PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SPEEDING UP MONERO'S BALANCE COMPUTATION

RAIMUNDO HERRERA SUFÁN

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
JUAN REUTTER

Santiago de Chile, August 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SPEEDING UP MONERO'S BALANCE COMPUTATION

RAIMUNDO HERRERA SUFÁN

Members of the Committee:
JUAN REUTTER
DOMAGOJ VRGOČ Domagoj Vrgoč
IGNACIO BAIXAS
WERNHER BREVIS

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, August 2021

Gratefully to my family and friends.

ACKNOWLEDGEMENTS

Thanks to my advisor, Juan Reutter, for always being there when I needed him. He provided guidance and knowledge and had the patience to support me when things were difficult, particularly through the pandemic. This work had demanding challenges that were easier to tackle with his full wisdom, support and belief in me, alongside the trust he gave me to approach this investigation in the way I did.

I also want to thank Martín Ugarte and Niki Hamidi-V. Martín guided me in the initial stages of this investigation and provided ideas alongside a vast understanding of the subject. Niki helped me in the later stages of this work with her coding skills and admirable drive to understand the complex Monero source code.

I would also like to thank my workplace, Platanus. They did everything they could to make the challenge of combining work and investigation an easy task.

Finally, I want to thank my family for always believing in me while not pressuring me in any way. I felt support and trust at all times.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

LIST OF FIGURES viii

LIST OF TABLES ix

ABSTRACT x

RESUMEN xi

1. Introduction 1
   1.1. Problem statement . . . 4
   1.2. Thesis outline and structure . . . 5

2. Preliminaries 6
   2.1. General elliptic curve notation . . . 6
   2.2. Monero keys and transaction scheme . . . 6
      2.2.1. Key pairs . . . 6
      2.2.2. One Time Addresses . . . 7
      2.2.3. Transactions directed to a user . . . 7
      2.2.4. Outputs spent by the user . . . 8

3. Computing the balance through Consolidation Transactions 9
   3.1. Consolidation Transactions . . . 10
   3.2. Balance keys and wallet implementation . . . 11
   3.3. Improvements . . . 13
   3.4. Churning and security concerns . . . 15
   3.5. Wallet Recommendations . . . 17

4. Indexing and searchable encryption 19
   4.1. Indexing transaction hashes . . . 21

4.1.1. Issues, attacks and security concerns 21
4.2. Indexing encrypted transaction hashes 23
4.2.1. Improvements 24
4.2.2. Remaining issues, attacks and security concerns 24
4.3. Indexing unforgeable encrypted transaction hashes 25
4.3.1. Using non deterministic encryption 26
4.3.2. Using a shared secret 27
4.3.3. Improvements 28
4.3.4. Remaining Issues, attacks and security concerns 28
4.4. Indexing recapitulation 29
4.5. Searchable encryption 30
4.5.1. PEKS scheme for Monero 31
4.5.2. PEKS procedure in Monero 33
4.5.3. Improvements 34
4.5.4. Remaining issues, attacks and security concerns 34

5. Privacy preserving outsourced services 36
5.1. Revocable indexing service 37
5.1.1. Sender procedure 37
5.1.2. Indexing procedure 38
5.1.3. Service usage 39
5.1.4. Revoking access 40
5.1.5. Trustless implementation 41
5.1.6. Improvements 42
5.1.7. Remaining issues and security concerns 43
5.2. Trapdoor transaction yielder 44
5.2.1. Procedure 45
5.2.2. Volatile wallets 46

6. Conclusions 48

REFERENCES 50

APPENDIX 56

A. First Appendix 57

A.1. Computer specifications 57

A.2. Syncronization times 57

LIST OF FIGURES

3.1 Blockchain with Alice's transactions highlighted in green 9

3.2 Scan depth for an uninformed balance computation 10

3.3 Blockchain with Alice's last CT highlighted in blue 11

3.4 Scan depth for a CT-aided balance computation 12

A.1 Blocks per month between December 2019 and May 2021 59

A.2 Average transactions per block for the months between December 2019 and May 2021 60

LIST OF TABLES

A.1 Restore wallet synchronization times 58

ABSTRACT

Cryptocurrencies have established themselves as relevant digital assets that aim at becoming the main medium of exchange for the coming decades. With the fast adoption and increased user base, usability and privacy have become crucial aspects for their success. Monero is a digital currency that offers properties required for any asset to be considered a viable money replacement, especially with regards to privacy and security. However, to offer those features, Monero compromises usability in day-to-day operations, such as the balance computation. This particular operation is slow due to the mandatory need to scan the whole Monero blockchain in order to perform it. In this work, we provide ways to decrease the time taken for the balance operation while minimally compromising non-critical private elements. Specifically, we first introduce a procedure to generate multiple consolidation transactions that allow users to avoid a full scan of the blockchain each time they need to retrieve their balance, reducing computation times significantly by only spending minimal amounts of money. We also provide schemes that use indexing techniques to retrieve one user transaction history in less time and therefore enable to compute the balance faster, only revealing the number of transactions to outside observers. We finally show how to take advantage of the proposed schemes by outlining third-party services and wallets that help the user offload part of the balance computation in new and secure ways. Throughout all our work, we discuss the compromises and trade-offs incurred when modifying the current state of Monero's protocol by introducing our various improved proposals since we aim to keep most privacy guarantees while offering usability improvements.

Keywords: Monero, balance, indexing, blockchain.

RESUMEN

Las criptomonedas se han establecido como activos digitales relevantes que apuntan a convertirse en el medio de intercambio principal en las próximas décadas. Con la rápida adopción y gran número de usuarios, la usabilidad y privacidad se han convertido en aspectos críticos para su éxito. Monero es una moneda digital que ofrece propiedades requeridas por cualquier activo que pretende ser considerado un reemplazo viable del dinero, especialmente respecto a la privacidad y seguridad. Sin embargo, para ofrecer esas características, Monero compromete la usabilidad de ciertas operaciones de uso diario, como el cálculo del balance de un usuario. Esta operación en particular es lenta debido a que para realizarla es necesario escanear la totalidad del blockchain de Monero. En este trabajo, presentamos diversas formas en las que disminuir el tiempo que toma el cálculo del balance, comprometiendo mínimamente elementos privados no críticos. Específicamente, introducimos un procedimiento para generar múltiples transacciones de consolidación que permiten a los usuarios evitar escaneos completos del blockchain cada vez que necesiten obtener su balance, reduciendo significativamente los tiempos de la operación gastando mínimas cantidades de dinero. Además proporcionamos esquemas que usan técnicas de indexación para obtener el historial de transacciones de un usuario en menos tiempo y que por lo tanto posibilitan realizar el cálculo de balance más rápido, solamente revelando la cantidad de dichas transacciones a observadores externos. Finalmente mostramos cómo aprovechar las propuestas a través de la descripción de servicios y billeteras manejadas por terceros que ayudan al usuario a ahorrarse parte del cálculo del balance de forma segura. A lo largo de todo nuestro trabajo, discutimos las concesiones incurridas al modificar el estado actual del protocolo de Monero e incorporar nuestras mejoras, dado que apuntamos a mantener la mayor parte de las garantías de privacidad mientras ofrecemos mejoras en usabilidad.

Palabras Claves: Monero, balance, indexación, blockchain.

1. INTRODUCTION

Monero[1] is a digital currency launched in 2014 and currently established as a top 30 cryptocurrency in terms of market capitalization[2]. It has a special focus in privacy, security and untraceability. To achieve those features Monero relies heavily on advanced cryptography, enabling users to safely spend their money without revealing sensitive information from their transaction history. Namely, users do not disclose who they have received money from, who they have sent money to, or the amounts transferred. However, in order to do so, daily operations that users must perform have their efficiency compromised.

This special focus in privacy differentiates Monero from other currencies. By spending Monero, users can rest assured that no unintended party can observe how much money they own or how they decide to use it. Unlike other currencies, Monero uses a distributed ledger of transactions –the blockchain– that is opaque, which ensures that even when funds are verifiable and secure, the anonymity of users is still guaranteed. Because Monero is private, coins can not be traced, which in turn makes the currency fungible[3], one of the main properties desired for currencies.

In any cryptocurrency, computing the balance is a crucial and frequent task for users. In Monero, nonetheless, as its aim is to keep every user-related piece of information concealed from unwanted observers, that operation requires a full scan of the network transaction history[4]. This brings the inconvenience of performing time extensive operations, which presents the user with an explicit trade-off between speed and privacy.

[1] `https://www.getmonero.org/`.

[2] According to `https://coinmarketcap.com/` as of June 2021.

[3] Fungibility is the property of a currency where two units can be mutually substituted and the substituted currency is equal to another unit of the same size, regardless of its origin or any other characteristics (*Moneropedia: Fungibility*, n.d.).

[4] A partial scan can be performed if the user knows the date of creation of the wallet and can therefore limit the scan to that particular moment. However, since that point, a full scan of every transaction is needed. Moreover, if the date of creation is unknown, a full transaction of the whole blockchain is required.

Monero's privacy centric approach implies that this trade-off becomes relevant. In other cryptocurrencies without focus in privacy, like Bitcoin, the time and space requirements for computing a user's balance can be delegated to external services, commonly referred as third-party wallets. Users can then use these wallets to access their transaction history without the need for any complex operation at all. Unfortunately, in Monero there is currently no way of delegating this computation without revealing information that is intended to be private. Because of the heavy focus in privacy, Monero wallets are designed to be used locally in a device the user trusts. Moreover, the community recommends (Reddit, 2018d) running Monero nodes locally and never trusting remote wallets unless extremely necessary (Reddit, 2019b; Monero-Hax123, 2018), because using them implies giving away keys that are sensitive or exposing to tracing (Tramèr, Boneh, & Paterson, 2020). Following those crucial recommendations entails a slow balance computation.

Our work focuses on both providing ways to compute the balance for a user faster, and describe ways to further granularize permissions in Monero to avoid compromising privacy, while leveraging third parties and services to perform day-to-day Monero tasks. We focus on two challenges that make the user experience in Monero greatly improvable, firstly the inability to compute the balance quickly and directly in user wallets, and secondly, the inability to rely on third parties that can speed up those operations without compromising privacy (Reddit, 2018f, 2019c; Alex & Herrera, 2020).

Our contribution is composed by three separate parts to tackle the described problem:

(i) Our first approach is to shorten the transaction history scan by using a procedure known as *churning*. This approach is handled at a wallet level without modifying the Monero protocol and achieves significant improvements in the time spent to compute the balance[5].

[5] We contribute a Monero fork with our modifications for the wallet available in `https://github.com/rjherrera/monero/tree/consolidation-tx`.

(ii) Our second approach is to modify the Monero protocol to index transactions and use searchable encryption. We discuss their practicality, impact and repercussions in the privacy and speed trade-off.

(iii) Our third approach shows applications that take advantage of the proposed modifications to the protocol as third party services and wallets. We explicitly outline how a user could take advantage of them in the Monero scenario to gain speed and usability in the balance task.

Additionally our contribution provides an analysis of the trade-off incurred by the application of our proposals. While the main focus of Monero is to provide untraceability and privacy to users, we show how to accelerate operations without compromising that main focus majorly.

The three main approaches we provide to solve the problem affect the trade-off differently. In the first proposal, we gain efficiency from a practical standpoint while not improving in time complexity. However, we do not compromise privacy. Our second proposal sacrifices minor user elements that are currently kept private in order to achieve significant time gains in terms of complexity. Our third approach compromises even less privacy and gains efficiency both in practice and in terms of complexity, but also delivers new ways to interact with third parties that move the weight of the trade-off from the protocol, to the interactions with them.

With those three contributions we provide new ways to improve Monero's protocol while also providing implementable solutions. Our proposals can help users obtain their balance faster, allow them to have more portable wallets and leverage third-party software without majorly compromising privacy by further dividing responsibilities in information safeguarding. In addition, our contribution is not only directed to the Monero community. Over the last few years, we have seen more popular cryptocurrencies like Bitcoin adopt techniques[6] already applied in smaller yet more private currencies like Monero, making

[6] Schnorr signatures is a notable example, as it is used widely in the Monero protocol whereas it is set to be adopted in Bitcoin after consensus is met (Shinobi & Folkson, 2021).

our contribution –and most investigations in Monero– potentially beneficial for the whole community.

1.1. Problem statement

The main problem that guides our work is the BALANCE problem. We give a general outline for this problem and the guarantees that should be met in order for a solution to be viable. We further refine this problem in the following sections.

In Monero, a user Alice with a view key pair (k^v, K^v) and a spend key pair (k^s, K^s), can compute her balance $\mathcal{B}$. To this end, we assume the current state of Monero is given by a blockchain B containing blocks $b_1, \ldots, b_m$.

Computing the balance for Alice involves searching for the following transactions:

(i) Every transaction directed to an address derived from Alice's public keys.

(ii) Every transaction that uses Alice's outputs as real inputs for other transactions.

The first set of transactions corresponds to every transaction directed to Alice, the *incoming funds*, while the second set corresponds to every transaction that the user has sent, the *outgoing funds*. Subtracting the second set amounts to the first set amounts yields the user balance $\mathcal{B}$.

Given those definitions, we can define the problem in which we focus as the following:

> The BALANCE problem.
>
> **Input**: The view key pair (k^v, K^v), the spend key pair (k^s, K^s).
>
> **Output**: The balance $\mathcal{B}$ for the user designated by K^v.

In order for the BALANCE problem to be solved satisfactorily, we impose requirements or guarantees that need to be met as using the solution should not compromise Monero's granted privacy. The guarantees are the following:

G1 Only the owner or authorized third parties can obtain the balance.

G2 Only the owner or authorized third parties can obtain any feature of the sets used to compute the balance, for example, the amount of incoming transactions or the total number of them.

G3 Transaction addresses and the corresponding K^v can not be associated by unwanted third parties.

G4 Security guarantees provided by Monero, unrelated to the balance obtention, must still stand.

1.2. Thesis outline and structure

In Chapter 2 we present the notation and definitions that will be used throughout the whole work. We specify the basics from which our work will be built, specifically regarding elliptic curve cryptography and Monero operations.

Through Chapter 3 we introduce the notion of Consolidation Transactions, we show how to use them and how to approach the BALANCE problem with them. Additionally we show a wallet implementation and the improvements achieved.

Chapter 4 proposes two techniques to approach a reduced version of the BALANCE problem. We constructively present an indexing scheme for the problem, which entails publicly indexing transactions without majorly compromising privacy. We also apply a searchable encryption framework and adapt it to approach the problem. For both subjects we provide a discussion for the security trade-offs.

In Chapter 5 we show concrete applications for both approaches presented in Chapter 4. We show how to build third party services that implement the techniques described in that chapter and how users can benefit from them.

Finally Chapter 6 summarizes the results and contributions of this work and proposes future lines of work.

2. PRELIMINARIES

2.1. General elliptic curve notation

As in Monero itself, in this work we use elliptic curve points and operate with them constantly.

We largely follow the notation introduced by (Alonso et al., 2020) in From Zero to Monero, therefore we refer to integers with lower case letters and to curve points with upper case letters, for example k and K. At the same time, they are interpreted as the private and public key from a pair, respectively.

We also use the curve generator G, the order l for the curve[1] and sample integers randomly from $\mathbb{Z}_l$[2].

Furthermore, it is important to note that, as it is done in (Alonso et al., 2020), hash functions are applied to points and integers. We denote hash functions mapping to integers as $\mathcal{H}_n$ and mapping to points as $\mathcal{H}_p$.

Any extra chapter specific notation will be introduced when necessary throughout this work.

2.2. Monero keys and transaction scheme

2.2.1. Key pairs

In Monero our user Alice has two pairs of keys. The first pair is the view key pair, with both a private and a public key (k^v, K^v). They private view key allows Alice to observe transactions and amounts directed to her. The second pair is the spend key pair, with both

[1]This means arithmetic between scalars is $\bmod l$.

[2]To sample an integer randomly from $\mathbb{Z}_l$ is equivalent to sample it from $\{1, 2, 3, \ldots, l-1\}$, therefore $\mathbb{Z}_l$ is all integers $(\bmod l)$.

a private and a public key (k^s, K^s). As the name suggests, the private spend key allows her to spend her funds, alongside checking if any funds have been already spent.

2.2.2. One Time Addresses

The public view key and the public spend key are both part of Alice's address, but when creating transactions, neither of them are exposed in the blockchain. Instead, Monero relies on One Time Addresses (OTA). An OTA is an address derived in a one-way procedure from the receiver's public view and spend key. This way OTAs are exposed in the blockchain without revealing the public keys they came from.

OTAs are derived in a way that allows only the sender and receiver to create them. They are built by the sender using a random number r, the public view key from the receiver K^v, the curve generator point G and the public spend key from the receiver K^s, as follows:

$$K^o = \mathcal{H}_n(rK^v)G + K^s$$

2.2.3. Transactions directed to a user

In order for Alice to verify if a transaction OTA is addressed to her, she needs to create a candidate OTA and check if they match. As r is never shared, she does that from her keys and the transaction public key rG which is posted in the transaction by the sender.

Knowing that for any key pair (k, K) it stands that $K = kG$, Alice can recreate the OTA using rG and her private view key k^v as:

$$K'^o = \mathcal{H}_n(rk^vG)G + K^s$$

It follows from the above that K'^o is the same OTA created by the sender, as $rk^vG = rK^v$, therefore $K^o = K'^o$. An observer can not derive an OTA from Alice's public keys unless he knows r or k^v[3].

2.2.4. Outputs spent by the user

To verify whether an output addressed to Alice has been spent, she needs to compute a candidate key image and check if it matches any key image stored in the blockchain. A key image is another public key of a transaction and it is used to verify double spend.

To compute the key image for an owned output, Alice needs the private pair of the OTA K^o, namely k^o, which is computed using both private keys and the transaction public key, this way:

$$k^o = \mathcal{H}_n(rk^vG) + k^s$$

The key image for that output is $\widetilde{K}^o = k^o\mathcal{H}_p(K^o)$. If $\widetilde{K}^o$ has appeared before in other transactions available in the blockchain, then the transaction has been spent.

[3]Monero and its elliptic curve both rely heavily on the Discrete Logarithm Problem (DLP). While it stands, it is infeasible for an observer to obtain r from rG and k^v from K^v (Alonso et al., 2020).

3. COMPUTING THE BALANCE THROUGH CONSOLIDATION TRANSACTIONS

While the security provided by the current Monero protocol is one of its main features, this comes at the expense of the efficiency of the balance computation problem. Indeed, while users may expect the balance computation to happen at near-instant speed –as one is accustomed to when dealing with banks, or even other cryptocurrencies–, this is generally achieved using indexes or random accesses that cannot be immediately deployed in Monero due to its policy of obfuscating the amount, senders and receivers of each transaction.

In Monero, the balance computation is performed via a complete scan of the blockchain. As illustrated in Figure 3.1, the blockchain B is a collection of n blocks b_i containing m_i transactions t^i.

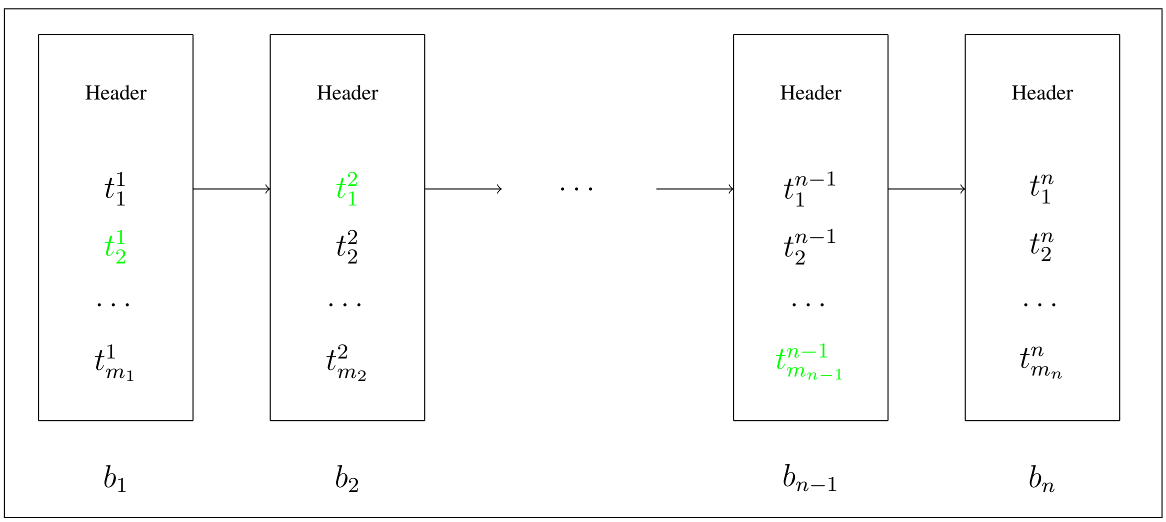

Figure 3.1. Blockchain with Alice's transactions highlighted in green

To compute the balance, Alice needs to scan the blockchain and the scan depth is not limited in any way[1], as every block needs to be scanned for incoming or spent transactions, as shown in Figure 3.2. Of course, Alice may decide to store her previously computed balance or her unspent transactions in her wallet. This comes with the risk of storing

[1] In many wallet implementations (like the official one found in the monero-project repository), the user is prompted for a date or a block height to avoid the full scan (Hyc, 2018). Typically users introduce the estimated address creation date, however, if a user does not provide this information, the scan is full.

sensitive information on a device and with the inconvenience of relying on one specific wallet in order to have fast answers for her queries.

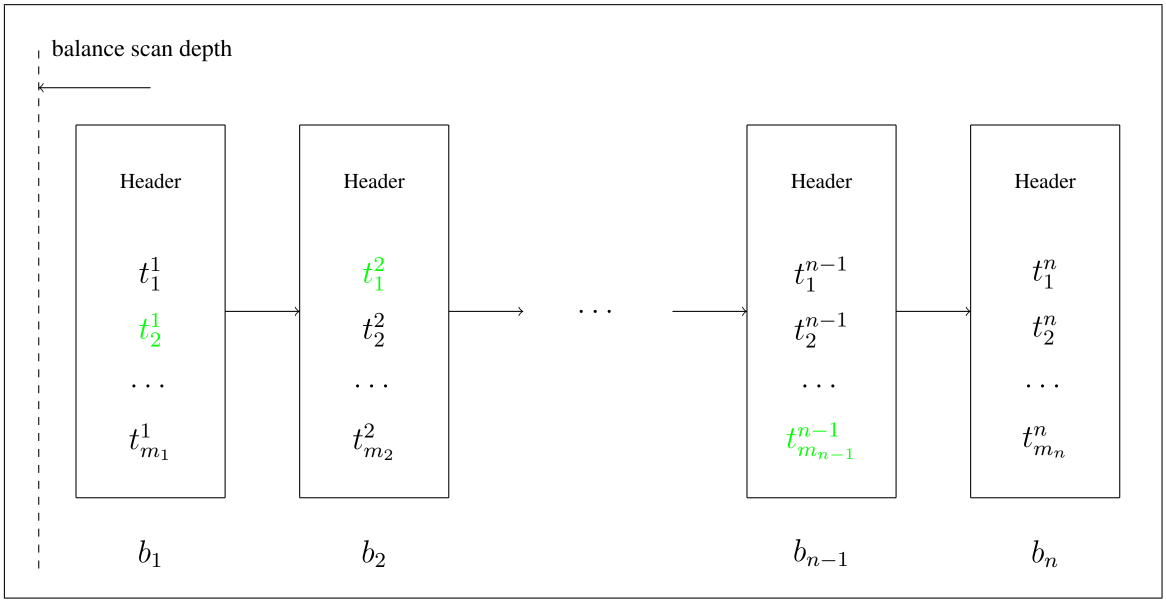

Figure 3.2. Scan depth for an uninformed balance computation

In this chapter we focus on the BALANCE problem by discussing one approach that does not need any modifications to the underlying Monero protocol. The general procedure to obtain the balance remains as is, but our user Alice is able to reduce the time spent computing her balance, while not compromising her privacy or relying on the security of trusted wallets.

Our approach uses the procedure known as *churning* to successfully escape the apparent trade-off between keeping information private using the Monero balance computation procedure *as is*, and accelerating it by storing private information in wallets. We discuss churning and its security concerns in a later section.

3.1. Consolidation Transactions

The idea of a consolidation transaction (CT) is to aggregate each of Alice's unspent transactions into a fresh new transaction, which is then sent back to herself.

Once a CT is submitted by Alice to the network and verified by miners, it will be included in a new block in the blockchain as displayed in Figure 3.3[2]. This procedure does not differ in any way to submit a normal transaction to the network.

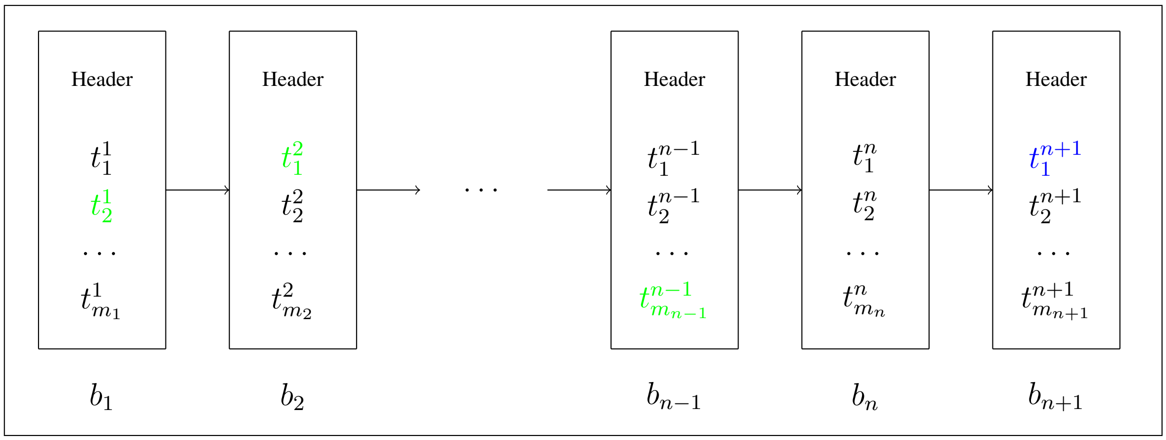

Figure 3.3. Blockchain with Alice's last CT highlighted in blue

If Alice remembers her last CT, she can successfully limit the scan depth needed to compute her balance, as every transaction addressed to her before the CT was performed, is already considered in it, making a further scan irrelevant and unnecessary[3]. Specifically, in contrast to the normal and uninformed balance computation described in Figure 3.2, there is no need to scan old Alice's transactions t_1 to $t_{m_{n-1}}$, as all the funds have been spent and are now completely available in the last consolidation transaction t_1^{n+1}. The new scan depth for the described scenario with a new CT is illustrated in Figure 3.4.

3.2. Balance keys and wallet implementation

While Consolidation Transactions do not immediately reduce the time required for the balance, one can speed up the process simply by noting down the hash of the most recent CT, as this effectively limits the scan depth to every block which is newer than this CT

[2]Regardless of the highlighting and colouring, Alice's transactions are actually not distinguishable from the others included in every block.

[3]There is an edge case in which there are too many transactions in the pool, and a new transaction addressed to Alice is added to a block preceding the block in which the CT is added but after the last block considered to create the CT. To address this, users could remember the last CT before the one being performed.

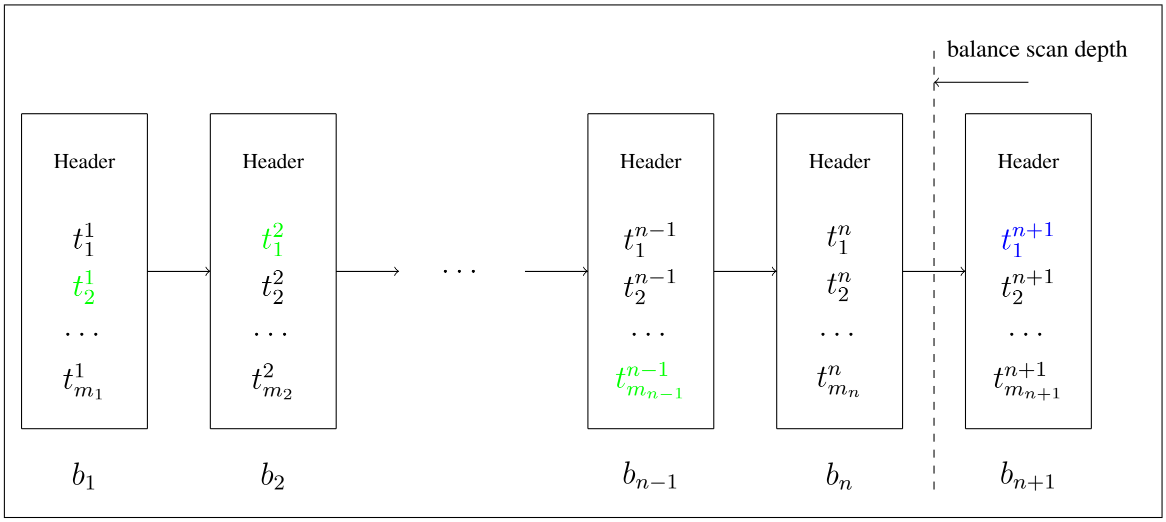

Figure 3.4. Scan depth for a CT-aided balance computation

(any unspent transaction older than this CT has already been consolidated into it). From now on, we refer to this hash as the *balance key* of Alice.

Our proposal is to improve wallets, allowing them to receive a balance key that serves to limit the scan depth of the blockchain. This is analogous to limiting the scan by introducing a date or a block height, which are already supported in the official Monero CLI Wallet[4].

The implementation receives a balance key as an input from the user, calculates the block height corresponding to it, and use this height to limit the scan up to this point. Further, Alice may decide to have multiple balance keys in order to split balance in more than one CT[5].

Our implementation is publicly available in our fork of the official Monero repository[6]. Specifically, our approach prompts the user for a CT through a balance key. Once the wallet has that information, it obtains the transaction directly from the node with the already

[4]There is no official documentation for the official CLI wallet, however, unofficial documentation can be found in `https://monerodocs.org/interacting/monero-wallet-cli-reference`.

[5]Splitting balance in multiple transaction is sometimes recommended across the community as there are concerns of possible traceability issues for recurrent churning (Reddit, 2018a)

[6]The code for this implementation can be found in `https://github.com/monero-project/monero/compare/master...rjherrera:consolidation-tx`.

available procedure to retrieve a singular transaction in the Monero Wallet CLI, and extracts the block height. With the height, the wallet sets it as the limiting point following the same procedure already implemented for dates and explicit block heights.

3.3. Improvements

In this section we discuss the gains obtained by using consolidation transactions. Improvements are evident in terms of balance computation time, but we also cover the achievements in security and wallet portability. We additionally provide a cost analysis for users frequently performing CTs.

In practice, our proposal removes the need of a full scan of the blockchain to obtain the balance. This means that Alice, with an arbitrarily old address, can reduce the scan time required to obtain her balance only using consolidation transactions. Reducing the scan time then translates to a reduction in the time needed to compute the balance, which can save up to 99% of the balance time for longstanding users[7]. A detailed analysis comprising experiments performed on an actual computer running the Monero official wallet and node, comparing scans of different depths and longevity is provided in the appendix.

Several contributors to the Monero project (Reddit, 2018g, 2018e, 2018b, 2019a) have indicated that creating this type of transactions –namely churning–, increases the mixing factor of the blockchain therefore improving the security of the whole network. We discuss the security concerns and implications of churning in the next section.

Wallets play a big role in the time spent by a user while performing daily operations. Usually Alice depends on her cold wallet(s)[8] to perform them. By doing so, her transaction history and sensitive information is stored there. If the information is lost, a full scan is

[7]Depending on the device and the history of transactions, full scans to compute the balance can take up to days. With a 2-days-old CT, the same result can be obtained with a sub 1 minute scan as we show in the appendix.
[8]Cold wallet is the term used to refer to wallets that are not connected in any way to the internet and that are used to safely store keys and relevant information, alongside signing transactions that are posted to the network through hot wallets, connected to the internet.

required to obtain the information again. Changing from a wallet to a new one only using Alice's private keys presents a time consuming operation, because she will no longer have access to the unlocked information the wallet stored, therefore having to perform several operations to rebuild her history.

With CTs, Alice can use her keys and the transaction identifier to initiate a completely new wallet. This wallet can be used without the need of a full scan, because a partial scan until the CT will suffice in order to obtain her balance and use her transaction to send funds. This implies gain in terms of portability as instantiating new wallets and avoiding the need of time consuming scans to do so, will effectively reduce the need of keeping long-term wallets with pre-computed sensitive information in order to perform operations quickly.

The only caveat of using CTs is that each CT entails the payment of the normal fee of a transaction. This implies that Alice could potentially spend transaction fees for dozens of transactions per year that do not carry out a funds exchange between users.

However, transaction fees in Monero have stayed in average under a tenth of a dollar for the last few years[9]. In 2020 the average transaction fee was approximately 0.0296 USD per transaction, fluctuating between 0.0072 USD and 0.0911, while not showing any trend towards increasing or decreasing significantly (BitInfoCharts, 2021).

CTs are not mandatory, hence the cost should be perceived as a payment for improved efficiency, reduced synchronization times and portability. We encourage wallets that implement this solution to warn the user about the incurred costs and allow the user to fine tune the CTs frequency as a function of not only time tolerance but of willingness to pay.

[9]Transaction fees dropped significantly in 2018 after the hard fork that introduced bulletproofs to the protocol (O'Leary, 2018; Aki, 2018).

3.4. Churning and security concerns

Using a single transaction to send all unspent outputs of a user to himself is not a new proposal. In fact, churning has been widely discussed in the Monero community (Reddit, 2019a), and it has been considered as a method to improve one user's privacy (Stackexchange, 2017b).

By churning, Alice mixes her incoming transactions –her unspent outputs–, with others from the network as Monero uses ring signatures with randomly[10] selected outputs to obfuscate senders and make the sender indistinguishable between the chosen set (Alonso et al., 2020).

Therefore, churning increases the amount of fresh outputs[11] available to select as decoys for other transactions in the whole Monero network, and decreases the probability for a given output to be identified as the real output being spent in a transaction (Reddit, 2018c; Alonso et al., 2020).

However, churning could be detrimental for privacy if used without care. Specifically, if users establish repetitive patterns when churning, attackers could try and deduce which outputs are the real ones used in transactions and void the objective of ring signatures (Stackexchange, 2019).

To achieve this, an attacker can identify the pattern and flag repetitive transactions as churning transactions[12]. Knowing which transactions are churning transactions can aid an attacker to discover when funds are effectively sent out and not re-churned, as the penultimate churning transaction would not appear in the last churning transaction as an input, with high probability.

[10] Outputs are selected from a gamma distribution across the range of historical outputs, and they must comply with a minimum amount of confirmations (Alonso et al., 2020).

[11] The random selection also considers a time frame, favouring newer transactions (Alonso et al., 2020).

[12] While doing this is not an easy task, it could be performed with statistical analysis (Möser et al., 2017). It could also be done by guessing the interval and trying to identify outputs as inputs in a chain of transactions.

The risk of churning transactions to be identified by attackers can be mitigated by using irregular time intervals and splitting transactions into multiple churning transactions.

By using irregular and seemingly random time intervals, the pattern recognition needed to flag transactions as churning becomes infeasible, as transactions would mimic normal spending behaviour found in the network. Moreover, splitting the balance in multiple churning transactions disallows the linearity of the analysis, as one normal transaction and one churning transaction could both be used as inputs for multiple churning transactions.

Apart from the risks related to churning, there are minor concerns worth analyzing raised by our balance key proposal. To effectively use CTs, Alice should keep track of a new key, the balance key. This identifier is already available for any user and it references a specific transaction, with obfuscated senders, recipients and amounts. If an attacker knows that one specific balance key is associated to a user, he only knows that the transaction holds balance, but does not know either the amount, nor the real inputs or outputs.

However, all of Alice's coins are kept in one single transaction[13] after the consolidation, so using CT can be conceived as the introduction of a *single point of failure*. This could be relevant as an informed attacker could target one transaction that holds all of Alice's balance, nevertheless, this transaction still complies with the security standards of any other transaction, so the risk is minimal and it only becomes apparent when the balance key is revealed, so, as any other key, the recommendation is to keep it private.

[13]Unless Alice follows the recommendation of creating more than one CT per churn. If Alice holds multiple balance keys, she can input the oldest un-consolidated transaction to the wallet implementation provided, as it will be the limiting block for the balance computation.

3.5. Wallet Recommendations

Based on the the user's waiting tolerance, consolidation transactions should be done with a certain frequency. Wallets should optionally recommend the user to do a CT whenever the last transaction gets too old[14]. It is of high importance to be careful with the recommendations to avoid exposing user patterns.

One way of doing this is for the wallet to ask Alice for a maximum amount per month to spend in transaction fees. As CTs imply spending a transaction fee every time a new one is performed, the wallet should estimate the amount of transactions that can be done monthly to avoid going over the defined limit. This number can be obtained as:

$$d = 30 \cdot \frac{\bar{f}}{l}$$

where d is the amount of days that the wallet should wait from the last CT performed, $\bar{f}$ the average transaction fee for the last month and l Alice defined expendable limit. However, we do not recommend to use d as a fixed interval, it should be used as a parameter for a random distribution in order to obtain the next moment for a transaction. Further improvements are encouraged in order to avoid pattern disclosure.

Another way is to allow Alice to input a maximum waiting time to compute her balance. This way, the wallet would need to keep track of the time spent computing the balance every time the operation is performed in comparison to the amount of new transactions. With that information then estimate the time that the operation of computing the balance would take by observing the amount new transactions since the last synchronization, and factoring it by the statistics gathered. One approach would be to compute:

$$n = l \cdot \frac{n_{last}}{t_{last}}$$

[14]Monero Wallet CLI already implements `sweep_all` and `sweep_single` methods, that allow churning (of every address or one address, respectively). Both methods can be leveraged in order to implement consolidation transactions and recommendations.

where n is the maximum amount of transactions that can be synchronized in the time limit l Alice defined, n_{last} is the amount of transactions synchronized in the last operation and t_{last} the time elapsed for the synchronization of those transactions. With n, the wallet can then suggest the user to create a CT as a function of the transactions being added to the blockchain.

Using the maximum waiting time strategy inherently yields a seemingly random pattern for new CTs as it depends on the amount of transactions organically posted to Monero's blockchain, however, the calculations could be reverse engineered by an attacker, therefore we recommend introducing a random factor to shield against them.

Both strategies, and many others, can be used in order to notify the user whenever a new CT should be made. This way, the periodicity of the operation is handled by the wallet, and the user always keeps his balance close to the present time, therefore the balance retrieval times are bounded and manageable.

4. INDEXING AND SEARCHABLE ENCRYPTION

We have already discussed how consolidation transactions can help to ameliorate the range of a blockchain scan, but transactions after the last consolidation still need a full scan. In this chapter we deal with the question: can we build additional data infrastructure so that we can solve the BALANCE problem in sub-linear time?

CTs provided an approach that improved the average case for obtaining the balance in its whole conception, as both the set of incoming and outgoing transactions can be obtained faster when using CTs. However, in this chapter we propose a rather different approach that implies modifications to Monero's underlying protocol: we use indexing techniques in order to retrieve transactions faster.

Therefore in this chapter we only focus on transactions directed to the user's K^v, that is, the first group mentioned in the BALANCE problem definition. It is important to note that the main improvements gained from solving the problem are located in this set of transactions –the incoming ones–, and not the outgoing, as the latter are a subset of the former[1].

Hence we further refine the BALANCE problem defined previously, to focus on the retrieval of incoming transactions directed to the user, as follows:

The partial BALANCE problem.
Input: The view key pair (k^v, K^v).
Output: The set T of transactions in the blockchain that are directed to OTAs derived from K^v.

[1]Because of how Monero –and most cryptocurrencies– work, every outgoing transaction has as input one or more incoming transactions, as every piece of information in the Blockchain is a transaction. Even coins obtained from mining are portrayed in the ledger as coinbase transactions directed to the user. This way in order to obtain the set of outgoing transactions, one must retrieve incoming ones and check their spent status.

Naturally, we want to do this without losing Monero's privacy guarantees. Specifically, we impose the same constraints **G1** through **G4** to our framework, as specified in the first BALANCE problem.

Let us remark that solving this problem on Bitcoin, or any similar cryptocurrency, can be carried out efficiently by means of standard index structures. Indeed, standard cryptocurrencies without heavy cryptographic measurements to aid privacy, index transactions not by means of an OTA K^o (for a key pair (k^v, K^v)), but simply by posting the public key K^v openly. Thus, for example, one can index each transaction by the public key to which it is addressed, and mount a hash-index to retrieve the set of T transactions addressed to a key K^v in time $\mathcal{O}(|T|)$ which is significantly faster than scanning the full blockchain.

For Monero, the problem is more complex, as each transaction is addressed to an OTA which can only be derived to K^v by means of cryptographic operations. All of this means that it is no longer possible to index Monero's blockchain in a standard way, and currently the only way of solving partial BALANCE problem is by a full scan on the blockchain.

Throughout this chapter we build an indexing scheme that in its most refined version allows one user to obtain the desired set of transactions in sublinear time while only leaking the amount of incoming transactions but not leaking any information about those transactions in particular.

We also show how to cast this problem to the searchable encryption domain has abundant oingoing research and in which sublinear search has been discussed but it is still an open problem[2]. We adapt the partial BALANCE problem to the notion of a PEKS scheme (Boneh, Di Crescenzo, Ostrovsky, & Persiano, 2004) that allows searching over encrypted data in a public key scenario and we show the implications of this proposal.

[2]There is a sublinear search scheme in a deterministic scenario (Bellare, Boldyreva, & O'Neill, 2007) not suitable for our problem but, to the best of our knowledge, the possibility for a scheme that allows sublinear search in a non-deterministic scenario still remains.

4.1. Indexing transaction hashes

The most straightforward way of hasting the balance calculation by solving the partial BALANCE problem is to build a simple hash index that relates public keys of users with all their transactions. Clearly this index is enough for our time requirements, but as expected, it presents a number of security issues. Nonetheless, we go over this solution as it helps us build the discussion on the time and security trade-offs of the solutions in this chapter.

The most basic index consists of a hash table, in which keys are the public view keys of any user, and the values or contents of that entry in the table, are the hashes of the transactions[3] associated to those view keys, stored as an array.

Considering a standard hash table with its original operations `insert(key, value)` and `lookup(key)`, we characterize the table described above with the same operations:

- `insert(public_view_key, transaction_hash)`: value insertion of the transaction hash in the entry array with the public view key as the entry key.
- `lookup(public_view_key)`: key lookup with only the view key.

This a straight forward approach that yields a dictionary-like hash table that can be used to access the list of one user's transactions hashes in $\mathcal{O}(1)$, given its public view key. This is enough for the partial BALANCE problem, as all we need to do is then iterating over this list of transactions, which is clearly linear in the total number of transactions ($|T|$) with OTAs directed to the user.

4.1.1. Issues, attacks and security concerns

It is clear that this approach does not comply with the security requirements needed in order to consider this a viable solution to the partial BALANCE problem. Specifically we

[3] In the Monero blockchain, transactions are handled as hashes of the actual content of the transaction. In turn, each transaction has the One Time Addresses of the destinations attached as keys available in the *vout* field. We sometimes refer to the OTA as the whole transaction, even when a transaction can be directed to multiple OTAs, because it is the output directed to the user what matters for the BALANCE problem.

identify 3 issues and their respective attacks that would compromise privacy of the users and the guarantees outlined in Chapter 1:

I1 Any observer could get every incoming transaction for any given user (**I1**), as this hash table would serve as a direct association between transaction hashes and public view keys. This goes directly against the purpose of one time addresses, revealing the public keys from which OTAs were derived. This yields the first attack, the **direct access attack**, in which one observer can directly use any user's public view key and execute the operation `lookup(public_view_key)` to immediately gain access to all his incoming transaction hashes.

I2 It follows from the above that an observer, besides associating transactions with public view keys, can know the amount of transactions directed to a given user, information that was previously unknown to him. Therefore the second attack appears, the **transactions count attack**, in which an observer can use `lookup(public_view_key)` and count the amount of transactions returned. This attack seems irrelevant when **I1** is still an issue, but with some improvements it becomes clear that this attack can still be performed even when **I1** no longer applies.

This two issues forbid this approach from complying with **G2** and **G3**, as unauthorized third parties can obtain the incoming transactions set and accordingly, features of it. Additionally the last issue appears:

I3 The update of the hash table produces another issue. If the insertion of entries to the hash table is not a validated procedure, anyone –not only the sender– could add values to any entry in the table (**I3**) and pollute the information, thus making the table useless. This yields the **polluted entry attack**, in which an observer can execute `insert(public_view_key, x)` with `x` any random transaction hash, effectively inserting unrelated information to the hash table entry.

Nonetheless, this approach does not compromise the funds integrity nor the ability to spend them, which remains only possible with the possession of the user's spend keys. This way, **G4**, the guarantee about not compromising Monero features unrelated to the balance calculation, is still met. Also, as this approach only modifies elements related to the incoming transactions and not outgoing ones, it does not compromise the balance as a whole, complying with **G1**.

4.2. Indexing encrypted transaction hashes

One straightforward improvement from the direct indexing proposal, is to store the hashes of the transactions with encryption so that only the receiver can effectively discover them.

This indexing procedure entails the same structure as before, with added encryption. This way, the encryption is done with the public view key, so that the user can decrypt the OTA with the corresponding private view key.

Considering the same standard hash table with its original operations specified before, `insert(key, value)` and `lookup(key)`, we characterize the table described above with the same operations:

- `insert(public_view_key, encrypted_transaction_hash)`: value insertion of the encrypted transaction hash to the entry array with the public view key as the entry key.
- `lookup(public_view_key)`: key lookup with only the view key.

Therefore, the sender needs to encrypt the transaction hash with a deterministic public key encryption algorithm[4] before posting it to the hash table. A user wanting to check his

[4]For this subsection we assume the algorithm used is deterministic and we discuss the drawbacks this decision brings. Deterministic encryption in the public key scenario is discouraged as it can not comply with strict notions of security. However, some deterministic schemes have been proposed specially for domains in which the content entropy is considerable (Bellare et al., 2007; Brakerski & Segev, 2011).

incoming transactions, would obtain the contents from the hash table, and decrypt each value in order to obtain each transaction hash.

4.2.1. Improvements

The **I1** issue is addressed with encryption, as an observer would need to decrypt the values in order to associate addresses to the public view key. This means that, in order to get transactions directed to a user, the private view key is needed, which is the same requirement needed in Monero without the introduction of this hash table or any proposal.

In terms of time complexity, the partial BALANCE problem is still solvable in linear time in the total number T of transactions directed to the user. Analogous to what we described in the direct indexing approach, the access time for the entry is $\mathcal{O}(1)$ and to iterate over the pointed list is $\mathcal{O}(|T|)$. However, the encrypted approach adds the need of an extra step for each transaction. This does not penalize the time complexity of the solution as it adds one constant step per transaction of the already retrieved transaction list.

4.2.2. Remaining issues, attacks and security concerns

Even though the **I1** issue is addressed with encryption, it is not solved completely. The **direct access attack** is no longer possible, as one observer can obtain the list of Alice's encrypted transaction hashes, but can not decrypt them, however:

I4 The array of encrypted transaction hashes is still available for any observer, and the method used for the encryption can be replicated and observed by attackers. Therefore, there are two new attacks that enable an unwanted observer to associate transactions to a public view key:

(a) **Re-encryption attack:** If one attacker encrypts every transaction hash available in the blockchain with Alice's public view key, he can then compare the encrypted values with the encrypted transaction hashes found in the table. The encrypted values would match as the encryption is made in the same way as the sender did it originally[5].

An attacker can execute `encrypt(transaction_hash, public_view_key)` with every transaction hash in the blockchain, or any subset of candidates he deems probable, to then compare those encrypted results with those found by executing `lookup(public_view_key)` with the user key.

(b) **Time correlation attack:** If one attacker watches the blockchain, he can correlate the time in which a transaction is timestamped, with the time one value is added to a specific entry in the hash table, therefore being able to guess the owner of those transaction hashes by reducing the possible users from the complete pool of users, to the ones involved in transactions in the same time frame.

An attacker would repeatedly execute `lookup(public_view_key)` as he observes new block additions to the blockchain, in order to try and guess which transactions were posted to the blockchain at the same time as the array in the entry grew.

This means this approach solves **I1** but by doing so introduces issue **I4** which means the guarantees **G2** and **G3** are still not met. Additionally, the aforementioned **I2** and **I3** issues that appeared with the direct indexing proposal, both still remain, as the amount of transactions keeps being publicly obtainable and entries can be polluted.

4.3. Indexing unforgeable encrypted transaction hashes

In order to avoid the **re-encryption attack** presented before, the ability for one outsider to re-encrypt any transaction hash with a user's public view key and compare with the ones available in the hash table entry, has to be disallowed.

[5]This happens when using deterministic encryption which is not a recommended procedure and we manage this issue throughout the next subsection.

To do that, the encryption can be done using non-deterministic techniques, or using elements that are not publicly available i.e. a part of the addresses that is only known by the sender and receiver. We describe both approaches.

4.3.1. Using non deterministic encryption

The most widely used public key encryption algorithms are non-deterministic as they introduce some kind of randomness into the cipher text. To avoid the **re-encryption attack** we need that the encryption algorithm does not yield identical ciphertexts for two identical inputs. Any ELGAMAL-like asymmetric encryption system suited for elliptic curves complies with this requirement.

Specifically, using a non-deterministic encryption system implies including additional information to the entries in the hash table[6], namely the parts of the algorithm needed for the receiving user to reconstruct the plain decrypted message. We call the *encrypted transaction hash cipher* ETHC.

Once again, the same standard hash table operations `insert(key, value)` and `lookup(key)` are used to characterize the table for this proposal:

- `insert(public_view_key, ETHC)`: value insertion of the encrypted transaction hash cipher (including scheme-specific extra cipher elements required) with the public view key as the entry key.
- `lookup(public_view_key)`: key lookup with only the view key.

Accordingly, the sender encrypts the transaction hash with Alice's public view key using a non-deterministic public key encryption algorithm, to then add it to the hash table entry denoted by the same key. In order to check her incoming transactions, Alice

[6]In ELGAMAL, the ciphertext is a pair (c_1, c_2) and in more complex non-malleable encryption systems like CRAMER-SHOUP, the ciphertext is the tuple (u_1, u_2, e, v) (ElGamal, 1985; Cramer & Shoup, 1998). In both cases, every element is needed to reconstruct the original plain text and they play a role in making the scheme secure.

would obtain all her ETHCs from the contents of the entry, decrypt them and obtain each transaction hash.

4.3.2. Using a shared secret

As explained previously, one OTA K^o is built by the sender using a random number r, the public key from the receiver, K^v , the curve generator point G and the public spend key from the receiver, K^s as follows:

$$K^o = \mathcal{H}_n(rK^v)G + K^s$$

We stated that this address can be constructed only by the sender or reconstructed by the recipient because knowledge of rK^v is required for it. Both sender and receiver involved in the transaction can unequivocally derive the OTA.

Our proposal uses rK^v, which we call the **shared secret**, as the central piece to add encrypted to the hash table entry. Once again, the same standard hash table operations `insert(key, value)` and `lookup(key)` are used to characterize the table for this proposal:

- `insert(public_view_key, encrypted_shared_secret)`: value insertion of the encrypted shared secret with the public view key as the entry key.
- `lookup(public_view_key)`: key lookup with only the view key.

Accordingly, the sender encrypts the shared secret rK^v with Alice's public view key using a public key encryption algorithm[7], to then add it to the hash table entry denoted by the same key. In order to check her incoming transactions, Alice would therefore obtain all her rK^vs from the entry, decrypt them, use her public spend key and standard operations in order to obtain each OTA representing her transactions.

[7]This algorithm can be both deterministic and non deterministic, as there is minimal risk of a chosen plain text attack because the content itself is seemingly random, therefore common attacks as dictionary ones are useless.

The Monero blockchain and its CLI implementations provide ways to obtain transactions from their hash directly but there is no current implementation to retrieve a transaction by the One Time Addresses assigned as outputs of it. However, auxiliary data structures could be publicly created in order to be able to access to a transaction hash given its OTA.

4.3.3. Improvements

The first attack that appeared in the normal encryption approach, the **re-encryption attack**, is handled by this approach. Now, an observer can not try and encrypt blockchain values as he could before, in order to check whether the encrypted values match with the values stored in the entry. This happens because (a) non-deterministic encryption systems yield different ouputs for the same inputs, or (b) he can not obtain the plain values to try with, because the rK^vs are not published in any part of the transaction, so he can not access the **shared secret**.

Time complexity remains the same as in the other approach, analogously, the partial BALANCE problem can still be solved in linear time in the total number $|T|$ of transactions directed to the user. The access time remains $\mathcal{O}(1)$ and to iterate over the list is $\mathcal{O}(|T|)$. The extra time spent for decryption depends on the the encryption system and the transactions themselves, but not on the amount of transactions, keeping the algorithm time linear.

4.3.4. Remaining Issues, attacks and security concerns

As explained, the **re-encryption attack** is addressed and solved by this approach meaning that for **I4**, only the second attack remains. The **time correlation attack** can still be done, however, correlation attacks can be performed to the original Monero protocol at the present time, and the guess work needed is still very ineffective[8].

[8] In (Tramèr et al., 2020) an attack to reveal transaction payees by analyzing traffic over side-channels was presented. It has subsequently been solved, but the possibility of those attacks can not be ruled out.

This way the only important issues remaining are **I2** and **I3** with their respective attacks.

4.4. Indexing recapitulation

So far, we have constructed a technique allowing senders to include transaction information to a publicly available hash table. This enables users to retrieve the collection of transaction hashes directed to them in linear time in respect to the amount of incoming transactions.

In the first approach, **indexing transaction hashes**, we proposed to store transactions hashes directly using the view key of the user as the key. This meant giving away the link between one user and transactions, compromising privacy heavily, but delivering fast access to transactions for a user.

For the second approach **indexing encrypted transaction hashes**, we introduced encryption providing a layer of security, but as we discussed, it was still vulnerable to simple guessing attacks. This meant that even though the link between a view key and a group of transactions was not immediate, it did not guarantee privacy.

However, in our last approach, **indexing unforgeable encrypted transaction hashes**, we explained how to properly use encryption in order to avoid the attacks that made the previous approach useless. In this last proposal, users would only compromise the amount of transactions directed to them and the possibility for unwanted users to add fake transaction hashes to the entries, but would not reveal their identity, the amounts of each transactions, the spendable state of them nor compromise their ability to spend them.

All 3 approaches had a major issue related to the practicality of them, by relying on senders to update the index entries related to a user, entries could be filled with noise and the real purpose of the indexes would have been lost.

In a later chapter of this work we propose services based on this proposals that address the pollution issue **I3** putting minimum trust in them and not majorly compromising privacy. As in all of our work, we touch on the trade-off between giving away different types of information and being able to retrieve information fast.

4.5. Searchable encryption

One of the issues that remains unsolved using the indexing approach –even in the encrypted version– is the capability of unwanted insertion to the hash table entries. An application of a searchable encryption scheme provides a way around it and gives a different framework to tackle the partial BALANCE problem.

Searchable encryption (SE), is an encryption scheme that supports searching keywords over encrypted data without decrypting it, hence, not compromising privacy (Bösch, Hartel, Jonker, & Peter, 2014). The two most investigated SE techniques are Searchable Symmetric Encryption (SSE) and Public Key Encryption with keyword search (PEKS). For our work, the PEKS one is the evident choice.

Public Key Encryption with keyword Search (PEKS) was firstly introduced in (Boneh et al., 2004) and it provides a mechanism that enables Alice to receive data in an outsourced server. Data can be sent from various other users encrypted with her public key and can afterwards be searched using trapdoors generated with her private key.

Various aspects of the PEKS model have been studied by the community. In particular, focus has mainly revolved around reducing keyword leakage[9] (Arriaga, Tang, & Ryan, 2014; Nishioka, 2012), allowing flexibility in the search through multi-keyword, fuzzy search, wildcards, subset queries, range queries, etc. (Boneh & Waters, 2007; Hwang &

[9]The original PEKS guarantees that a searchable ciphertext leaks no information about keywords, but it does not give a guarantee concerning leakage of a keyword from the trapdoor (Nishioka, 2012).

Lee, 2007), and reducing search time[10] (Phan, Dang, & Nguyen, 2014; Chen, Zhang, Lin, & Zhang, 2016; Zhang, Li, & Wang, 2019; Bellare et al., 2007).

4.5.1. PEKS scheme for Monero

To apply PEKS to the partial BALANCE problem we propose to use keys and transaction hashes as keywords and encrypted data respectively. Specifically, we propose to use the PEKS scheme where each document contains a transaction hash and the receiver public view key, encrypted[11]. The searchable keyword is the public view key and it would only be accessible by those in possession of a trapdoor derived from the private key.

As outlined in (Boneh et al., 2004), the non-interactive PEKS scheme consists of the following polynomial time randomized algorithms:

(i) `KeyGen(`s`)`: Takes a security parameter, s, and generates a public/private key pair A_{pub}, A_{priv}.

(ii) `PEKS(`A_{pub}, W`)`: for a public key A_{pub} and a word W, produces a searchable encryption of W.

(iii) `Trapdoor(`A_{priv}, W`)`: given Alice's private key and a word W produces a trapdoor T_W.

(iv) `Test(`A_{pub}, S, T_W`)`: given Alice's public key, a searchable encryption $S =$ `PEKS(`A_{pub}, W'`)`, and a trapdoor $T_W =$ `Trapdoor(`A_{priv}, W`)`, outputs 'yes' if $W = W'$ and 'no' otherwise.

[10]Only one of the studied PEKS schemes achieves sublinear search time, the one presented in (Bellare et al., 2007) but it does so by using deterministic encryption over the keywords, which is not suitable for our solution. However, investigation in this field keeps appearing and the focus is still dedicated to improve that particular issue.

[11]The PEKS scheme states that the document must be encrypted non-deterministically with the public key, therefore for the partial BALANCE problem, the pair (tx_id, K^v), can be employed as the document. As it is non-deterministic, including the public view key in the document itself does not introduce the risk of re-encryption attacks.

Therefore Alice runs the `KeyGen` algorithm to generate her public/private key pair[12]. She uses `Trapdoor` to generate trapdoors T_W for any keywords W that she wants to enable search for. The server uses the given trapdoors as input to the `Test` algorithm to determine whether a given document contains one of the keywords W specified by Alice.

For our problem it can be applied using each of the four algorithms with the correct arguments as follows:

(i) `KeyGen(`s`)`: Takes a security parameter, s, and generates a public/private key pair A_{pub}, A_{priv}.

(ii) `PEKS(`A_{pub}, K^v`)`: for a public key A_{pub} and the keyword K^v, produces a searchable encryption of K^v.

(iii) `Trapdoor(`A_{priv}, K^v`)`: given Alice's private key and the public view key acting as keyword, produces a trapdoor T_W.

(iv) `Test(`A_{pub}, S, T_W`)`: given Alice's public view key, a searchable encryption $S =$ `PEKS(`A_{pub}, K'^v`)`, and a trapdoor $T_W =$ `Trapdoor(`A_{priv}, K^v`)`, outputs 'yes' if $K^v = K'^v$ and 'no' otherwise.

Notably, the PEKS algorithm is called with K^v as a keyword. This allows searchability using that view key, which is semantically concordant with the purpose of view keys in the Monero protocol.

With PEKS in place, Alice can produce a trapdoor that enables searching for transactions associated to her public key K^v, by using `Trapdoor(`A_{priv}, K^v`)` and sending it

[12]The user would need to hold a new pair of keys besides the spend and view key pairs already existent. This is due to the need of a suitable key pair for the encryption and posterior search. Boneh's et al. proposal (Boneh et al., 2004) includes the usage of bilinear maps, and the chosen keys must allow it efficiently. Due to its characteristics, the Monero base elliptic curve Ed25519 is not suitable for use in bilinear maps as its degree is too high (Lynn, 2007; Stackexchange, 2017a; Bernstein & Lange, 2013) and computation of the mappings would be infeasible.

to the server. The server would then test every document against the trapdoor in order to know if the document must be returned[13].

4.5.2. PEKS procedure in Monero

With the outlined algorithms and the usage of Monero pieces, we can further describe the procedure for a normal transaction to include the information required, the responsibilities and the way for Alice to retrieve her transactions.

(i) Sender:
   (a) Create a Monero transaction (denoted by the hash tx_hash) normally.
   (b) Encrypt the document consisting of the pair $d = (tx\_hash, K^v)$ obtaining $Enc_{A_{pub}}(d)$.
   (c) Generate the searchable keyword with `PEKS(`A_{pub}, K^v`)`.
   (d) Include both elements in the transaction metadata[14].

(ii) Miner:
   (a) Post the encrypted document and searchable keyword to the document collection[15].

(iii) Receiver:
   (a) Generate a trapdoor T_W with `Trapdoor(`A_{priv}, K^v`)`.
   (b) Query the collection with the generated T_W.

We have found that the best way to apply such an scheme to the partial BALANCE problem is to use a third party service as a transaction yielder that serves the encrypted

[13] This yields a time complexity of $\mathcal{O}(n)$ with n the amount of documents, which is not sublinear. We believe sublinear search is not possible in this scheme, however, if discovered and applied to a PEKS scheme without deterministic encryption for keywords, it could be of use for our scheme.

[14] In Monero it is possible to include additional information for each transaction in the tx_extra field (Alonso et al., 2020). However, by doing so, the amount of information included in every transaction increases and the blockchain size also increases. We do not discuss this problem throughout our work, but it is a relevant issue as keeping the blockchain size as low as possible is desirable. We believe that it is an acceptable tradeoff, but we encourage further analysis.

[15] The location, behaviour and relevance of this collection is discussed in the last chapter as a third party service proposal.

collection of transaction identifiers and that is only searchable by those in possession of adequate keys. We describe the details of this implementation in the last chapter.

4.5.3. Improvements

The proposed procedure addresses most of the issues outlined while building the indexing proposal. **I2** is no longer a problem as the public view key of one user is not enough to obtain the amount of transactions, with a PEKS scheme like the one introduced, an attacker needs A_{priv}, which is not revealed in any moment.

Most notably, **I3** is also solved with this procedure. As the entity responsible of adding new entries to the document collection is the miner, there is no risk for attackers to introduce unwanted information. Moreover, this procedure can be checked by the entity holding the collection as the document inserted in the collection is available in the blockchain.

The last remaining issue (**I4**) is also handled with this approach. While any user attempting to discover whether a transaction is directed to Alice can encrypt a transaction hash with her A_{pub}, as the encryption is non-deterministic, the encrypted element included in the transaction metadata can not be feasibly reproduced.

4.5.4. Remaining issues, attacks and security concerns

The first issue we observe when using the proposed PEKS scheme is that, as outlined before, the search is not sublinear in every non-deterministic implementation yet (Bösch et al., 2014; Wang, Wang, & Chen, 2016). We believe sublinear search is not possible in this scheme, however, if discovered and applied to a PEKS scheme without deterministic encryption for keywords, it could be adapted for our scheme[16].

Secondly, the trapdoor generation proposed in (Boneh et al., 2004), is deterministic. This enables the server being queried for the documents to store trapdoors for further use.

[16] Depending on the implementation, a sublinear scheme may or may not be adapted successfully to our problem, as we require more conditions than only time related ones.

Nonetheless, the only information the server gains is that for a given trapdoor there are some associated documents, but it does not learn the public key employed as keyword nor the transaction hash stored as content of the document. This implies that trapdoor privacy is not completely achieved with this scheme[17].

However, the PEKS procedure to append keywords to a document is not deterministic, therefore the server is not able to guess the keyword content from any documents with the same keyword, as all encrypted keyword material for the same keyword is different. This means that even if the server holds a trapdoor, it is not able to associate it successfully to a particular public key.

[17] There is investigation covering trapdoor privacy in PEKS schemes. In (Nishioka, 2012) and (Arriaga et al., 2014), solutions and improvements are proposed, however implementations for our problem are not straightforward.

5. PRIVACY PRESERVING OUTSOURCED SERVICES

Although we have not provided solutions that allow sublinear time and complete security, in this chapter we show how our solutions open up possibilities for what wallets can do by leveraging third-parties in order to enable faster functioning times. We do so while giving users the ability to compromise less security than giving away the private view key as it happens currently.

While we provided an implementation that does not require any protocol modifications to Monero in Chapter 3, our indexing and PEKS scheme proposals in Chapter 4, require the introduction of new procedures for senders, receivers and miners. It becomes clear in this chapter the practical usage of those modifications.

We have already approached the privacy and efficiency trade-off by discussing the way transactions are built and retrieved in the protocol. Now we provide two third party services that allow our user Alice to retrieve her transactions by offloading the time intensive processing to the third party while minimally compromising privacy and minor elements that were previously unknown to them.

The last version of our indexing protocol achieved the storage of transaction identifiers unforgeably encrypted in a hash table for Alice to retrieve the identifiers in sublinear time. However, the pollution issue remained unsolved and made futile any straightforward implementation of that proposal. Similarly our proposal of a PEKS scheme implementation still needs a holder for the document collection containing the encrypted association between keys and transactions.

We present a **revocable indexing service** allowing a third party to provide access to one user's transactions without compromising the user's private view key. We also outline how a third party could act as a **trapdoor transaction yielder** for a PEKS scheme, storing the encrypted documents collection.

5.1. Revocable indexing service

A revocable indexing service is a third party that obtains user authorization to store indexed encrypted references to transactions for particular keys. It is revocable because a user can decide to change keys and disallow the service for future indexing without the need of changing view nor spend keys.

Users will need a third pair of keys which we call the index key pair with both a private and a public index key (k^x, K^x). This is a separated pair as it is used to allow the service view access to transactions directed to the user, while not compromising the balance nor the amounts of each transaction, information bound to the view key pair.

This pair works analogously to the view key pair. Senders will need the public index key in order to correctly build a transaction directed to the owner of that key pair. Similarly, a user in possession of the private pair will be able to access to all of the user transactions signaled by those keys but will not be able to decode the amounts.

For the indexing service to work, both the sender and this new third party will have specific responsibilities. The sender will need to provide new information in the transaction for the service to recognize it and add it to the index.

5.1.1. Sender procedure

In order to send funds to Alice, the sender creates a transaction normally. Additionally the sender needs to append a new element to the transaction derived from Alice's index key, the One Time Indexing Key (OTIK):

$$K^z = \mathcal{H}_n(rK^x)G + K^v$$

The sender builds it with the public keys of Alice, and the random r he computed for the construction of the OTA[1]. Similarly, Alice, owner of the private index key, can reconstruct this one time key by using her private key k^x and the public key of the transaction rG as follows:

$$K'^z = \mathcal{H}_n(rk^xG)G + K^v$$

As $rk^xG = rK^x$ stands, then $K^z = K'^z$, confirming it is the same OTIK created by the sender.

Posting the OTIK and rG in the transaction enables any observer to try and verify if the transaction is directed to Alice, but only a party owning the private index key will successfully reconstruct the OTIK and match it.

Alongside the OTIK, for indexing purposes, the sender needs[2] to create the same element we described for Section 4.3. This is, he needs to include the encrypted shared secret or the non-deterministic cipher in the transaction. However, it has to be encrypted once again with the index key[3]. Without loss of generality, we will use the method proposed in Section 4.3.1 featuring ETHCs:

$$index\_metadata = Enc_{K^x}(ETHC)$$

5.1.2. Indexing procedure

The third party will firstly enroll any user wanting the indexing service. For this, the service will acquire Alice's private index key k^x. With this key the service will be able to scan the blockchain in order to identify transactions directed to Alice.

[1]The procedure is the same as the one presented for OTAs. The relevant change is the usage of K^x and K^v instead of K^v and K^s, respectively.
[2]The need of this information alongside the OTIK is important for a more *trustless* implementation of the indexing service, which will be discussed in the following sections.
[3]The shared secret or cipher is encrypted with both the public view key and the public index key to allow the service to decrypt the metadata. This prevents observers of watching the same element available in a transaction to be seen in the index the service provides.

Alongside getting the private index key, the service needs to create one unique key in which the array of identifiers will be stored. This key has to be accessible to Alice and has to appear to be seemingly random. As both Alice and the indexing service know the private index key, we suggest this key to be derived from it and the public key of the indexing service K^j as follows:

$$key = k^x K^j$$

This way, both parties can construct it only if they know the private index key of Alice.

Finally, to add an element to the entry, the service must scan the blockchain and construct OTIKs for every transaction. Whenever a transaction is positively identified as directed to Alice, then it must search in the transaction for the index metadata available there, encrypted as $Enc_{K^x}(ETHC)$. The service, who knows k^x, can decrypt that information in order to obtain the ETHC which is the same *cipher* discussed in Section 4.3.1. This piece of information can be safely added to the hash table entry.

5.1.3. Service usage

Considering Alice is already enrolled to the service and therefore her index keys are known by the indexing service, the steps needed for both sender and indexing service in order to take advantage of it are:

Sender:

(i) Follows the normal transaction building procedure.

(ii) Computes the OTIK as: $K^z = \mathcal{H}_n(rK^x)G + K^v$.

(iii) Computes the indexing information as: $index\_metadata = Enc_{K^x}(ETHC)$.

(iv) Adds K^z and $index\_metadata$ to the transaction data.

(v) Posts the transaction to the blockchain normally.

Indexing service:

(i) Scans the blockchain for new transactions.

(ii) Computes a candidate OTIK for every new transaction using Alice's private index key as $K'^z = \mathcal{H}_n(rk^xG)G + K^v$ using each transaction rG, Alice's k^x and K^v.

(iii) Checks whether the OTIK contained in the transaction matches the one computed $K^z \stackrel{?}{=} K'^z$.

(iv) When a match is found, decrypts the $index\_metadata$ in order to obtain the ETHC.

(v) Computes the key for the entry k^xK^j and adds the ETHC to the entry.

After those steps, Alice is able to access the index[4] and retrieve the content of the entry labeled by k^xK^j. She can then proceed to use the ciphers obtained in order to get her incoming transaction hashes.

5.1.4. Revoking access

One of the features this procedure provides is the ability to revoke access to any third party acting as this indexing service. In order to do so, Alice can generate a new index key pair. This is fundamentally different to generating a new view key pair[5] as the index pair is not used to generate the OTAs to which funds are addressed.

If Alice needs to deny the indexing service of further indexing, and eventually move to another service provider, she could churn her funds into one transaction using her new index key pair. This makes transactions indexed by the denied service no longer usable.[6]

[4]The hash table itself could be publicly available as nor the keys nor the content reveal information about the user. The key is seemingly random and the content is encrypted.

[5]Currently in Monero Alice can generate subaddresses derived from her main address and direct funds differentiating between them (Alonso et al., 2020). However, in order to allow someone to retrieve transactions for her, she needs to give away private information about those addresses. If she wants to keep the subaddress and disallow anyone in posession of the private view key, she cannot do that.

[6]The indexing service as described so far adds to every user entry an encrypted transaction hash which cannot be read by the service. However, as the service is scanning the blockchain, it could maliciously store the transaction hashes that matched the OTIKs elsewhere. This makes the trust in the third party crucial. We tackle this issue in the next section.

5.1.5. Trustless implementation

We have discussed so far an implementation that inserts an encrypted transaction hash to the hash table entry assigned to a particular user. The encryption allows the element to be safely posted in the blockchain and also allows for the hash table contents to be publicly available. However, it does not prevent the indexing service to know exactly which transactions belong to the user.

To make the indexing service more trustless, in practice, senders can incorporate decoys[7] additional to the real OTIK when including it for each transaction. This way, the service must check every OTIK in the transaction to add the corresponding metadata to the entry, but only if any of the OTIKs matches the one generated from the enrolled user index key.

The decoys work by adding valid OTIKs for other users. This part of the procedure must be enforced widely, to ensure Alice appears in other transactions as a decoy. This way, the entry for Alice will contain encrypted transaction hashes that are effectively directed to her and ones that are not[8]. In terms complexity, if the amount of decoys is fixed[9] and the algorithm employed to select decoys to obfuscate the real OTIK is adequately uniform and does not favor or punish specific index keys, this will only increase the time required to obtain the real transaction hashes by a constant[10].

By introducing decoys, the service will now only know that the transactions that matched the generated OTIKs are possibly owned by the enrolled user, but will not have certainty, requiring less trust from the user.

[7]We do not provide a thorough investigation in the way this decoys should be selected. However, we believe using a procedure similar to the one employed to select output decoys for ring signatures should be sufficient.
[8]Extra transaction hashes are added to the entry because valid OTIKs appeared as decoys in other transactions. The indexing service will have found a matching OTIK and added the ETHC to the entry, without being able to identify whether it was a decoy or not.
[9]We propose a similar number to Monero's current ring size, which is 11 (Spagni, 2018).
[10]This is not a completely accurate description of time complexity for this method. However, based on a uniform algorithm to select index keys as decoys, the chances for a specific user of appearing in transactions as a decoy, is roughly proportional to the amount of decoys.

5.1.6. Improvements

The revocable indexing service tackles most of the issues still present in our last indexing proposal in Section 4.3. Firstly one observer is no longer able to perform the **transactions count attack** as the key to access the entry holding the identifiers for transactions is only obtainable by those in possession of the private index key. Even if the hash table is public, the association between one user view key and the entry is not public.

Secondly as this index is managed by a third party that provides a service, there is no risk of unwanted **entry pollution**. However, this means that by trusting a third party, the integrity of the information retrieved relies in the service.

Thirdly, the **time correlation attack** is mostly addressed with the usage of the revocable indexing service. While an observer can still watch the blockchain and see the index metadata, this metadata is not the same that gets posted into the entry in the index, because it is firstly encrypted by the sender with the index key and afterwards it gets decrypted by the indexing service. This implies an observer could only try and deduce whether something new was added to the index, but could not associate this addition to a particular user or group of them[11].

As explained earlier, the indexing service is revocable therefore Alice can decide to stop using the indexing service without the need to renew her view or spend keys, and the service will not be able to index new transactions for Alice. This allows to reduce the trust needed towards the third party as the indexation can be done temporarily. Furthermore if this approach is combined with churning, all previous indexed transactions can be deemed as useless information, because the new ones with the real funds would be indexed under another key.

[11] Without the index key encryption, an attacker could base his deduction on the fact that *"a transaction in this new block is addressed to a view key as the entry for that key had one transaction added in this timeframe"*. Whereas with the index key encryption, the fact becomes *"a transaction in this new block is addressed to one user of the indexing service as one entry for that key had one transaction added in the timeframe"* which is not useful enough for the attacker.

In terms of time complexity the data structure used behind the indexing service is the same as in the previous sections. The operation needed to be performed when a user retrieves his list of identifiers is still a hash table entry lookup that takes linear time in the number T of transactions. If we take into consideration the trustless decoys approach, the complexity does not worsen but gains a constant factor.

This implementation touches the privacy and efficiency tradeoff we have discussed throughout our work in a moderately compromising way. By giving away a private index key to a third party, in this case the service, it only gains the ability to inspect the blockchain for transactions directed to the user, but it does not disclose the amount sent in each transaction, as giving away the private view key does. Furthermore, in our trustless implementation, owning the private index key only enables the holder to associate a group of transactions without specificity, because it contains the real group of transactions alongside decoys, meaning the third party is no longer trusted fully.

Thus finally, while this approach implies willingly compromising the identifiers of incoming transactions for Alice, it enables her to access transactions quickly without performing a full scan of the blockchain and without having to compromise the amount received. This can be seen as further granularizing the permissions scheme in Monero and giving the user even more control over funds.

5.1.7. Remaining issues and security concerns

We have already addressed all the important issues that appeared in our theoretical discussion of indexing implementations and discussed how the revocable indexing service approached them at least partially and provided a practical implementation. However there are two new considerations related to this proposal that make further improvement possible.

In the first place, when a sender is building a transaction directed to Alice, he can remember the encrypted transaction hash cipher before the index key encryption. This piece of information enables him to observe where that piece of information appears in the indexing service if it is publicly available. By doing so, he can deduce the key of the entry where transactions directed to Alice are stored. In previous implementations this was not an issue as the key was directly the user view key, but now it is meant to be private. This can be solved by the service performing a re-encryption of the ETHC.

In the second place, with our trustless approach, we introduced mixing by adding one time indexing key decoys to the sender procedure. With the decoys used for the ring signatures in Monero, care must be taken in order to avoid disclosing the real inputs, in our case, we do not currently know if a particular algorithm for selecting decoys may affect the obfuscating factor of using them. However, we believe that picking random ones uniformly from any user does not imply a security risk, as they are not transactions but index keys[12], nevertheless, we encourage further investigation to confirm our hypothesis.

5.2. Trapdoor transaction yielder

A trapdoor transaction yielder is a third party that can store a collection of encrypted documents containing the association between transaction hashes and user view keys. It enables a user to generate a trapdoor from its view key and query the service for it to yield the set of documents corresponding to the key.

In Section 4.5 we discussed the requirements and implications of leveraging a PEKS scheme to tackle the partial BALANCE problem. However, in the current section, we propose using it as a third party that yields transactions to any user regardless of sublinear search requirements not being met.

Even with linear search times, the application of a PEKS scheme for our problem is still useful. A third party can store the encrypted document collection and allow queries over it to those in possession of trapdoors. This securely outsources the procedure and

[12]For ring signatures, outputs from the blockchain are used as decoys, this means that outputs could have been spent, could be old, could be stale, etc. In our case decoys are formed by selecting index keys, which are a user related piece of information, that does not depend on age nor spend status.

moves the responsibility of the load-and-time-intensive scan from the user to a third party that can optimize its resources for the task.

The compromise that a user makes in order to outsource the retrieval is giving away a trapdoor, however, the trapdoor does not reveal information about the public view key nor the transactions associated to it, therefore we consider the trade-off reasonable and comparable to previous sections.

5.2.1. Procedure

Both the scheme and the procedure outlined in 4.5 remain the same for senders and receivers. Namely, the sender must (a) create a transaction, (b) encrypt the PEKS document, (c) generate the searchable keyword and (d) include the elements in the metadata, whereas the receiver must (a) generate a trapdoor and (b) query the collection.

This way, our user Alice needs a new pair of keys suitable for the scheme, making the public key available. Senders following the procedure will generate transactions with the metadata for Alice and then Alice will generate trapdoors in order to query the third party, that is populated with the metadata or document collection.

The entity that holds the document collection acts as the third party and in the previous chapter the collection update responsibility relied on the miner, however, it is optional. In a competing environment, multiple third parties could act as transaction yielders for this scheme, therefore the are new requirements for miners.

If the miner needs to post new information to the third party transaction yielder, users would need to signal in the transactions which third parties are to be notified[13]. However, a more robust scheme dispenses the need of miners involvement altogether and allows for the third party to search for transaction metadata in already mined transactions. This

[13]This implies including even more information in the metadata, which is not encouraged as it increases transaction sizes which is mainly avoided by the community. Also this could leak innecessary information to both the transaction yielder and the sender.

implies storing it proactively instead of needing miners to signal, allowing a third party to provide the service to any user at any time yet it implies high space requirements[14].

By storing the document collection the third party only retrieves information spread through the blockchain, and does not gain any extra information that was not already available in the blockchain by posting the metadata. Therefore this collection has the security characteristics we discussed in Section 4.5, but those are not affected in any way by the goodwill of the thirdparty[15].

5.2.2. Volatile wallets

One upside of using this approach, as with our indexing service, is the possibility for users to leverage them in order to avoid performing time intensive scans in their devices. Users can rely on those third parties and securely retrieve their transactions allowing them to compute their balance quickly. With normal wallets, there is currently the possibility to use third party nodes and even third party wallets. With the former our user Alice trusts the node to give reliable information and to have secure up-to-date software when being queried for information[16]. In the latter, Alice directly compromises her private view or spend keys, making both approaches inadvisable.

With third party transaction yielders, wallets can communicate with them in order to retrieve an encrypted collection of identifiers only giving away keyword trapdoors. Then, Alice can use the retrieved set in order to obtain the decrypted transactions and compute the balance locally and safely in her device.

[14] Storing proactively every transaction metadata would yield collections with size proportional to the amount of transactions in the blockchain, however, the amount of data stored per transaction is small and dependant of the encryption algorithm for both the document and keywords.

[15] If a user Bob does not use the service but transactions directed to him comply with the PEKS scheme required metadata, he does not compromise information that was not compromised before the application of this protocol modification proposal, even when metadata is stored by the third party.

[16] Bugs related to accessing third party nodes in which the node could obtain information about the user querying have been fixed in the official implementation of the monero wallet (Tramèr, 2019; Moneromooo-monero, 2019; Tramèr et al., 2020), however, by design, remote nodes require trust (Monero-Hax123, 2018; Tramèr et al., 2020).

Wallets implementing such an approach could be created without the need of storing identifiers themselves or processing raw data obtained from nodes[17], and be recreated in various devices, without needing to perform scans in each of them. As with the indexing service, our user Alice can decide whether to use or not the described services, but by doing so she can take advantage of reduced time for balance computation.

[17]When using a remote node, users get the stream of transactions and analyze them in order to check if directed to them. This means that even though the blockchain is not stored in the device, the processing of each transaction happens in it.

6. CONCLUSIONS

In this work, we provided different ways to speed up the balance computation in Monero without compromising crucial privacy elements nor funds integrity. The balance computation in Monero is a time intensive operation as it requires complete blockchain scans due to the obfuscated addresses used to conceal identities. Currently, users have to choose between enduring the process and circumventing it by compromising their privacy which deteriorates the user experience and weakens Monero's main features. Our contribution offers users the ability to obtain shorter balance computation times by introducing consolidation transactions and wallet implemented modifications to shorten the blockchain scan.

We also provided schemes for indexing transactions. Enabling users to access indexes of transactions allows them to compute the balance quicker, as the retrieval of incoming transactions is not linear in the total amount of existing transactions. We analyzed different approaches for indexing and their implications in user privacy. Our most refined indexing scheme achieved transaction retrieval in sublinear time, which enables the user to obtain their balance quickly. Additionally, we provided a PEKS scheme adaptation to the BALANCE problem in Monero, which in turn encourages further investigation in that area. We expect that future investigation about PEKS schemes could allow sublinear transaction search with our adaptation.

Finally, we described the application of the proposed indexing techniques in order to create third parties that can offload the balance computation securely. We outlined an indexing service that could serve transactions to any user enrolled, introducing indexing keys that gave users the ability to revoke access if needed. This allowed wallets to rely on third parties without compromising their private view keys in order to retrieve balance faster. This, in turn, enables the user to have volatile wallets with less setup and less trust.

We look forward to further study the implications of the decisions taken for building the third party indexing services, such as the trustless implementation and the use of decoys. While there is ample evidence of how decoys perform in the obfuscation of senders in Monero, we would like to investigate them in the context of indexing transactions. We would also like to investigate the theoretical limits in the complexity of PEKS schemes. Finding a sublinear PEKS scheme and applying it to Monero would open the possibilities for implementations like ours with even faster computation times.

REFERENCES

Aki, J. (2018, October 22). Monero transaction fees reduced by 97% after bulletproofs upgrade. *Bitcoin Magazine*. https://bitcoinmagazine.com/culture/monero-transaction-fees-reduced-97-after-bulletproofs-upgrade. (Accessed: 2021-05-30)

Alex, B., & Herrera, R. (2020, October). Private communication. (MyChurnero creator email discussing issues surrounding Monero balance computation, churning and usability)

Alonso, K. M., et al. (2020, April). *Zero to monero: Second edition.* https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf.

Arriaga, A., Tang, Q., & Ryan, P. (2014). Trapdoor privacy in asymmetric searchable encryption schemes. In *International conference on cryptology in africa* (pp. 31–50).

Bellare, M., Boldyreva, A., & O'Neill, A. (2007). Deterministic and efficiently searchable encryption. In *Annual international cryptology conference* (pp. 535–552).

Bernstein, D. J., & Lange, T. (2013). *Safecurves: choosing safe curves for elliptic-curve cryptography.* https://safecurves.cr.yp.to/transfer.html. (Accessed: 2021-02-27)

BitInfoCharts. (2021, June). *Monero avg. transaction fee historical chart.* https://bitinfocharts.com/comparison/monero-transactionfees.html. (Accessed: 2021-05-30)

Boneh, D., Di Crescenzo, G., Ostrovsky, R., & Persiano, G. (2004). Public key encryption with keyword search. In *International conference on the theory and applications of cryptographic techniques* (pp. 506–522).

Boneh, D., & Waters, B. (2007). Conjunctive, subset, and range queries on encrypted data. In *Theory of cryptography conference* (pp. 535–554).

Bösch, C., Hartel, P., Jonker, W., & Peter, A. (2014). A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, *47*(2), 1–51.

Brakerski, Z., & Segev, G. (2011). Better security for deterministic public-key encryption: The auxiliary-input setting. In *Annual cryptology conference* (pp. 543–560).

Chen, Y., Zhang, J., Lin, D., & Zhang, Z. (2016). Generic constructions of integrated pke and peks. *Designs, Codes and Cryptography*, *78*(2), 493–526.

Cramer, R., & Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Annual international cryptology conference* (pp. 13–25).

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, *31*(4), 469–472.

Hwang, Y. H., & Lee, P. J. (2007). Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *International conference on pairing-based cryptography* (pp. 2–22).

Hyc. (2018, December). *Monero pull request #4981: Add –restore-date param.* `https://github.com/monero-project/monero/pull/4981`. (Monero repository contribution to allow date as an input to limit scan height)

Lynn, B. (2007). *On the implementation of pairing-based cryptosystems* (Unpublished doctoral dissertation). Stanford University Stanford, California.

Macbook air (13-inch, early 2015) - technical specifications. (2021, April). `https://support.apple.com/kb/SP714`. (Accessed: 2021-05-29)

Monero-Hax123. (2018, March). *Corrupt rpc responses from remote daemon nodes can lead to transaction tracing.* https://hackerone.com/reports/304770. (Accessed: 2021-05-30)

Moneromooo-monero. (2019, November). *Monero pull request #6074: Fix info leak when using a remote daemon.* https://github.com/monero-project/monero/pull/6074. (Monero repository contribution to allow fix remote daemon leak)

Moneropedia: Fungibility. (n.d.). https://www.getmonero.org/resources/moneropedia/fungibility.html. (Accessed: 2021-03-01)

Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., ... others (2017). An empirical analysis of traceability in the monero blockchain. *arXiv preprint arXiv:1704.04299*.

Nishioka, M. (2012). Perfect keyword privacy in peks systems. In *International conference on provable security* (pp. 175–192).

O'Leary, R. (2018, October 22). Monero fees fall to almost zero after 'bulletproofs' upgrade. *Coindesk*. https://www.coindesk.com/monero-fees-fall-to-almost-zero-after-bulletproofs-upgrade. (Accessed: 2021-05-30)

Phan, T. Q., Dang, V. H., & Nguyen, T. D. (2014). A novel construction for peks scheme using matrix group. In *Ubiquitous information technologies and applications* (pp. 335–343). Springer.

Reddit, U. (2018a, November). *Churn 1000+ monero.* https://www.reddit.com/r/Monero/comments/9xzvq6/churn_1000_monero/e9x61f4. (Monero Core Team developer comment in Reddit post)

Reddit, U. (2018b, November). *Churn 1000+ monero.* https://www.reddit.com/r/Monero/comments/9xzvq6/churn_1000_monero/e9x4vyo. (Monero Core Team developer comment in Reddit post)

Reddit, U. (2018c, September). *Please clarify this regarding anonymity/probability.* https://www.reddit.com/r/Monero/comments/9ebtne/please_clarify_this_regarding_anonymityprobability/e5ntie4. (Monero Core Team developer comment in Reddit post)

Reddit, U. (2018d, January). *Running your own node vs using a remote node?* https://www.reddit.com/r/Monero/comments/7no0nh/running_your_own_node_vs_using_a_remote_node/ds4auw7. (Monero Research Lab researcher comment in Reddit post)

Reddit, U. (2018e, July). *Small question about churn.* https://www.reddit.com/r/Monero/comments/8vflqg/small_question_about_churn/e1nkxm1. (Monero contributor comment in Reddit post)

Reddit, U. (2018f, November). *What are the biggest issues with monero and its technology?* https://www.reddit.com/r/Monero/comments/9z3rz6/what_are_the_biggest_issues_with_monero_and_its/ea66ikv. (Monero community user comment in Reddit post)

Reddit, U. (2018g, December). *What's the latest consensus on minimum amount of time between monero churns to oneself and why?* https://www.reddit.com/r/Monero/comments/a6b3ea/whats_the_latest_consensus_on_minimum_amount_of/ebtvqk1. (Monero Core Team developer comment in Reddit post)

Reddit, U. (2019a, January). *If you haven't churned your monero balance yet, thanks to bulletproofs there is no reason not too.* https://www.reddit.com/r/Monero/comments/ahiszf/if_you_havent_churned_your_monero_balance_yet/eef1ygq. (Monero contributor comment in Reddit post)

Reddit, U. (2019b, November). *Is xmrwallet.com safe and also legit?* https://www.reddit.com/r/Monero/comments/dt6wd1/is_xmrwalletcom_safe_and_also_legit/f6v6k7p. (Monero community user comment in Reddit post)

Reddit, U. (2019c, November). https://www.reddit.com/r/Monero/comments/dxw9ov/scheme_for_reduced_wallet_scanning_time. (Monero community user post in Reddit)

Shinobi, B., & Folkson, M. (2021, April). *Mandatory activation of taproot deployment.* https://github.com/bitcoin/bips/blob/master/bip-0343.mediawiki. (Accessed: 2020-05-20)

Spagni, R. (2018, October). *Monero 0.13.0 "beryllium bullet" release.* https://www.getmonero.org/2018/10/11/monero-0.13.0-released.html. (Accessed: 2020-11-04)

Stackexchange, U. (2017a, July). *Can curve25519 be used for pairing-based cryptography?* https://crypto.stackexchange.com/questions/50387/can-curve25519-be-used-for-pairing-based-cryptography. (Comment in Cryptography Stack Exchange post)

Stackexchange, U. (2017b, July). *What is churning?* https://monero.stackexchange.com/questions/4565/what-is-churning. (Monero contributor comment in Monero's Stack Exchange post)

Stackexchange, U. (2019, March). *Is automated churning of xmr bad for privacy?* https://monero.stackexchange.com/questions/11010/is-automated-churning-of-xmr-bad-for-privacy. (Monero contributor comment in Monero's Stack Exchange post)

Tramèr, F. (2019, November). *Exploiting network and timing side-channels to break monero receiver anonymity.* https://hackerone.com/reports/713321. (Accessed: 2021-05-30)

Tramèr, F., Boneh, D., & Paterson, K. (2020). Remote side-channel attacks on anonymous transactions. In *29th USENIX security symposium (USENIX security 20)* (pp. 2739–2756).

Wang, Y., Wang, J., & Chen, X. (2016). Secure searchable encryption: a survey. *Journal of communications and information networks*, *1*(4), 52–65.

Zhang, Y., Li, Y., & Wang, Y. (2019). Secure and efficient searchable public key encryption for resource constrained environment based on pairings under prime order group. *Security and Communication Networks*, *2019*.

APPENDIX

A. SYNCHRONIZATION TIMES FOR MONERO BLOCKCHAIN SCANS

We provide the timing data to compare between bounded and unbounded blockchain scans. While limiting the depth of a scan is already possible by means of providing a block height, with consolidation transactions we provided a way of limiting the depth by supplying wallets with a balance key, therefore the following experiments are useful for both approaches, but explicitly show the statistics for our proposal.

A.1. Computer specifications

The following measurements were performed in a standard unmodified 13" Early 2015 MacBook Air with 8GB LPDDR3 1600MHz RAM, 128GB PCIe-based flash storage and 1.6GHz dual-core Intel Core i5-5250U(*MacBook Air (13-inch, Early 2015) - Technical Specifications*, 2021), running Ubuntu 20.04.02 LTS. Tests were executed while connected via ethernet to a 900Mb/s fiber optic internet service. The computer was connected at all times to the Mac power adapter.

A.2. Syncronization times

The main focus of the proposal presented in Chapter 3, is to improve synchronization times by avoiding full blockchain scans without compromising privacy. To help gauge the magnitude of the problem, Table A.1 displays a list of syncronization attempts made using the standard Monero CLI wallet connected to a local up-to-date full node[1] using an address and its keys, asking the wallet to synchronize from specific dates in the past.

We use our private keys as inputs for the CLI wallet in order to restore the required information and compute the balance. Therefore, we are recreating a scenario where the wallet does not have any information stored about the address, hence it needs to retrieve

[1]For mobile wallets this is not a likely scenario, access to a remote node would probably be required and therefore introducing the querying latency into consideration. We do not consider that extra time in our tests. This test can be considered as a best case scenario for the specified hardware.

everything from the blockchain. We only limit the scan by providing a date, which is translated by the CLI into a block height from which the scan begins.

We show the amount of days between the provided date and the date of the experiment, the block height for the starting date and the current date, the time taken for the scan and the speed in blocks per second in Table A.1.

Synchronized days	Initial block height	Final block height	Number of blocks	Time (s)	Blocks per second
2	2380459	2382787	2328	5.58	417.58
3	2379295	2382787	3492	8.78	397.86
7	2375805	2382787	6982	17.27	404.24
15	2371151	2382787	11636	29.41	395.69
30	2359516	2382787	23271	69.05	337.0
60	2338573	2382787	44214	138.86	318.41
180	2251312	2382787	131475	350.13	375.5
270	2187322	2382787	195465	468.99	416.78
365	2118677	2382787	264110	564.61	467.78

Table A.1. Synchronization times for wallet restauration with given initial block heights.

As we can observe in the 30 day experiment, it takes more than a minute to synchronize 1 month of transactions.

It is important to note that in Monero the amount of new blocks added per month to the blockchain does not fluctuate majorly. Using data from a block explorer[2] we observe that in the period between December 2019 and May 2021, the amount of blocks ranged

[2]This data is publicly available in `https://localmonero.co/blocks/stats/transactions/m/20` accessed on June 2021.

from a minimum of 20187 to maximum of 22652 blocks, while not showing any evident upward or downward trend, as shown in Figure A.1.

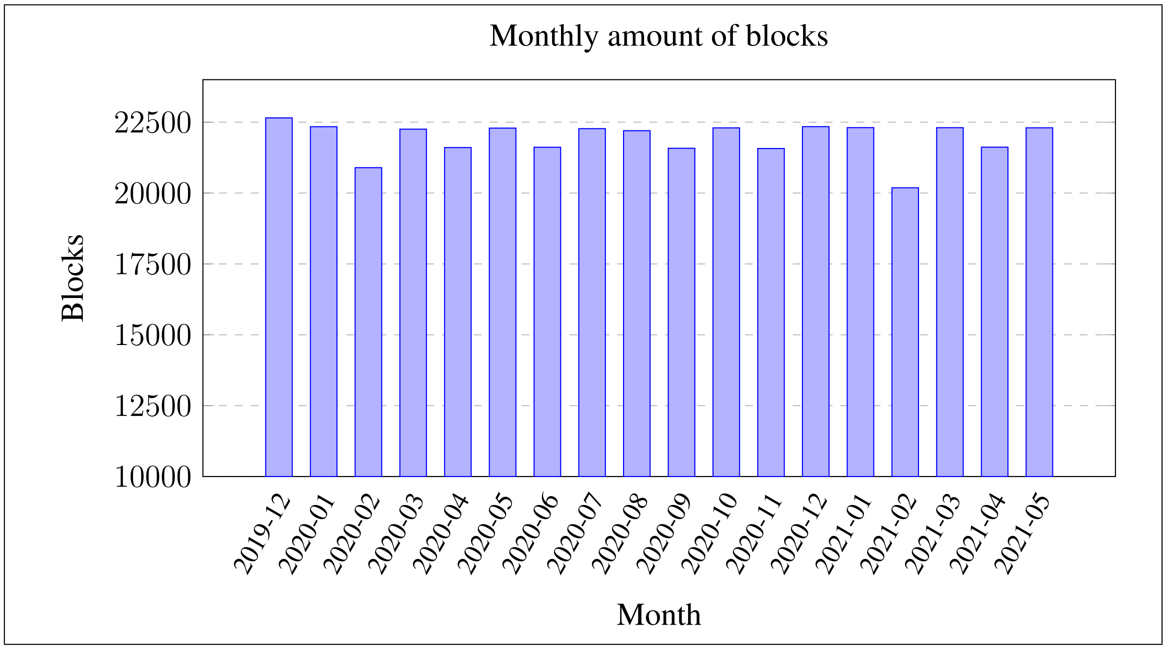

Figure A.1. Blocks per month between December 2019 and May 2021

This behaviour is not an accident. By design, the mining difficulty set by the network aims to achieve a time of 120 seconds in order to mine a block, and it adjusts automatically to keep around that time. This means the amount of blocks per month has remained and should remain around the 22000 mark[3].

However, transactions do not follow the same rule. In the same period, the amount of transactions –and consequently transactions per block– does vary significantly. In fact, it presents an upward trend as shown in Figure A.2.

[3]One month has approximatedly 44000 minutes.

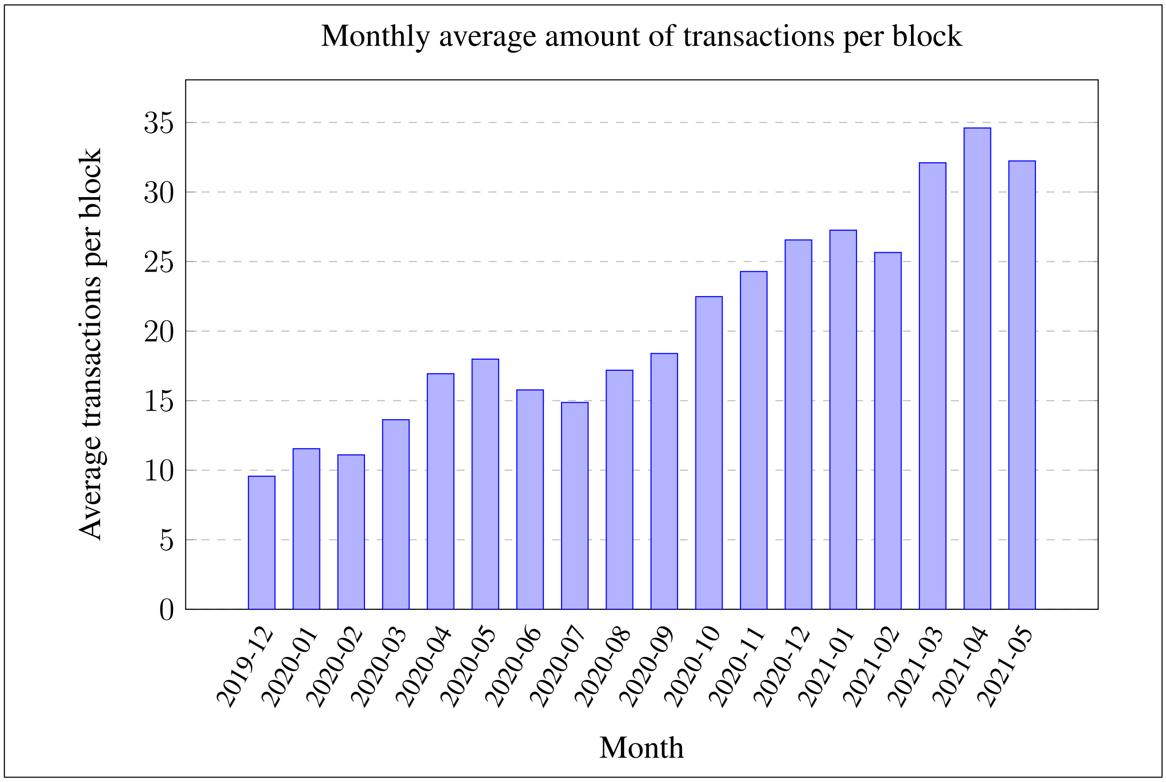

Figure A.2. Average transactions per block for the months between December 2019 and May 2021

The above is relevant because Monero wallets require to scan every transaction, therefore the amount of blocks being added is not as relevant as the amount of transactions. Thus, even when our experiments for time spent to synchronize yielded no significant upward or downward trend[4], the time spent synchronizing could vary significantly. Firstly because the time spent could increase due to a growth in the amount of transactions, and secondly because it could decrease due to improvements in the code required to process every transaction.

[4]Our experiments yielded a weighted average of 413.25 blocks per second. The first 4 experiments gave averages of around 400 blocks per second and it decreased for the 30, 60 and 180 days experiments, however, it increased again in the 270 and 365 days experiments. We conclude it is not possible to determine with our experiments an evident trend.