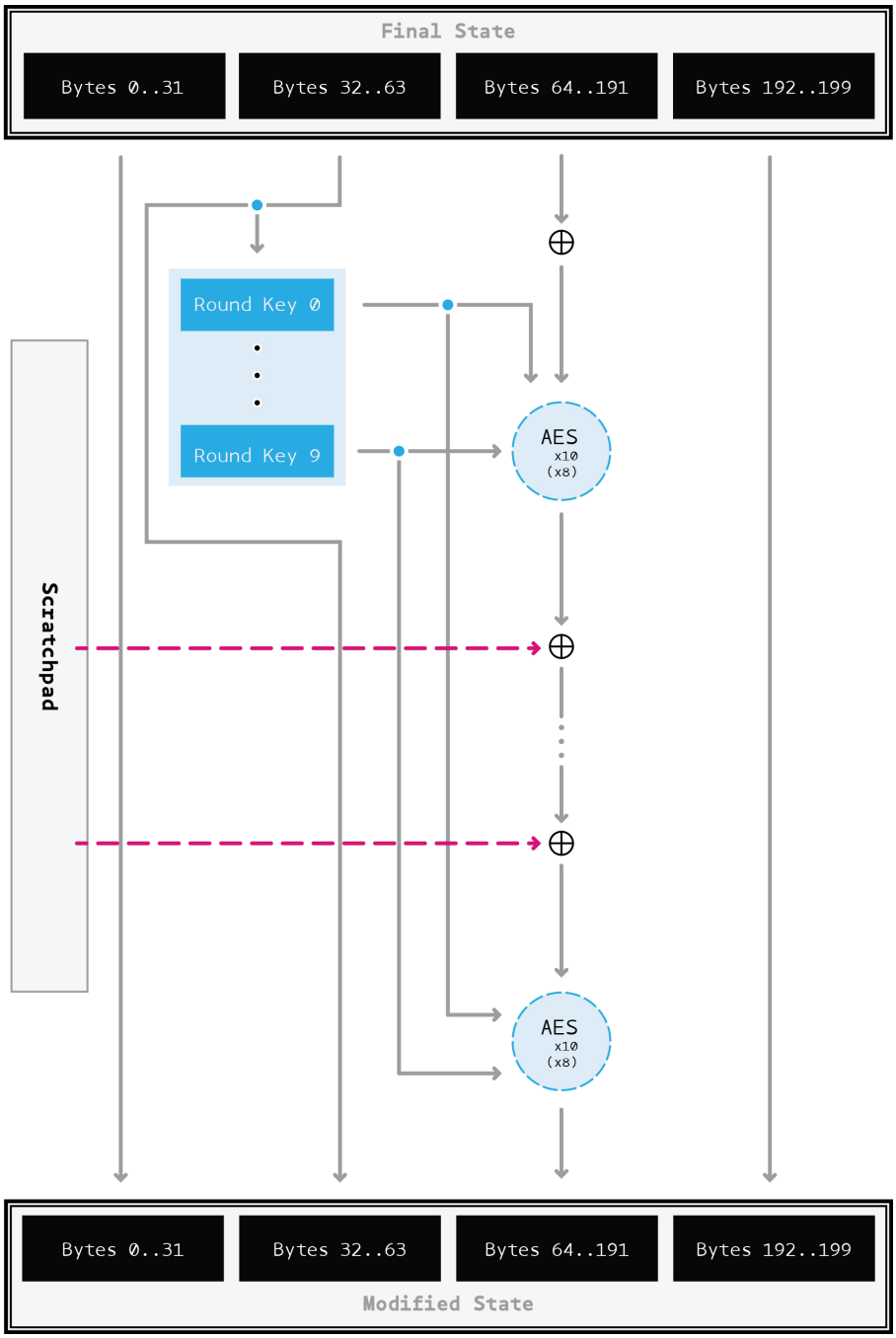

Figure 4.4a: The third stage. [76]

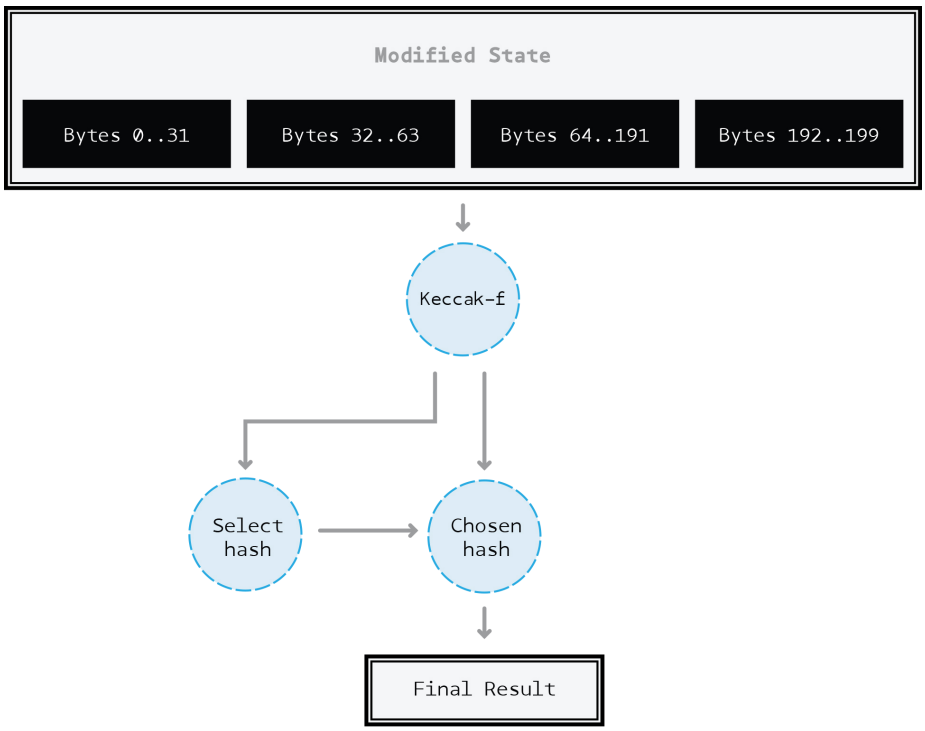

Figure 4.4b: The third stage. [76]

4.3 Analysis

Now that we have described the way this function computes its output, we will try to build a model for our theoretical analysis. There are several assumptions that we have to make to abstract the function's operation. For this purpose, we build our model and define its security. At first, we have to focus on the part that is linked to the *memory-hardness* property (see section 2.1.3). The reader now should have an understanding about the general purpose of each stage.

The first stage sets the scene for the memory-hardness part. We will assume that it initializes the scratchpad in a way, which is indistiguishable from the uniform distribution. It is safe to assume that, for an honest miner, the input is chosen uniformly at random as well. Our goal is to focus on the second stage, analyze it and then try to imagine, how an adversary miner can attack the memory-hardness property of this function. Due to the aforementioned assumptions we can conclude that the input of the second stage is chosen uniformly at random from its domain. Just to remember, the second stage's input is:

- a
- b
- Scratchpad, from now on denoted as SP

4.3.1 Parameters

One of the first things that we need to do is to parametrize the input. We can't talk about complexity or security without the relative size between our objects or calculations. Moreover, this kind of analysis can help to generalize results and conclusions.

We will arbitrarily choose n as symbol for the size of a and fix everything else respectively. With that analysis in mind, we are fixing our language. Symbols:

Size of a:	n
Size of b:	n
Size of SP:	$\beta_n = 2 \cdot 10^6$ bytes $\approx 2 \cdot n^5$ (polynomial)
Value of SP in address x:	SP_x (of size n)

4.3.2 AES as PRF

The Advanced Encryption Standard (AES), also known by its original name Rijndael [25], is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [69].

AES is a subset of the Rijndael block cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, who submitted a proposal to NIST during the AES selection process. Rijndael is a family of ciphers with different key and block sizes.

For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.

AES has been adopted by the U.S. government and is now used worldwide. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

In CryptoNight function AES is used for its properties as a cryptographic function. If a different key and a different input is chosen every time, then it is safe to assume that no prediction about the output of AES on this input can exist. To make this assumption more formal, we assume that AES is a pseudorandom function (PRF) and the seed is the key of AES.

In section 2.1.4 the reader can find the mathematical definition of PRFs. We define the notion and then we define a game. Based on this game we describe a security model to base the above assumption.

4.3.3 Operations

Apart from the AES use, the memory-hard stage of CryptoNight function performs one addition, two XOR-operations and one multiplication. In order to be able to produce conclusions, we try to analyze what the side effects of these operations are. All of these operations get two inputs of 16 bytes size and produce a 16 byte result.

Let's examine them one by one. In the case of the XOR operation, it is easy to see that if the two inputs are chosen uniformly at random, then the result is also uniformly chosen at random. In the case of addition, reproduced from CryptoNote [67]:

> *The* `8byte_add` *function represents each of the arguments as a pair of 64-bit little-endian values and adds them together, component-wise, modulo* 2^{64}*. The result is converted back into 16 bytes.*

It is trickier to see the same here, but with a little effort one can see that if the input is chosen uniformly at random, then the result is uniformly chosen at random too. In the case of multiplication, reproduced from CryptoNote [67] again:

> *The* `8byte_mul` *function, however, uses only the first 8 bytes of each argument, which are interpreted as unsigned 64-bit little-endian integers and multiplied together. The result is converted into 16 bytes, and finally the two 8-byte halves of the result are swapped.*

The last case, is more complex. At first we notice the following: if one of the inputs is null, then the result is also null. That could be a problem. We try to calculate the probability that the result is null, due to the null value of one or both inputs. We don't care about the null value of the result due to modulo operation. That probability is obviously equal to the probability that the result is equal with some other value. We would like the "extra" probability, that the value of the result is null due to input's null value, to be negligible.

Because of the special way CryptoNight function performs the multiplication, the inputs' sizes are 64 bits = 8 bytes = $\frac{n}{2}$. In addition, considering that AES is a PRF (assumption) and SP is unifomly random (at least at the first round by assumption), then the two inputs are independent. The probability of one or both inputs to be null is:

$$\frac{1}{2^{n/2}} + \frac{1}{2^{n/2}} - \frac{1}{2^{n/2} \cdot 2^{n/2}} = \frac{2 \cdot 2^{n/2} - 1}{2^n} < \frac{1}{2^{(n/2)-1}} = \textbf{negl}(\text{n}) \tag{4.2}$$

That seems to be fine for our analysis. But, the next problem is this: Multiplication of two 8 byte numbers produces a result modulo 16 bytes (mod 2^{128}). The reader can see that the probability of a value to be an output of the multiplication declines, as the values grow. Even after the swap that is performed at the end of the multiplication, the problem persists. We have not a uniformly at random distributed result, even if the inputs are chosen uniformly at random.

The above multiplication is perfomed this way because of the modern CPU registers' size. The 8 bytes multiplication is optimized. ASICs couldn't do this as fast as a modern CPU could, but technology advances and now there are chips that do the same computation roughly with the same time cost.

However, after our theoretical analysis we think that we should propose something better. The time cost of a 16 byte multiplication or maybe an 8 byte multiplication implementation that maps inputs' uniform distribution can make a system less efficient against ASICs, compared to the official implementation proposal, but not completely inefficient. Nevertheless, this is a detail and does not make great difference in our analysis. From now on, we can assume that the above problem is solved and we have a multiplication implementation that produces a uniformly at random distributed result.

CHAPTER 5

CRYPTONIGHT ANALYSIS

> Nobody can give you freedom. Nobody can give you equality or justice or anything. If you're a man, you take it.
>
> *Malcolm X*

5.1 Introduction

In order to prove that a function is *memory-hard* (see section 2.1.3) we need to show that no implementation exists, that can produce the same result using less memory without significant time cost. In other words, using an implementation which needs less memory is not something that can give advantage to some miner because the time factor will make the procedure equally or more *expensive*, even if the miner uses parallel computation techniques. Reproduced from CryptoNote [67]:

> *CryptoNight is a memory-hard hash function. It is designed to be inefficiently computable on GPU, FPGA and ASIC architectures.*

In Monero mining, CPU's cores are only efficient if they can use the super fast 2MB cache over and over. Each core needs about 2MB for CryptoNight to stay cached. So a miner should check how much L2 cache or - in rare cases - also L3 cache the CPU has. Then divide by 2MB and this will be how many cores he/she can run at the same time.

There are several reasons to suspect that CryptoNight could be a memory-hard function. One of the most popular arguments was that a megabyte of internal memory is an almost unacceptable size for a modern ASIC pipeline. But hardware is constantly evolving and eventually there was recently an effort for Monero ASIC production. The first documented effort was the ASIC called *Antminer X3* by Bitmain [14]. The announcement was made in March 2018. Observing the raise of hashrate in the network, it was obvious that there were ASICs used for mining.

Founded in 2013, Bitmain, is a firm that produces ASIC chips and mines Bitcoin. The firm also operates *Antpool*, which according to observers is the largest Bitcoin mining pool. An ASIC device by Bitmain has been mining Bitcoins for many years.

The reason Monero is planning to make Bitman's Antminer X3 ineffective, is that it could enable some forms of attacks. These attacks could result in the mining pool taking over principal cryptocurrency's hashrate. The act may enable double spending of coins, false transaction histories and censoring payments.

Riccardo Spagni, in a response to a Twitter comment which sees ASICs as a good thing [68], said:

> *Removing all of the hashrate distributed among tens of thousands of miners, in favour of a handful of miners that can afford an overpriced machine from a single manufacturer is GOOD for security? I doubt even you believe that.*

There were some thoughts like "*How did they do this? Isn't CryptoNight memory-bound?*". Well, one thing is that CryptoNight is *ASIC-resistant*, not *ASIC-proof*[1]. But, that was not the problem. Another thought is that L3 cache supports a lot of extra functionality like being shared across cores, writing back to RAM, being behind two other levels of cache, etc. which all makes it a lot less efficient (among other issues with the approach). But, again, that was not the case in that particular effort.

L3 latency wasn't the issue. ASICs just traded latency for bandwidth the same way GPUs do. They were built on stacks of DRAM, not lightning fast caches. The costs of cache complexity aren't only latency but also power usage and die space. Raw speed isn't even necessarily the goal for either CPUs or GPUs or ASICs here, it is efficiency.

But Monero project reacted and announced upgrades bi-annually in order to keep ASICs at bay. Upgrades are a problem, because upgrades produce bugs and vulnerabilities. Especially when they are that frequent. On the other hand, upgrades in Monero are minor, with no changes in the memory-hard part. From this experience, we understand that a formal proof, or even a better understanding of the memory-hardness property in practice, is vital for its mining function in order to protect a cryptocurrency from centralization.

5.2 Proof approach

Our starting plan was quite simple. The moment we understood the operations that took place in the computation, we had a specific strategy in mind. We would prove:

First case (*honest* miner):

- If the input is uniformly at random chosen, then the output is of the same nature *and* independent from the input. (one round)
- The above expands to the whole function, not just one round.

What we wanted to show, with the two steps above, is the following: No shortcuts exist for the calculations involved. If the miner is honest, then the hash of the block will be uniformly at random chosen and so will be the elements of the input (a, b and SP) of the second stage. If that implies that the output of this process is also chosen uniformly at random, then every operation on the input cannot be guessed except with negligible probability, ergo no shortcut of this process exists.

Of course, another requirement is the input (a, b, SP) to be independent from the output (a', b', SP'), where SP' is the modified SP after the round. If this is not the

[1] ASIC manufacturers are discouraged from building an ASIC for Monero mining, but there is no formal mathematical proof stating that an ASIC cannot be built.

case, then there is a relation between two or more elements and in the nature of that relation a chance for an attack may hide.

It is easy to expand the above hypothetical result to the function as a whole. Every round produces a uniformly at random chosen result. Thus, the last round will produce a uniformly at random distributed result too.

Second case (*malicious* miner):

- If the input is *not* chosen at random, then the output is *still* uniformly at random chosen *and* independent from the input. (one round)
- The above expands to the whole function, not just one round.

If we had the results for the honest case, then the next step would be to show that even if the miner is malicious he cannot do any better for himself. Even if the input is not chosen uniformly at random, the result will be distributed uniformly. That means that even if we ”fixed” the input to our taste, we could not guess the result except with negligible probability. The second step follows a similar train of thought as in the case of the honest miner.

If we managed to prove the above, that would be a proof of the CryptoNight's memory-hardness property. If we cannot find a shortcut for the process, then we cannot find a way to calculate the result with less memory or less time. Let us elaborate on the details of our research.

5.2.1 The model

Based on the aforementioned plan and with help from the observations and assumptions we made in section 4.3, we now use the model of computation we assume for CryptoNight's second stage (see figure 5.1) to begin our analysis.

The reader can see that there is a natural division of the process in three parts. The first part is the set of all operations performed on the first address' value. Explicitly, that involves:

Read Input to an AES operation. Under our assumption, AES is a pseudorandom function (PRF).

Write The result of a XOR ($\oplus$) operation.

The second part involves any operation performed on the second address' value. More precisely:

Read Store the value of the address.

Write The result of a multiplication and an addition.

The third part is not of much interest for our purposes. It extracts the two outputs needed as input for next round of the second stage:

- A XOR ($\oplus$) operation on the result of the addition that produces the first input of the next round (a').
- The result of the PRF operation (AES) that is passed as the second input of the next round (b').

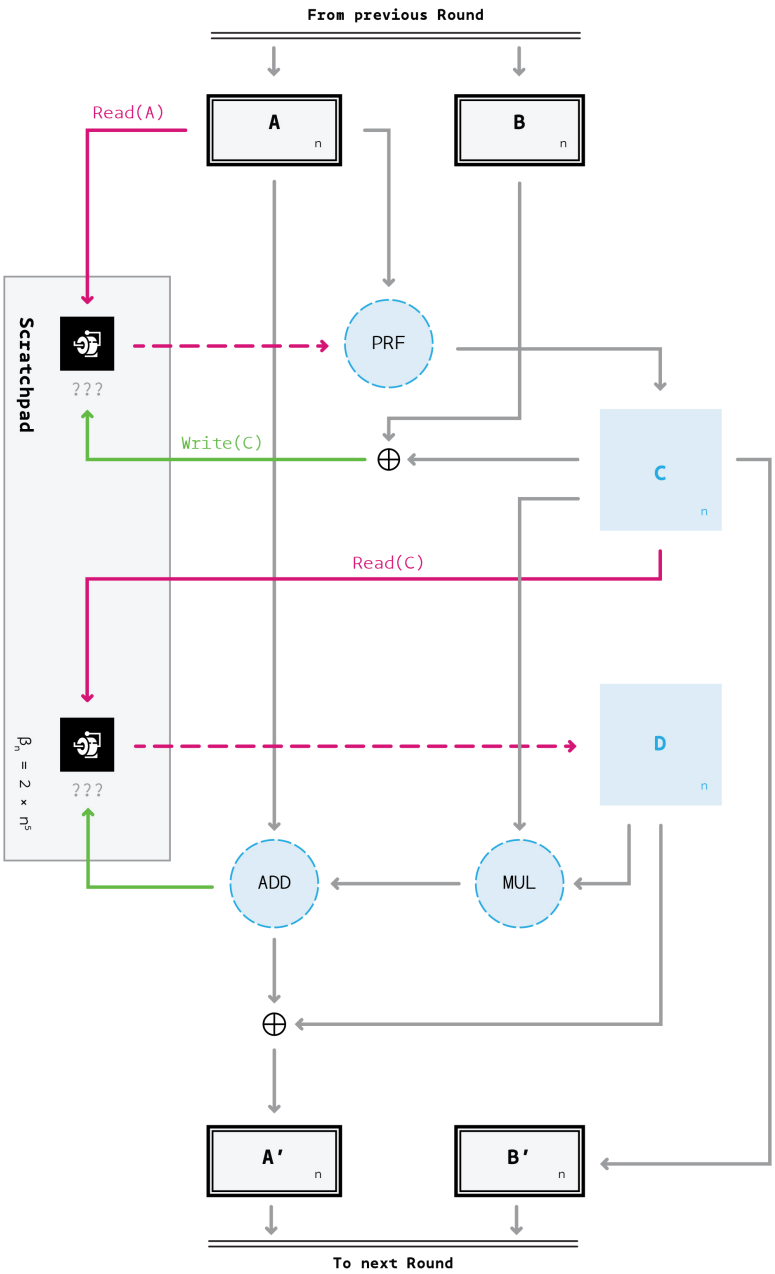

Figure 5.1: The model. [76]

Here we note again (see section 4.3.3) the special nature of the multiplication. Just for our purposes, we assume that it sustains the properties of its input. If the input (the two integers, each of size n) is uniformly at random chosen then the output of the multiplication is of the same nature. We strongly declare that this is not the case for the specific implementation of this operation. However, we will see that even with this assumption, we will not achieve our goal.

5.2.2 The road to proof construction

Based on the above we begin our analysis. Under the following assumptions,

- a, b are uniformly at random chosen
- AES is a PRF

we get that the first address of the round is changed in a way that does not change the distribution of the scratchpad. Everything seems fine.

For the second address we get the same result. Our assumptions are:

- AES is a PRF
- $\forall c : SP_c$ is uniformly at random chosen
- Multiplication and addition sustain the distribution of their inputs

Let's see the values of a' and b':

$$a' = \Big(a + \big(PRF(a, SP_a) \cdot SP_{PRF(a,SP_a)}\big)\Big) \oplus SP_{PRF(a,SP_a)} \tag{5.1}$$

$$b' = PRF(a, SP_a) \tag{5.2}$$

Here we have a major problem. The problem is not the distribution of the output. Under our assumptions that is appropriate. But,

- a' is *not* independent from a
- b' is *not* independent from b
- SP' (scratchpad after the round) is not uniformly at random chosen

The third statement comes out of the observation that the value of SP_a, after the write operation, is dependent on the value of b. These are observations that spoil our plan.

However, this is a hint to the possibility of the existence of a successful attack on the memory-hardness property of CryptoNight function. Part of the input of this stage (values of a, b) is produced right from the hashed input of CryptoNight (see sections 4.2.1, 4.2.2). If someone can control the value of a, or maybe some bits of a, then he has a partial control of the value of a' or at least some control over the range of a'. The same apply to the values of b and b'.

This is, of course, nothing more than an intuition, so let's see if we can achieve the planning of said attack.

5.3 Attack approach

The above intuition, apart from the assumptions involved, implies a level of control over the value of a or b or both. This can be done with grinding techniques. In a very high level of abstraction, the miner can hash the input multiple times with Keccak, changing every time something in the block (nonce, sequence of transactions, etc.). That will give him a different digest every time, ergo a different value for a and b. This is quite efficient, especially if an ASIC is involved.

But we will make a stronger assumption. We will assume that some adversarial miner has a total control over the value of a and b. We will show that even in that case, an attack seems impossible. Due to the nature of our results, it doesn't make sense to analyze the way an attacker can gain control over the value of a or b.

5.3.1 Details

What is the level of impact that we can cause, if we could control the values of a and b? What is the best we can do in order to achieve our goal?

If someone takes his/her time observing the process, he/she can see the only thing that we can hope for. That is to find an address a, with a content SP_a such that:

$$\text{AES}(a, SP_a) = a$$

The reader can see the visualization of this scenario in figure 5.2. Now it is time to measure the probability of this event. If this probability is negligible, we cannot find anything more that we can do. We have β_n addresses on the scratchpad and in the CryptoNight description [67] we see that:

> *When a 16-byte value needs to be converted into an address in the scratchpad, it is interpreted as a little-endian integer, and the 21 low-order bits are used as a byte index. However, the 4 low-order bits of the index are cleared to ensure the 16-byte alignment.*

This means that, for a uniformly at random chosen 16-byte number, the probability for the value to be converted into a specific address in the scratchpad is:

$$\frac{1 \cdot \textbf{sizeof}(\text{address})}{\textbf{sizeof}(SP)} = \frac{1 \cdot n}{\beta_n} = \frac{n}{2 \cdot n^5} = \frac{1}{2 \cdot n^4} = \frac{1}{\textbf{poly}(n)}$$

That is, for all intents and purposes, *not* a negligible probability with respect to the size of n. Hence, that is something that we can do efficiently enough. Let's see what this scenario gives us, as a round outcome. If the reader cares to do some math, he/she will see that equations 5.1, 5.2 become:

$$a' = \Big(a + \big(a \cdot (a \oplus b)\big)\Big) \oplus (a \oplus b) \qquad \text{and} \qquad b' = a$$

This looks really promising. Until we see what happens to the next round. The one thing that the reader is suggested to observe is that, although a' is a function of a and b, it is certainly not equal to either of them.

In the next round, the value a' will point to a different location on the scratchpad ($SP_{a'}$) and the value $\text{PRF}(a', SP_{a'})$ will point to a random address in the scratchpad. And this is exactly the moment we lose the control we had over the process.

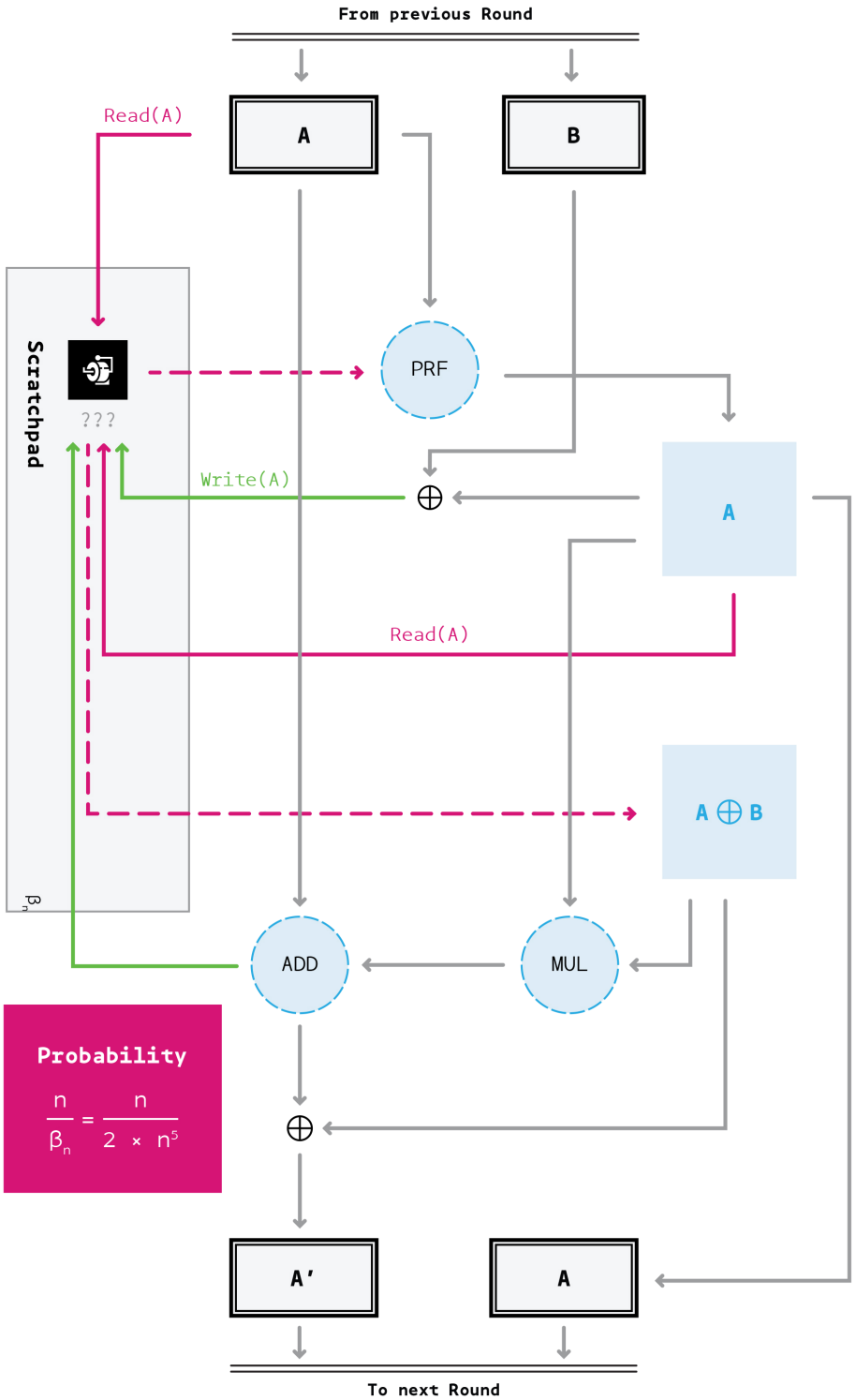

Figure 5.2: The attack scenario. [76]

CHAPTER 6

CONCLUSION

> Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say.
>
> *Edward Snowden*

6.1 Summary

The intuition behind the nature of the problem lies in the relation between the first and the second address written in each loop, in the second stage of the function. As we saw, the first address is a random pointer produced from the input. This is something that can be leveraged to some adversary's advantage, at least in the first round. But the way the second pointer is produced makes it hard to keep track of the computation.

Now, let's take a step back. If we want to attack the memory-hardness property, what exactly is our goal? Well, the train of thought is the following: If we managed to compute the first 128 sequential bytes of the state of the scratchpad (after the second stage), without using the rest of it, then it would be a win. We can compute in steps the digest of the CryptoNight function, using just 128 bytes of the scratchpad in each step.

We can calculate the initialization of the first 128 bytes of the scratchpad, then move on to the second stage and finally compute the first part of our solution, running just the first step of the third stage. Then, we repeat the above for every other 128 bytes of the scratchpad (15,625 times). This will give us the digest of the function, using 256 bytes of memory ($2 \cdot 128$, in order to remember last round's outcome) and time complexity of the same order of magnitude as the proposed implementation. That would complete a successful attack.

The reason we failed is that it seems impossible to control the pointers to the addresses in the second stage. We cannot compute the final stage of the scratchpad, 128 bytes at a time. We need all of it to produce a correct computation. Let's see the details.

Without loss of generality, let the first address be a and the second address be b, within some loop. Then,

$$b = AES(a, SP_a)$$

Under the assumption that AES is a PRF, the above can be leveraged as we described but for one round at most. Then, control is lost. The reason is that the output of a PRF cannot be guessed with more than negligible probability. That means that we can't control the sequence of the addresses written in the second stage. Thus, we can't compute partially the final stage of the scratchpad. It has to be used as a whole.

6.2 Future Work

At first, our intuition was that the next step towards the goals of this thesis is to analyze the relation between the inputs and the outputs in each round. Maybe there is something there, we could not find. But with a closer look, it seems that if we cannot "fix" the relation to the equality one, then any control we might achieve by this analysis is lost in the next round, due to our assumption that AES is a PRF.

It is obvious that a successful attack on the AES function could lead to a successful attack on CryptoNight's memory-hardness property. However, AES being a PRF, is a reasonable assumption and it is expected to be hard to find a vulnerability in AES and a distinction between AES and PRFs. AES is a cryptographic primitive that has been thoroughly reviewed and checked.

We don't know yet, whether the reverse is true. Namely, whether a successful attack on CryptoNight's memory-hardness property is a step towards a successful attack on AES. This is a step that can follow our work, but it seems a really tricky and difficult problem.

Fresh point of view

If someone wants to continue our research, we recommend a huge step back. Maybe another model for the problem or something that is not produced from a similar point of view. As we presented in this thesis, we have found problems very early in our analysis and we couldn't pass beyond the second round of the second stage.

If someone wants to perform a cryptographic analysis on CryptoNight's memory-hardness property, then he/she is supposed to analyze this second stage. This second stage is the place where the origins and the basis of the memory-hardness property lie.

6.3 Epilogue

This thesis is the first, to our knowledge, to analyze the memory-hardness property of the CryptoNight function. We hope that our analysis is helpful to the researcher or the researchers, who would like to continue this effort. We tried to make this document a good start for further exploration of this interesting and valuable problem.

We wish good luck to the people that will continue this work and help the Monero community in their effort for a better world. Surveillance amplification in the physical and digital world is apparent in our days. We believe that progress in research for anonymity and privacy in the public domain, solving important privacy issues that still do not admit efficient solutions (see [75]), is something that many people seek. After all, privacy is a fundamental human right that many - if not all - human rights are based on. Without privacy, human rights, i.e., *freedom of speech*, collapse.

BIBLIOGRAPHY

[1] Ben AdidaSusan Hohenberger and Ronald L. Rivest. “Ad-Hoc-Group Signatures from Hijacked Keypairs”. In: (Feb. 2019).

[2] Joël Alwen and Vladimir Serbinenko. “High Parallel Complexity Graphs and Memory-Hard Functions”. In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*. STOC ’15. Portland, Oregon, USA: ACM, 2015, pp. 595–603. ISBN: 978-1-4503-3536-2. DOI: 10.1145/2746539.2746622.

[3] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. “Depth-Robust Graphs and Their Cumulative Memory Complexity”. In: (2016). URL: https://eprint.iacr.org/2016/875.

[5] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O’Reilly Media, Inc., 2014.

[6] Man Ho Au et al. “Short Linkable Ring Signatures Revisited”. In: *Public Key Infrastructure*. Ed. by Andrea S. Atzeni and Antonio Lioy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 101–115. ISBN: 978-3-540-35152-8.

[7] J. P. Aumasson. “SHA-3 proposal BLAKE”. In: *http://131002.net/blake/blake.pdf* (2010). URL: https://ci.nii.ac.jp/naid/10030667226/en/.

[8] Adam Back et al. “Hashcash-a denial of service counter-measure”. In: (2002). Original system developed in 1997. URL: ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf.

[9] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *Public Key Cryptography - PKC 2006*. Ed. by Moti Yung et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228. ISBN: 978-3-540-33852-9.

[10] Daniel J. Bernstein et al. “High-speed high-security signatures”. In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89. ISSN: 2190-8516. DOI: 10.1007/s13389-012-0027-1.

[16] Sean Bowe, Ariel Gabizon, and Matthew D. Green. “A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK”. In: *Financial Cryptography and Data Security*. Ed. by Aviv Zohar et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 64–77. ISBN: 978-3-662-58820-8.

[17] B. Bünz et al. “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.

[18] David Chaum and Eugène van Heyst. “Group Signatures”. In: *Advances in Cryptology — EUROCRYPT ’91*. Ed. by Donald W. Davies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 257–265. ISBN: 978-3-540-46416-7.

[25] Joan Daemen and Vincent Rijmen. “AES Proposal: Rijndael”. In: (1999).

[26] W. Diffie and M. Hellman. “New Directions in Cryptography”. In: *IEEE Trans. Inf. Theor.* 22.6 (2006), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638.

[27] Cynthia Dwork and Moni Naor. “Pricing via processing or combatting junk mail”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 139–147.

[28] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. “Key-evolution Schemes Resilient to Space-bounded Leakage”. In: *Proceedings of the 31st Annual Conference on Advances in Cryptology*. CRYPTO’11. Santa Barbara, CA: Springer-Verlag, 2011, pp. 335–353. ISBN: 978-3-642-22791-2. URL: http://dl.acm.org/citation.cfm?id=2033036.2033061.

[30] Harold M. Edwards. “A normal form for elliptic curves”. In: *Bulletin of the American Mathematical Society* 44.3 (2007), pp. 393–422. DOI: 10.1090/S0273-0979-07-01153-6.

[33] Niels Ferguson et al. “The Skein Hash Function Family”. In: (2008). URL: http://www.skein-hash.info/sites/default/files/skein1.1.pdf.

[34] Christian Forler, Stefan Lucks, and Jakob Wenzel. “Catena: A Memory-Consuming Password Scrambler”. In: *IACR Cryptology ePrint Archive, Report 2013/525* (2013). URL: http://eprint.iacr.org/2013/525/20140105:194859.

[35] Eiichiro Fujisaki. “Sub-linear Size Traceable Ring Signatures without Random Oracles”. In: *Topics in Cryptology – CT-RSA 2011*. Ed. by Aggelos Kiayias. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 393–415. ISBN: 978-3-642-19074-2.

[36] Eiichiro Fujisaki and Koutarou Suzuki. “Traceable Ring Signature”. In: *Public Key Cryptography – PKC 2007*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 181–200. ISBN: 978-3-540-71677-8.

[37] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 281–310. ISBN: 978-3-662-46803-6.

[38] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol with Chains of Variable Difficulty”. In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 291–323. ISBN: 978-3-319-63688-7.

[39] Praveen Gauravaram et al. “Grøstl – a SHA-3 candidate”. In: (2011). URL: http://www.groestl.info/Groestl.pdf.

[41] S. Goldwasser, S. Micali, and C. Rackoff. “The Knowledge Complexity of Interactive Proof Systems”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012.

[42] M. Hellman. "A Cryptanalytic Time-memory Trade-off". In: *IEEE Trans. Inf. Theor.* 26.4 (2006), pp. 401–406. ISSN: 0018-9448. DOI: 10.1109/TIT.1980.1056220.

[45] Dimitris Karakostas et al. "Cryptocurrency Egalitarianism: A Quantitative Approach". In: (2019).

[46] Kostis Karantias. "Constructing Interoperable Blockchains Using NIPoPoWs". In: (2019). URL: https://arctan.gtklocker.com/thesis.pdf.

[47] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. ISBN: 1584885513.

[50] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. "Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups". In: *Information Security and Privacy*. Ed. by Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 325–335. ISBN: 978-3-540-27800-9.

[51] Joseph K. Liu and Duncan S. Wong. "Linkable Ring Signatures: Security Models and New Schemes". In: *Computational Science and Its Applications – ICCSA 2005*. Ed. by Osvaldo Gervasi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 614–623. ISBN: 978-3-540-32044-9.

[53] Ralph C Merkle. "A digital signature based on a conventional encryption function". In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1987, pp. 369–378.

[57] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48.177 (1987), pp. 243–243. DOI: 10.1090/s0025-5718-1987-0866113-7.

[58] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *forum online* (2008). URL: http://bitcoin.org/bitcoin.pdf.

[59] Shen Noether. "Ring Signature Confidential Transactions for Monero". In: (2015). URL: https://eprint.iacr.org/2015/1098.

[61] Abhi Shelat Rafael Pass. "A Course in Cryptography". In: *A Course in Cryptography*. 2010.

[63] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565. ISBN: 978-3-540-45682-7.

[64] Phillip Rogaway. "The Moral Character of Cryptographic Work". In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1162. URL: https://web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf.

[65] Nicolas van Saberhagen. "CryptoNote v 2.0". 2013. URL: https://cryptonote.org/whitepaper.pdf.

[69] Information Technology Laboratory (National Institute of Standards and Technology). *Announcing the Advanced Encryption Standard (AES) [electronic resource]*. English. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, MD, 2001, 52 p. : URL: https://nla.gov.au/nla.cat-vn4183631.

[71] Nick Szabo. “The idea of smart contracts”. In: *Nick Szabo's Papers and Concise Tutorials* 6 (1997).

[75] N. Unger et al. “SoK: Secure Messaging”. In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 232–249. DOI: 10.1109/SP.2015.22.

[77] Hongjun Wu. “The Hash Function JH”. In: (2011). URL: http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf.

[78] Qianhong Wu et al. “Ad Hoc Group Signatures”. In: *Advances in Information and Computer Security*. Ed. by Hiroshi Yoshiura et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 120–135. ISBN: 978-3-540-47700-6.

Web resources

[4] David Andersen. *Personal blog*. URL: https://da-data.blogspot.com/2014/08/minting-money-with-monero-and-cpu.html.

[11] *Bitcoin Explorer wiki*. URL: https://github.com/libbitcoin/libbitcoin-explorer/wiki/bx-base58-encode.

[12] *Bitcoin Forum*. URL: https://bitcointalk.org/.

[13] *Bitcoin Wiki*. URL: https://en.bitcoinwiki.org/wiki/.

[14] *Bitmain*. URL: https://www.bitmain.com/.

[15] *Blockchain*. URL: https://www.blockchain.com/.

[19] *CoinDesk*. URL: https://www.coindesk.com/.

[20] *Coinhive*. URL: https://github.com/cazala/coin-hive/.

[21] *Cryptoanarchy Wiki*. 1998. URL: https://cryptoanarchy.wiki/.

[22] *CryptoNote coins*. URL: https://cryptonote.org/coins.

[23] *CryptoNote description*. URL: https://cryptonote.org/inside.

[24] *CryptoNote Test Address*. URL: https://xmr.llcoins.net/addresstests.html.

[29] The Economist. *The great chain of being sure about things*. The Economist Newspaper Limited. 2015. URL: https://www.economist.com/.

[31] *Elliptic Curve Digital Signature Algorithm (wiki)*. URL: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

[32] *Esperanto language*. URL: http://esperanto.net/en/.

[40] *GetMonero*. URL: https://getmonero.org/.

[43] *Investopedia dictionary*. URL: https://www.investopedia.com/dictionary/.

[44] *Invisible Internet Project*. URL: https://geti2p.net/en/.

[48] *Kovri Gitlab repository*. URL: https://gitlab.com/kovri-project/kovri.

[49] *Kovri I2P Project*. URL: https://kovri.io/.

[52] Gregory Maxwell. *Confidential Transactions*. URL: https://elementsproject.org/.

[54] *Monero fees charts*. URL: https://bitinfocharts.com/.

[55] *Monero project - Github*. URL: https://github.com/monero-project.

[56] *Monero stack exchange*. URL: https://monero.stackexchange.com/.

[60] *Open Whisper Systems*. URL: https://signal.org/.

[62] *Reddit*. URL: https://www.reddit.com/.

[66] *Segregated Witness proposal*. URL: https://github.com/bitcoin/bips/.

[67] Seigen et al. *CryptoNight*. URL: https://cryptonote.org/cns/cns008.txt.

[68] Riccardo Spagni. *On Twitter*. URL: https://twitter.com/fluffypony/status/974670746772496385.

[70] *Standards for Efficient Cryptography Group*. URL: https://www.secg.org/.

[72] TeamKeccak. *Keccak*. URL: https://keccak.team/keccak.html.

[73] *The Monero project*. URL: https://monero.org/.

[74] *Tor Project*. URL: https://www.torproject.org/.

[76] *Vasilis Agiotis*. URL: https://www.linkedin.com/in/vassilis-agiotis-829928172/.

[79] *Zcash*. URL: https://z.cash/.

[80] *Zero-Knowledge proofs science*. URL: https://zkp.science/.