MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

CoinJoin protocols and implementations analysis

Master's Thesis

DENIS VARGA

Brno, Spring 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

CoinJoin protocols and implementations analysis

Master's Thesis

DENIS VARGA

Advisor: doc. RNDr. Petr Švenda, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Denis Varga

Advisor: doc. RNDr. Petr Švenda, Ph.D.

Acknowledgements

I want to express my gratitude to my advisor doc. RNDr. Petr Švenda Ph.D., for his help, patience, suggestions and provision of resources needed for evaluations. His enthusiasm for my work has kept me motivated throughout the most tedious parts of my work. I also want to thank my family and friends for their support during my entire studies and my partner for helping me make time for and focus on my work.

Abstract

Bitcoin, as we know it now, is not anonymous but only pseudonymous. If we discover a link between an address and its owner, we can track the flow of funds indefinitely. There have been many methods created to increase the privacy in the Bitcoin network to help solve this problem. One of such methods is CoinJoin. This thesis analyses two known CoinJoin protocols and their implementations; Wasabi and Samourai (Whirlpool) CoinJoins. The theoretical part describes these protocols and bitcoin wallets that use them, their security measures and possible attacks against them. In the practical part, we set up these wallets in a testing environment to collect their activity data and network traffic. We then compare the collected data to the specifications written by the developers and discuss any present discrepancies. The last part discusses the most important differences between these implementations ranging from the protocol architectures to wallet functionalities.

Keywords

Bitcoin, CoinJoin, mixing, anonymity, privacy, blockchain, Wasabi Wallet, Samourai Wallet, Sparrow Wallet, blinded signatures

Contents

List of Tables

List of Figures

Introduction

As we know it now, Bitcoin was the first attempt to propose a decentralized cryptocurrency. It does not require any centralized institutions and allows to send currency between peers easily. Thus, since its publication by Satoshi Nakamoto in 2008 [24] and release in 2009, Bitcoin quickly became a popular way to transfer currency between peers and with absent regulations, even attracting illegal activities.

Even though often promoted as a tool for privacy, the problem with Bitcoin transactions is that they are not anonymous but only pseudo-anonymous. Like in every multi-user environment, where each user has a pseudonym (either username or an id), the same applies to Bitcoin, where the actual addresses are considered pseudonyms. Thus, user privacy is broken when a link is discovered between a person and an address.

Every transaction of the bitcoin protocol is stored on a blockchain, which is publicly available. This means that anyone can quickly look up which address has sent money over to which address. Thus, doing transactions on the Bitcoin network is not anonymous. Anyone can trail the funds outgoing from a particular address on the blockchain using a so-called taint analysis.

This reality has created a need for anonymization of Bitcoin funds, leading to the creation of many protocols that could potentially increase privacy and sever the link between a user and his addresses. There are various protocols with varying complexity and different aims to achieve privacy.

One of many such protocols is Coinjoin, a trustless method for combining multiple Bitcoins from multiple users into a single transaction. CoinJoins make it more difficult for potential attackers to determine which output belongs to which user.

There are several implementations of these protocols and software that use them. However, we will mainly focus on these three:

- Wasabi Wallet
- Samourai Wallet
- Sparrow Wallet

The thesis aims to provide an overview, study, and evaluate the behaviour of these three implementations of bitcoin wallets. We installed and analyzed these wallets, collected their logged data and captured their network communications. We then compared the claims in the specification as opposed to the actual behaviour of the implementations. Finally, we discussed the differences between these CoinJoin protocols and their wallets.

In the first chapter, we introduce Bitcoin and aim to inform the reader about the privacy problem in Bitcoin and privacy-enhancing methods commonly used in the Bitcoin network. Chapter two describes ZeroLink, a Bitcoin fungibility framework, proposing a Chaumian CoinJoin protocol. The chapter explains how and why the Chaumian CoinJoin works and describes each phase of the CoinJoin protocol in detail.

The third chapter introduces the three beforementioned privacy-focused Bitcoin wallets that provide the user with an option to participate in CoinJoin transactions. We cover their essential functions, way of synchronizing, and what sort of CoinJoin protocol they provide to their users. In chapter four, we discuss the security of CoinJoin protocols from the perspective of different attack models and their goals. Any differences in the protocols with varying security implications are also pointed out.

Chapter five describes the setup, methodology and results of the assessment tests. Specifically, we go through each protocol and describe the data sent and received from a client's perspective. We then discuss how the implementation adheres to the specifications published by the developers for each protocol. The sixth chapter then compares these protocols from different aspects, ranging from protocol differences to policies for using such wallets.

1 Background

This chapter presents a background overview of the terms and points discussed in the thesis. Its purpose is to briefly explain the workings of Bitcoin, a view of its transactions, the aspects of anonymity in Bitcoin and a brief overview of current methods that increase its privacy.

1.1 Bitcoin

Bitcoin is a decentralized digital cryptocurrency with many advantages over traditional fiat currencies. Bitcoin has no central bank or a single administrator and can be on the peer-to-peer bitcoin network. Bitcoin cryptocurrency was invented in 2008 by an unknown person or a group of people using the name Satoshi Nakamoto. The use of Bitcoin began in 2009 when its implementation was released as open-source software.

The unit of the Bitcoin system is called *bitcoin*. Bitcoin is divisible up to 8 decimal points, meaning that bitcoin consists of 100 000 000 (one hundred million) smaller units called satoshis (sat.). There is also a less used denomination called millibitcoin (mBTC), which is 0.001 of bitcoin.

1.1.1 Transactions

Transactions on the Bitcoin network consist of one or more inputs and one or more outputs. When a user sends bitcoins, the user defines unspent output addresses from previous transactions as inputs to the transaction. Then he defines addresses as the transaction's outputs along with the values intended to send to them. Each input must refer to a previous unspent transaction output (UTXOs, also called coins) in the blockchain to prevent double-spending.

In the Bitcoin network, UTXOs have to be spent whole. When we use a transaction output of ten bitcoins as an input, the whole ten bitcoins are used. Thus, if we want the change, we have to create an output pointing to one of the addresses that we will use to store the change. This way, we are breaking a coin into multiple smaller coins.

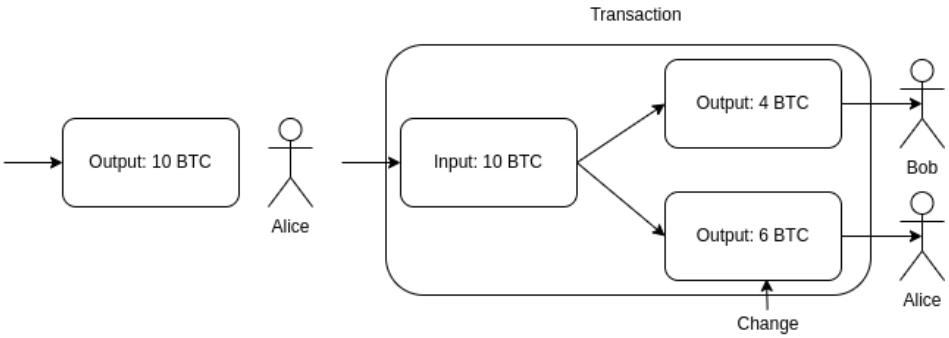

Figure 1.1: An example of a bitcoin transaction where Alice sends Bob 4 BTC and receives change.

An example of a transaction can be seen in Figure 1.1. Alice wants to send Bob four bitcoins but only has a ten bitcoin UTXO. She then creates a new address and adds it as an output in the transaction to act as change.

There are also transaction fees that the users have to pay the miners to confirm the transaction. These miner fees do not show as the transaction output. Therefore, Alice would receive less change than six bitcoins.

1.1.2 Blockchain

The Bitcoin blockchain is a public record of all Bitcoin transactions, implemented as a chain of blocks. Each block consists of a header with a hash of the previous block, timestamp, nonce and the Merkle root hash of all the transactions. The rest of the block is the body, which contains a list of transactions in that block. We can see an example of a Bitcoin blockchain in Figure 1.2.

Nodes in the Bitcoin network all hold a copy of the blockchain and can broadcast these blocks to other nodes. A new block is created and added to the blockchain at varying time intervals, averaging every 10 minutes. The new block is then quickly published to all other nodes.

We can explore these blocks and their underlying transactions using public blockchain explorer, such as OXT. [27]

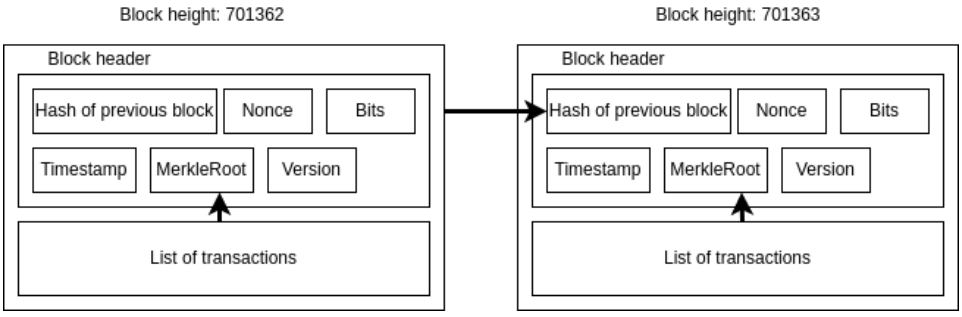

Figure 1.2: An example part of a Bitcoin blockchain

1.1.3 Ownership

Bitcoins are registered to bitcoin addresses. Creating a bitcoin address is a simple process. One needs to generate a random private and public key pair and compute a corresponding bitcoin address by deriving it from a public key. By owning a private key, an owner can prove the ownership of an address by providing a signature for it with the corresponding private key, which others can then verify.

Nowadays, bitcoin wallets are created as hierarchical deterministic wallets described in the bitcoin improvement proposal 32 (BIP32). [49] These wallets feature one extended key pair (a seed), from which a user can derive multiple key pairs. As such, it is only necessary to remember one key pair to be able to recover the whole wallet with various addresses.

1.1.4 Mining

Whenever a new transaction is created, it is sent from a node to its peers, who will then pass it to further peers. The transactions are yet unconfirmed, and nodes store them in a mempool. Mempool is a mechanism for storing information on unconfirmed transactions. Its purpose is to be a kind of waiting room for the transactions waiting to be put into a block and confirmed.

Miners select transactions from the mempool to create a block and then attempt to put it in the blockchain by computing a proof-of-work (PoW). Proof-of-Work requires miners to find a number, a nonce in the block, such that when the block content is hashed along with the

nonce, the result is numerically smaller than the network's difficulty target. Each block stores a packed representation (called "Bits") for its actual hexadecimal target.

Proof-of-work is easy for any node to verify, as it is a single hash and compare function, but it is extremely time-consuming to generate such a hash. Miners must try many different nonce values before a result happens to be less than a difficulty target. The difficulty target is typically tiny compared to a typical SHA-256 hash. Thus block hashes usually have many leading zeros, as shown in an example.

000000000000000000338be57b0541cd1ad8f90dfc9142569140af068b56470

With the proof-of-work, the miner will add the block on the top of the blockchain, and the miner will receive a reward for mining in the form of transaction fees and a new block reward. As of the time of writing, the miner reward for a new block is 6.25 BTC.

1.2 Privacy in Bitcoin

Bitcoin does not offer complete anonymity. Anonymity for a particular element, be it either human or a computer, refers to the property of that element of not being identifiable within an "anonymity" set. An anonymity set could be defined as a group where the specified element is observed. In Bitcoin protocols, every transaction input or output is linked to an address. Therefore we can assign an owner of these bitcoins to an address.

Every action that the owner does with the bitcoins on that address can be linked back to the initially owned address, meaning that the Bitcoin network does not offer anonymity, only pseudonymity. This becomes an issue when specific addresses become linked to a real-world individual. These occurrences could happen when buying bitcoin Bitcoin exchanges where users have to input their real-life credentials. Therefore, all further transactions can be linked to that individual, and even the pseudonymity will be broken.

1.3 Privacy-enhancing methods

Pseudonymity in the Bitcoin network has sparked an incentive to anonymize Bitcoin users. People have invented many methods to

increase users' privacy, ranging from simple to relatively complex protocols with varying degrees of anonymity.

1.3.1 Not reusing addresses

One of the simplest methods to increase users' privacy is not reusing Bitcoin addresses. Satoshi Nakamoto has proposed the method in the original whitepaper. The method breaks the link of a common owner of bitcoins such that an observer cannot tell how many bitcoins the owner of an address has in total.

However, some linking would be unavoidable with multi-input transactions, possibly implying that the same owner owned their inputs. The implication is also called a common-input-ownership heuristic. [1] It assumes that all inputs to a transaction are from the same owner.

1.3.2 PayJoin

PayJoin is an attempt to break the common-input-ownership heuristic. In essence, it is a technique for paying someone while including one of their inputs in the payment. Therefore, PayJoin enhances the privacy of both the spender and the receiver.

PayJoin is a BIP (Bitcoin improvement proposal) with the number 78, proposed by Nicolas Dorier. [9] Including the inputs from both the spender and the receiver, PayJoin makes it difficult for an observer to determine which inputs and outputs belong to what participant.

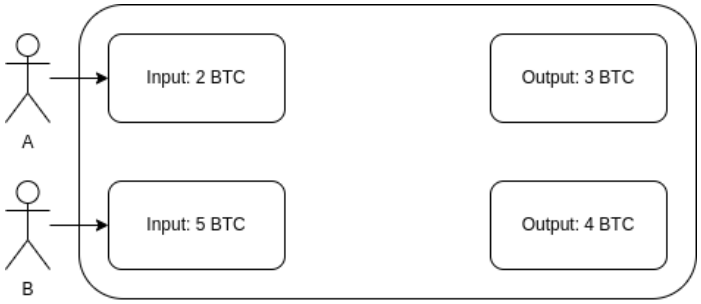

Figure 1.3: A PayJoin example, where B is paying A. To an onlooker, A and B could seem like the same entity.

In Figure 1.3, we can see an example of a PayJoin transaction. The transaction could be interpreted in many ways. In one way, it could be a simple transaction with actors A and B being the same person, and it is a payment to someone with a change.

However, it could also be a transaction with inputs from two different people, with actor B paying one bitcoin to actor A. We have no way to tell which one of these interpretations is correct.

If PayJoin became moderately used, the common-input-ownership heuristic would be an utterly flawed assumption. It would not be necessary to have every transaction be a PayJoin one, but having enough PayJoins would invalidate it.

1.3.3 Atomic swap

Atomic swaps, in essence, are swapping one cryptocurrency for a different one, for example, between Bitcoin and Monero. The most crucial feature of atomic swaps is the anonymity they provide. While Bitcoin is only pseudo-anonymous, there is no way to trace it once it has been swapped for Monero.

Monero is a privacy-oriented, decentralized and privacy-oriented cryptocurrency.[47] In Monero, there are privacy-enhancing technologies in place that obfuscate transactions to achieve anonymity and fungibility. While exploring the blockchain, a simple observer cannot decipher addresses, amounts, balances, or transaction histories.

Thus by swapping bitcoins for moneros and then back again, the user can sever a link between his old addresses and transactions done in the past. Swapping bitcoins for moneros on the main Bitcoin network has been available since August 2021. [23]

1.3.4 Schnorr signatures

Schnorr signature is a digital signature described by Claus Schnorr. [36] The algorithm makes efficiently computable and verifiable signatures of short length. The Schnorr scheme was covered by a US patent until 2008, heavily restricting its utilization. Although Bitcoin was invented at the same time as the patent expired, it was decided that Schnorr signatures lacked the popularity and testing to secure Bitcoin.

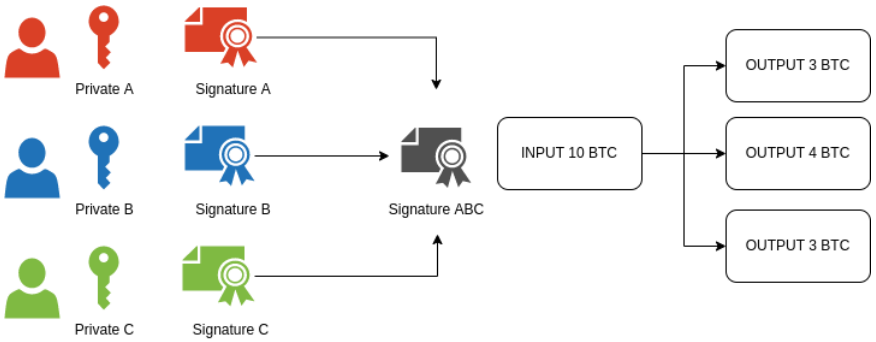

Figure 1.4: An example of a Schnorr signature where multiple users coordinate to create a single signature.

The most significant advantage offered by the Schnorr signature scheme is key aggregation. Many Bitcoin transactions contain multiple inputs. These inputs require different signatures, which means that owners of the inputs must enter all of these signatures into the transaction. All the data must be sent over the network and included in a block. With Schnorr signatures, all inputs require only one combined signature representing all signatures.

Schnorr public keys and signatures can be aggregated so that if three parties want to sign a transaction, they can combine their three public keys to form a single public key. In the same way, they can combine their three signatures, which would be valid for the aggregated public key. A verified must only use the one aggregated public key to ensure that the signature is correct. We can see an example of a Schnorr aggregated signature in Figure 1.4.

Combined Schnorr signatures help decrease transaction sizes and have significant privacy implications. Because multiple parties can aggregate keys and signatures, multisig transactions resemble a typical, single party transaction. Like PayJoin, Schnorr signatures break the common-input-ownership heuristic.

1.3.5 Centralized mix

Centralized mixers are platforms that accept users' bitcoins and send back different bitcoins for a fee. They provide an easy solution for

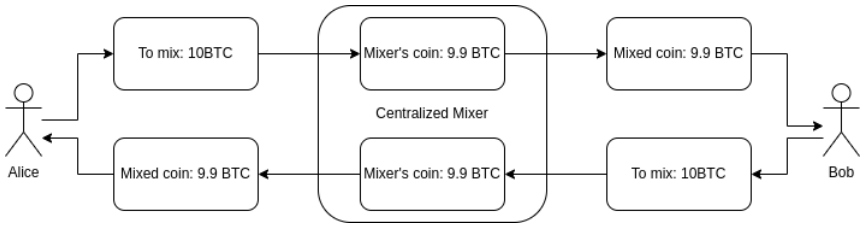

Figure 1.5: An example of a centralized mix. Alice receives UTXOs previously owned by Bob, and Bob gets UTXOs previously owned by Alice.

anonymizing bitcoin, but they still present a privacy challenge. We can see an example of mixing in Figure 1.5.

These centralized mixers carry the information and link between the incoming and outgoing bitcoins. Thus, mixing with centralized mixers requires trust in their services, that they do not keep a record of links, and the users receive their coins. One such example of a centralized mixer is Blender.io. [3]

1.3.6 Decentralized mix - CoinJoin

Unlike a centralized mix, a decentralized mix employs protocols to entirely obscure transactions. These protocols can either be coordinated or a peer-to-peer method. One of these methods is CoinJoin. Coinjoin was first detailed in 2013 by Gregory Maxwell on BitcoinTalk [22].

He states that when N amount of users agree on uniform output size and provide inputs amounting to at least that size, the transaction would have N outputs of that size. Potentially this transaction could have N more if some users provided input above the target. A CoinJoin transaction would then obfuscate which of the user owns which output.

A CoinJoin is possible to do without trusting the other participants. For each input, the users must sign a transaction separately and then merge their signatures so that the transaction can be transmitted. It cannot happen when one participant decides not to sign the transaction. Even when the user has signed the transaction, he has signed that specific transaction, which can not be misused. Therefore there is

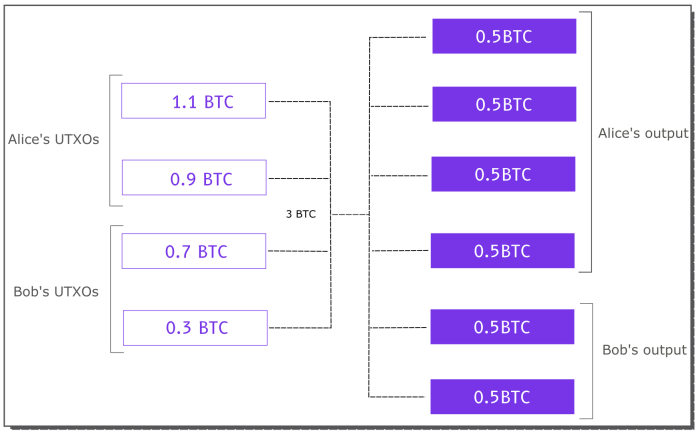

Figure 1.6: An example of CoinJoin for mixing UTXOs [29] Alice and Bob decide to pool their UTXOs into one transaction with common denomination of outputs.

no risk of theft at any point. An example of a CoinJoin can be seen in Figure 1.6.

Gregory also proposed that we can also use the idea more casually. When users want to make a payment and find somebody else who also wants to make a payment, they can create a payment together. Doing so increases privacy a little and creates a smaller transaction size. We can see an example in Figure 1.7.

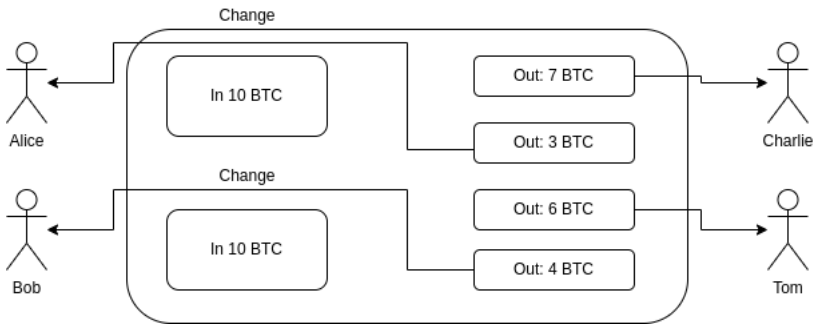

Figure 1.7: Alice and Bob use a CoinJoin to pay Charlie and Tom. An observer can not tell who is paying whom.

In the case of Wasabi and Samourai, both use the implementation of a chaumian CoinJoin called ZeroLink [50]. We describe the protocol and its parts in detail in the following chapter.

2 ZeroLink

The ZeroLink framework defines a pre-mix and a post-mix wallet and a mixing technique. Pre-mix wallet functions as any other Bitcoin wallet without much overhead. On the other hand, post-mix wallets have strong privacy requirements and policies. These requirements for both wallets together define the Wallet Privacy Framework. [50]

Coins from pre-mix wallets are moved to post-mix wallets by mixing. There are many techniques for mixing (i.e. CoinShuffle), but ZeroLink defines its own technique: Chaumian CoinJoin. As the name suggets, Chaumian CoinJoin uses Chaumian blind signatures. The contents of messages to be signed are disguised (blinded) before the actual signature, and the requestor can later unblind the signed content. By unblinding the message from the signer, he will acquire a signature for his data without the server knowing its content. [4]

2.1 Chaumian Blinding

In an analogy, Chaumian Blinding could be imagined as a physical act of a voter enclosing an anonymous ballot in a carbon lined envelope with the voter's credentials printed on the outside. An official will verify the envelope and sign it, effectively signing the ballot inside the envelope via the carbon liner. The voter can then take out the inside of the envelope, which the official will sign without knowledge of the contents of the ballot. [4]

Chaumian blinding is based on RSA signing. A traditional RSA signature is computed by raising the message m to the exponent d modulo the public modulus N. For blinding, we have to generate a random value r, which is relatively prime to N, and then raise it by the exponent e. The resulting $r^e \mod N$ is used to blind the message, to create $m' \equiv mr^e \mod N$.

The signing authority, who receives m' can sign the message by raising the blinded message to exponent d, creating a signature for the blinded message $s' \equiv (m')^d \mod N$

The signature is sent back to the author, who can remove the blinding factor to reveal $s \equiv s'.r^{-1} \mod N$. This works, because RSA keys satisfy the equation $r^{ed} \equiv r \mod N$, therefore:

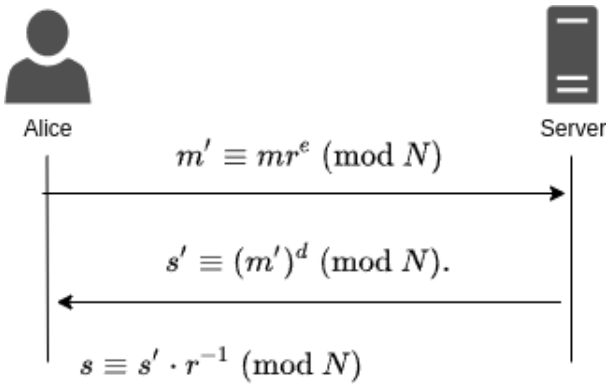

Figure 2.1: Alice gets a signature using the Chaumian blinding while the server does not know the contents of m.

$$s \equiv s'.r^{-1} \equiv (m')^d.r^{-1} \equiv m^d.r^{ed}.r^{-1} \equiv m^d.r.r^{-1} \equiv m^d \mod N \tag{2.1}$$

2.2 CoinJoin Phases

The ZeroLink Chaumian CoinJoin protocol is split between multiple phases. We will describe each one in more detail to understand and identify the messages sent between the client and the coordinator during network analysis.

2.2.1 Input Registration

The user selects which UTXOs he wants to register to the protocol during the input registration phase. The client has to generate input proofs by creating signatures to prove that the registered inputs belong to him. Then the client generates the output addresses for the result of CoinJoin. These addresses are cryptographically blinded, meaning that they are not known to the coordinator when he receives them. The only not blinded address is the change address, which can be easily linked to the input; therefore, it is sent in cleartext.

After that, the client generates a new tor identity (Alice). Alice is a separate entity and is generated again for each round of CoinJoin.

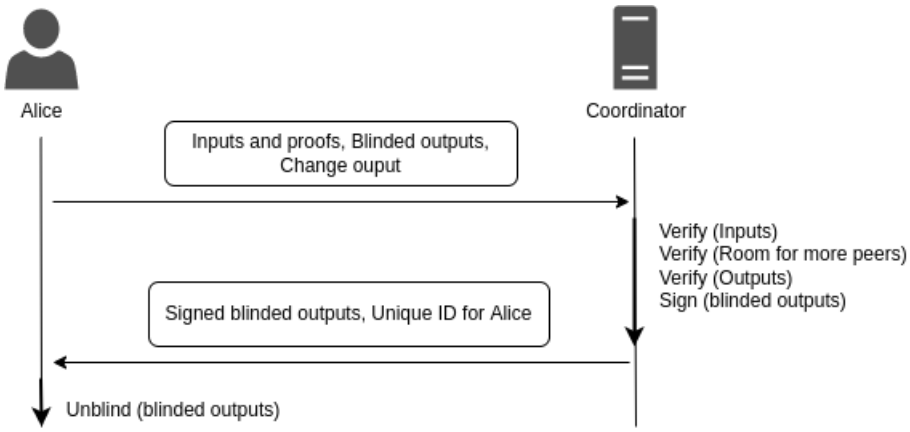

Figure 2.2: Alice registers her inputs in the input registration phase.

Alice sends all the generated components to the coordinator server. The sent data includes:

- Input coins that the user wants to register, along with their proof signatures
- Blinded set of output addresses
- Cleartext change address

Now the coordinator has to verify a few things:

- There is room for more peers
- The output has not been registered before
- The input is confirmed, not registered before, spent, and the proof is valid
- The sum of the inputs is higher than the minimum amount to join (0.1 BTC)

After everything checks out, the coordinator signs the blinded outputs. Since they are blinded, the coordinator has no idea what address he is signing. The signature is proof that Alice is not cheating and is a valid peer. The following message that is sent back to Alice contains:

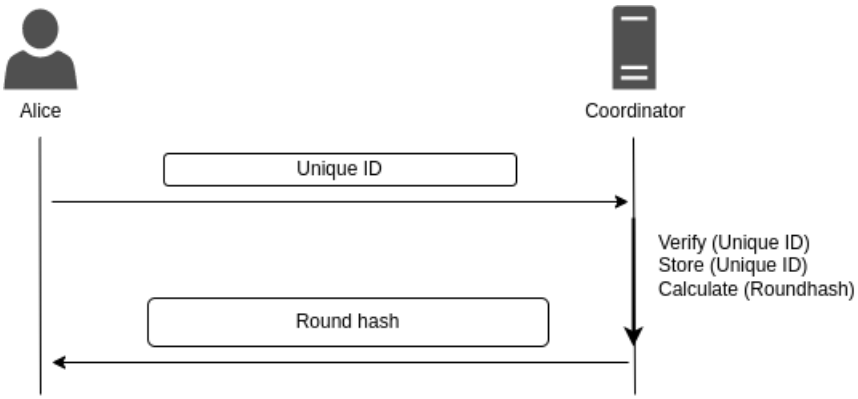

Figure 2.3: Alice polling the server during the connection confirmation phase

- Signed blinded outputs
- Unique ID for Alice in the particular round

The phase ends when the necessary number of registered inputs is registered or when the CoinJoin has not yet finished, is between rounds, and it has been one hour since the last transaction. The protocol then continues to the connection confirmation stage.

2.2.2 Connection confirmation

The coordinator must confirm that every user (Alice) is still connected, online and ready to continue. The confirmation phase takes a long time, as there are a lot of different users. Clients will send the server their unique ID and return a round hash of all the registered inputs. The round restarts if it does not reach a desired minimum anonymity set. The phase ends when everyone has provided their unique ID or at the end of the confirmation round when the number of online Alices is sufficient.

2.2.3 Output registration

Every Wasabi client now generates another Tor identity. Let us call it Bob. Bob is not tied to the previous Alice, but he is still the same user as Alice. Bob sends these items to the coordinator server:

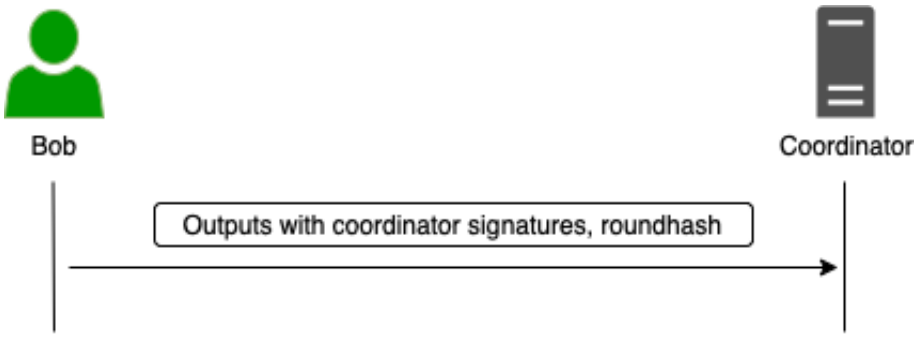

Figure 2.4: Bob posting cleartext outputs to the server during the output registration phase.

- The cleartext addresses for the CoinJoin output along with its signature
- Round hash of all the inputs

These addresses come from the blinded output addresses registered by Alice. As unblinding them does not remove the coordinator's signature, the coordinator can verify that the output and the peer are valid.

The coordinator cannot link Alice to Bob since he has never seen the beforementioned addresses in cleartext; thus, the peers cannot be deanonymized.

The phase ends when the value of cleartext outputs and the change is equal to the value of inputs, meaning that all the Bobs have registered. If the client's output registers time out, the coordinator will abandon the round, the dropped out peers are banned, and a new round will start.

2.2.4 Signing

At the end of the output registration phase, all clients have registered their inputs and outputs, and the coordinator can start the signing phase. It begins by creating the transaction with all the registered inputs and outputs, with the coordinator fee and change outputs in mind.

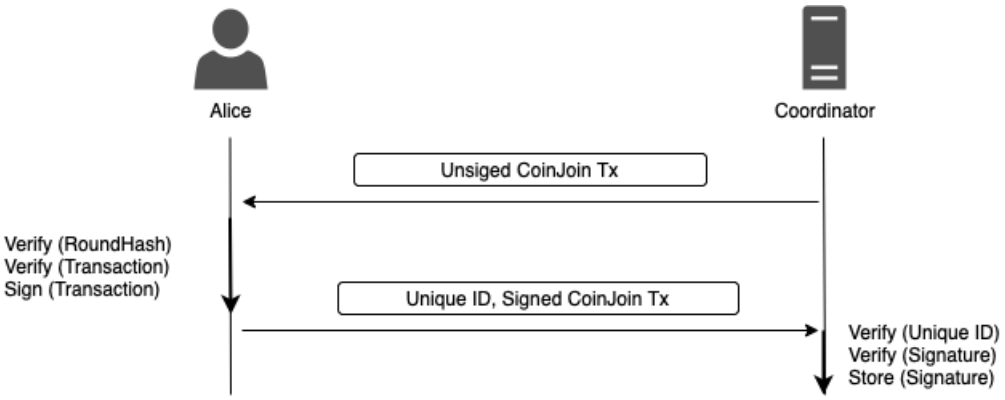

Figure 2.5: The signing phase, when Alice receives the CoinJoin transaction in order to sign it.

The created transaction is then sent to all the Alices of the round. Each Alice then has to verify that the round hash is equal to the hash of all inputs in the transaction and that the coordinator has included her inputs and outputs.

When the verification of the transaction succeeds, Alice will sign the transaction with the private keys of her inputs, and she sends her UID, signature and the input index to the coordinator, which is again verified.

The phase ends once the coordinator has received the signature message from all Alices and he confirms that all of the signatures are valid.

2.2.5 Broadcasting

In the broadcasting phase, the CoinJoin transaction is built and signed and can be broadcast to the bitcoin network. The coordinator sends the transaction over the Tor network to a random bitcoin node. The transaction is then further broadcast to other nodes and miners.

3 Overview of implementations

During the development of ZeroLink, the creators had different ideas on how to move forwards. Unable to compromise, they each created a hard fork of ZeroLink and made their own wallets and mixers. These two wallets are now known as Wasabi Wallet and Samourai Wallet. There is also an independent wallet called Sparrow Wallet, which uses Samourai's mixing services; we will briefly go over it as well.

In this chapter, we will create an overview for each implementation and describe their version of the Chaumian Coinjoin, so we can better analyse and differentiate any deviations in the real world implementations compared to their specifications.

3.1 Wasabi wallet

Wasabi is an open-source, non-custodial, privacy-focused Bitcoin wallet owned by zkSNACKs. [53] zkSNACKs claim that the estimated amount of Wasabi Wallet downloads is 200 000.

Wasabi allows users to create and load Bitcoin wallets and send and receive coins like any Bitcoin wallet. What makes Wasabi stand out from other ordinary Bitcoin wallets is the additional functionalities oriented towards privacy.

The most basic and essential feature is address reuse prevention. Address reuse prevention means that a new unique address is generated whenever a change output is sent "back" to the user instead of reusing an old one.

A more advanced feature would be coin control and mandatory address labeling. To receive any Bitcoin into the wallet, the user must label the address, thus, marking the coins incoming onto that address. When the user wants to spend his coins, this allows him to prevent mixing different sources of coins, thus increasing privacy measures as separate UTXOs can be chosen for transactions individually.

To hide the user's activity further, Wasabi includes the Tor package and uses it by default to route the communication between the client and the server. It is essential to point out that to function, Wasabi needs a central server to get the correct UTXOs for a given wallet and use CoinJoin. There is also an option of running a different full node

software (Bitcoin Core), and Wasabi will use the local downloaded blocks instead of asking for blocks from the centralized server.

As of the time of writing, zkSNACKs has announced a beta testing of *Wasabi 2.0* along with a new CoinJoin protocol called *WabiSabi*. [39] We will not cover Wasabi 2.0 implementation in this thesis.

3.1.1 Synchronisation

Every wallet needs to know how many coins (UTXOs) it currently has and synchronize with the blockchain on every startup. There are various methods for synchronizing with varying speed, privacy and storage needs. Wasabi is foremost a lightweight wallet, which means that it does not use a full node or does not store the blockchain with it. However, if a system has a blockchain stored, it can use it, thus removing the necessary dependency during synchronization.

The wallet only requests BIP 158 filters [44] from the centralized server to prevent leaking transaction information to attackers able to decrypt the communication with the server. These filters represent a compact list of addresses in one block. Wasabi locally checks if any of these filters contain transactions with the wallet addresses. After knowing which blocks have the UTXOs for the given wallet, Wasabi downloads them in addition to some fake ones from bitcoin peers. The attacker can not easily guess which addresses belong to the wallet, whether he is an eavesdropper or the server.

3.1.2 Wasabi CoinJoin

CoinJoin, as described in the Wasabi documentation, does not deviate from the original ZeroLink framework proposal. [40] The developers have decided to implement the ZeroLink framework as it was previously designed with large transactions breaking inputs into common denominations.

It should be noted that even though they follow the ZeroLink framework, Wasabi does not use Chaumian blinding anymore. During its update from 1.0 to 1.1, Wasabi has changed the Chaumian RSA Blinding to Schnorr Signature Blinding, which achieves the same goal. [12]

For every CoinJoin, the participants have to pay a fee to the coordinator. The participants pay the Wasabi coordinator a percentage cut of their input. The percentage is 0.003% times the anonymity set of the transaction for their input. For example, if the anonymity set is 50, a client pays 0.003% * 50, which equals 0.15% of the input intended to mix. There are also edge cases where the peer does not pay the coordinator or expends more.

The fee paid to the coordinator is done so during the CoinJoin transaction, as the coordinator's address will be one of the transaction's outputs.

3.2 Samourai Wallet

Just like Wasabi, Samourai wallet [19] is a free, open-source and non-custodial bitcoin wallet software. It primarily focuses on mobile applications (android) and has over 100 000 downloads on Google Play Store.

Samourai offers all the functionality that an average Bitcoin wallet does but has a different architecture. The wallet is divided into five "accounts". The user only interacts with two of them, the deposit and post-mix wallet.

- **Deposit** - consists of all outputs within the standard Samourai Wallet. Funds in the deposit account have not been cycled through Whirlpool yet.

- **Pre-mix** - consists of all the outputs that are prepared to mix in Whirlpool but are still pending. The users do not see or interact with the pre-mix account directly. The outputs remain there until confirmed and selected for a Whirlpool cycle.

- **Post-mix** - consists of all the outputs that have completed at least one cycle in Whirlpool. These outputs are available for spending or open to "remix" for free and increase their privacy.

- **Ricochet** - account acts as an address for each 'hop' of a Ricochet transaction. We will not cover these transactions here.

- **Bad bank** - a feature that will act as a storage for the "doxxic" (that can doxx a user) change in Whirlpool.

Thanks to separating mixed and non-mixed UTXOs, the user cannot accidentally spend unmixed and mixed UTXOs together, thus preventing revealing a link between the addresses.

Like Wasabi, Samourai also defaults to routing its traffic via the Tor network and needs to cooperate with a centralized server to work. Like with Wasabi, there is an option of running a full node.

There are more features that Samourai Wallet has to offer, but they are not a part of our scope. These features are documented along with tutorials on their documentation site [32].

3.2.1 Synchronisation

As Wasabi, Samourai is also a lightweight wallet that does not come with a complete local copy of a node. Therefore, it must cooperate with Samourai's default servers to find out what funds are available to the user's addresses. This, of course, can be mitigated by using Samourai's Dojo software to run a private Bitcoin node to download blocks.

Currently, if the user is not hosting his Dojo, Samourai will ask for his transactions from the default servers. The clients send the user's extpubkeys to the servers, so the client can then later check his balance by asking the server for all the transactions with those addresses. Thus the hosting server will know any sorts of transactions pre- or post-mix addresses to the server.

Samourai is not a completely trustless lightweight wallet if using their default servers. It requires trust in the centralized server. Even when using a privately hosted dojo creates a vulnerability on the network level, as the attacker can intercept the traffic and decrypt the extpubkey.

3.2.2 Whirlpool CoinJoin

Samourai Whirlpool took a different architectural approach than Wasabi. Whirlpool transactions have a strict five input and five output structure with identical input and output values. Samourai also distinguishes several pools for mixing depending on the amounts being mixed. At

the time of writing, there are four pools, each determining minimum and maximum amount a user can cycle through that pool.

- 0.001 Pool - Minimum funds of 0.001 BTC, pool fee: 0.00005 BTC, maximum: 0.025 BTC
- 0.01 Pool - Minimum funds of 0.01 BTC, pool fee: 0.000425 BTC, maximum: 0.75 BTC
- 0.05 Pool - Minimum funds of 0.05 BTC, pool fee: 0.0014875 BTC, maximum: 3.75 BTC
- 0.5 Pool - Minimum funds of 0.5 BTC, pool fee: 0.014875 BTC, maximum: 35.00 BTC

In general, cycling a little over 0.5 BTC in the 0.01 Pool would create 50 outputs of 0.01 BTC. We can see a sample transaction here.

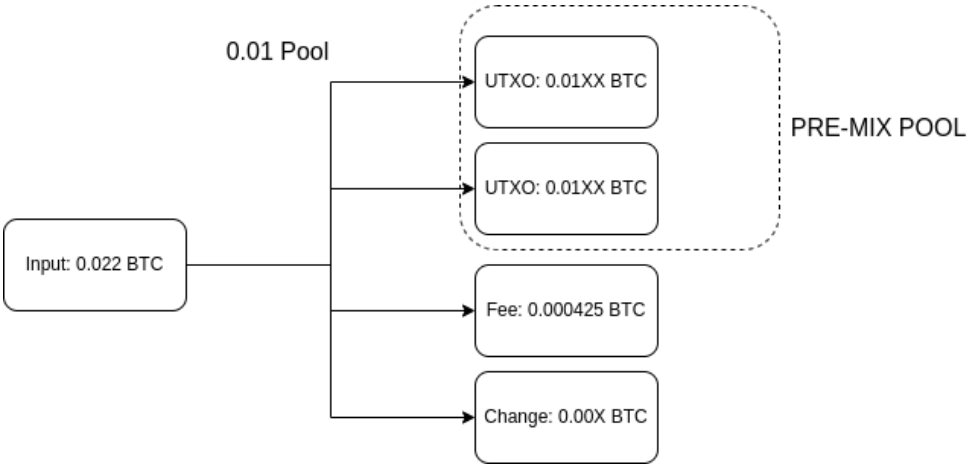

Figure 3.1: 0.022 BTC being split into denominations by the Samourai tx0 transaction into a 0.01 pool

Whirlpool protocol is performed in two steps. The first step is what Samourai calls a "tx0" transaction. Tx0 is a transaction which splits the UTXOs into the pre-mix wallet. These UTXOs in the pre-mix wallet have the amount of the pool denominations along with the miner fees for the mixing transaction. The pool fees are paid during the "tx0",

and any change is also outputted into a different address. The change will be marked as "doxxic", and Samourai wallet will try to prevent the user from spending it and warn him of the risk. An example can be seen in Figure 3.1.

The second step is the CoinJoin transaction itself, which does not differentiate from the ZeroLink CoinJoin in principle. Whirlpool follows each phase of the ZeroLink CoinJoin as described in the framework, except not needing to handle coordinator fees or the change outputs, as tx0 takes care of those.

3.3 Sparrow Wallet

Sparrow is another open-source Bitcoin wallet. It is available for Windows, Mac and Linux. We could not find a statement regarding the number of downloads, but we could find the download statistics from GitHub, which are around 78000 during the whole lifetime of the project. [15]

Sparrow offers similar functionality as the two previous wallets. The one functionality that is unique to Sparrow out of all three wallets mentioned here is that Sparrow offers a built-in transaction editor that also works as a blockchain explorer. [37]

3.3.1 Synchronisation

Sparrow uses a different method of receiving transaction information than the previous two wallets. It depends on the Electrum server protocol for synchronizing itself with the blockchain.

An Electrum server is a Bitcoin address index. It indexes UTXOs by their addresses. We can provide an electrum server with an arbitrary address, and it will return transactions associated with that address.

Sparrow, by default, connects to public electrum servers without Tor. This requires a trust in the selected public Electrum server not to log these requests, as these could easily break the anonymity of any CoinJoin. The server could, for example, log the address request before a CoinJoin with it is made, and then, when the client requests blocks for the output address of the mix, the server can link it to the input of the mix.

Sparrow allows the user to access a private bitcoin node, a private electrum server, or provide Sparrow with an onion link for an electrum server to synchronize the wallet without the inherent trust in the central server in the Tor network.

3.3.2 CoinJoin

Sparrow does not implement any new CoinJoin protocol but uses Samourai's Whirlpool. Sparrow uses Samorai's Whirlpool libraries and connects to their pool mixing servers. Therefore, it is essentially a different wallet if a user wants to use Whirlpool without being a part of the entire Samourai ecosystem or wants a desktop wallet.

4 Security of coinjoin

This chapter discusses various possible attacks on the Wasabi Wallet and the Samourai Wallet. Firstly, we will define the attacker's goals and the potential attack vectors. After that, we will describe attacks that are yet known to us and their effects on the privacy of CoinJoin.

4.1 Attacker goals and models

CoinJoin protocols aim to anonymize UTXOs and cut the link of their outputs to the original owners. They achieve it by constructing transactions on the Bitcoin network that obfuscate the ownership of the output's addresses. There are three main goals that the attacker can attempt to achieve:

- Link inputs and outputs of CoinJoin transactions to further track the input owner's funds, ultimately attempting to undo the CoinJoin.
- Potentially steal the bitcoins of participants in the CoinJoin
- Interfere or prevent mixes from being completed

An adversary can attempt at one or more of these goals either passively, i.e. only observing the information being transferred, or actively trying to manipulate the communications or the system.

In this section, we will examine the possible models of the attacker depending on the data he can access. The different models are naturally not only restricted to the set of data they obtain in their position. These models have access to the public data, the transaction itself and all the transactions before and after the CoinJoin. We only have one model restricted to it, and all other models can use the public information as an auxiliary.

It is important to note that every mentioned attacker model can repeat attacks of the beforementioned models. For example, a malicious CoinJoin participant can also use attacks available to a model restricted to viewing public data. A malicious coordinator can also imitate malicious clients or act as a man-in-the-middle. Therefore, for

each attacker model, we will mention attacks unique to its position regarding the protocol or the network.

4.2 On-chain data analysis

The on-chain data analysis model describes the attacker as he analyses the blockchain after the transaction, meaning that the attacker only has access to the blockchain. While it looks limiting, it is still possible to reveal certain UTXOs of CoinJoin transactions.

4.2.1 Transaction analysis

The first action that the attacker can take is analyzing the transaction itself. This means he can look at the inputs and the outputs and try to recognize a pattern or group-specific inputs and outputs based on their values. An excellent example of an analysis breaking privacy is CoinJoin Sudoku [7].

Let us imagine we have a transaction with an equal number of inputs with an equal amount of outputs, and even the amount of coins is comparable on both sides of the transaction. Specifically, we can take a look at Figure 4.1.

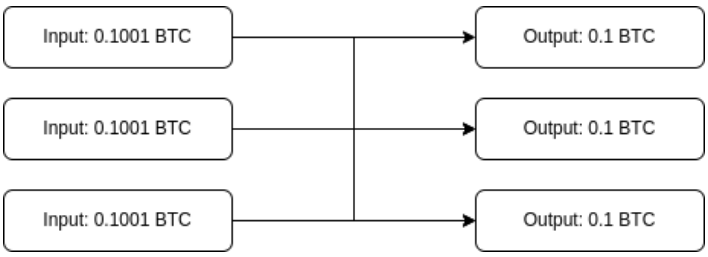

Figure 4.1: A perfect CoinJoin transaction where we can not tell which output belongs to which input

Here, three different users, or just one using three separate addresses, have inputted 0.1001 BTC to the transaction, and the three output addresses have received 0.1 BTC. The remaining part was used as a miner's fee to have the transaction confirmed. Due to all the inputs being equal and the outputs being the same, it is a perfect CoinJoin

transaction with no information leaks. Any input could be linked to any output with the same probability.

Figure 4.1 shows that all inputs and outputs must be the same to prevent information leaks. This condition is hard to achieve in the real-world application as users usually have different values of UTXOs they want to mix. There are two possible answers to that problem.

We could choose only to accept users with the same amount of funds, restricting the number of users who can participate, which creates long waiting times and can be frustrating for users eager to anonymize their coins. We could further solve the problem by reducing the number of participants in the transaction. Next, we could make users split their UTXOs before the transaction, thus leaving a change that must be handled separately and labeled as a potential privacy risk. The Samourai wallet chose this approach.

Alternatively, we can accept that the users will have different input values and create an algorithm to divide the total input value into as many identical outputs as possible whilst returning non-mixable change. Wasabi wallet has chosen this alternative.

However, Wasabi's approach opens up information leaks in which specific inputs can be linked to their outputs. In an intuitive example, when a user input coin has a lower value than some of the outputs, we can automatically assume that these outputs do not belong to the user. Thus, the anonymity set does not strictly rise with the number of participants in the transaction.

With these large Wasabi transactions, analysis is complicated. At first glance, even trying to figure them out manually proves problematic. Even automatic analysis without sophisticated heuristics is challenging. Figuring out which inputs and outputs are linked is a subset-sum problem. The subset sum problem is when given a set of numbers; is there a subset whose sum is zero? We can apply this problem to CoinJoin transactions by treating the input as positive values and outputs as negative values.

Many algorithms solve these problems, but their complexity is exponential at best. However, for non-equal change outputs, an attacker can do a feasible subset-sum analysis and find a link between inputs and change outputs. In most cases, especially when mixing large amounts, we can create the connection easily. Therefore, it is best to assume that the change's anonymity set size is zero.

4.2.2 Taint analysis

Another action the attacker can take is doing a taint analysis of the outputs and seeing where they have been used and ended up. Even though CoinJoin transactions depend on the algorithm's efficiency and security, incorrect handling of CoinJoin outputs can defeat the purpose of the entire mixing transaction. Naturally exposing one or more outputs of a CoinJoin, the anonymity set of other outputs strictly decreases along with the number of outputs linked back to their inputs.

If a user pools together some of the CoinJoin outputs to a single transaction again, he substantially decreases the anonymity set of the entire mix. For example, if a user pools together two 0.1 BTC outputs of one CoinJoin, an attacker can assume (under the common input heuristic) that these two 0.1 BTC outputs cannot be linked back to CoinJoin inputs that were lower than 0.2 BTC. Of course, this could also be a variant of a PayJoin and could leave an attacker with false assumptions.

Another way to expose the ownership of outputs is mishandling the change and pooling it with other transaction outputs. As CoinJoin changes can be linked to inputs, their anonymity set is low, and any further UTXOs used together with change will have their anonymity sets reduced to that of the change. Other attacker models such as coordinator and a man-in-the-middle can undoubtedly tell which change belongs to which inputs from the data in the communication protocol.

4.2.3 Wasabi advisory

It should be noted that LaurentMT and ErgoBTC of OXT Research have released two alleged vulnerabilities of Wasabi CoinJoins. [17] They claim that these vulnerabilities supposedly cancel out the effect of remixing if an adversary has knowledge of all coins in a user's wallet.

The Wasabi team, zkSnacks, disagreed with these claims and wrote that they are trying to undermine Wasabi's reputation in the hopes of getting more users for Samourai. [25]

4.3 Malicious client

The malicious client attack model represents a malicious client that could theoretically represent not just one but multiple of them. A malicious client does not have any data about other participants of the CoinJoin transactions, but there are still two possible attacks.

4.3.1 Sybil attack

A Sybil attack is a possible way to easily distinguish the user's UTXOs resulting from a CoinJoin transaction. It is done by registering as many fake peers in the mixing process as possible. This way, the attacker knows which UTXOs are his and can rule out other users.

There are a few barriers that make Sybil attack unfeasible. As just posing as another client, there is no way of telling which peers are currently participating, only in the end at the signing phase, in which if the attacker drops out, the attacker's inputs get banned. Therefore, if the attack has a selected target, it is hard to deduce at which point the target is participating in a mix unless the target can get the information in other ways (e.g. has infected the target's device, is sniffing on the network). Otherwise, the attacker would have to flood the mixing rounds every time they are created.

Each CoinJoin costs a certain amount; therefore, the cost goes up the more inputs or rounds the attacker wants to participate in. Blindly flooding all registrations is cost-ineffective and does not guarantee a hit since there could be other unknown peers already registered in the transaction. Even if the attack lowers the anonymity set, the attacker still has to guess the target's output.

4.3.2 Denial of service

Another attack an attacker could make is trying to deny the service of mixing to other clients. There are various ways malicious users can abort a round, for example:

- Attacker spending his input prematurely
- Refusal to sign the transaction in the signing phase

- Bob not registering an output
- Providing signed output from previous rounds

The most prominent method of defending against these attacks is banning all UTXOs provided by a malicious Alice and their possible future addresses up to a certain level for some time. This level is imposed to counter the possibility of an honest client receiving funds from a previously malicious attempt.

The attacker could still move the UTXOs to a different address to remove the ban. However, he would have to pay the transaction fees for each transaction, which could amount to a sum that would deter the attacker.

In the case of Bobs not posting their outputs, the peers enter a blame phase. A blame phase consists of all Alices revealing their signed outputs. The Alice that does not provide the outputs has its input UTXOs banned.

The fourth attack could follow the third one, where an input is registered but never provided. [31] The attacker can provide the signed output even with no input provided.

The developers have implemented several actions to defend against this sort of attack. The coordinator will ban the same blinded signatures from registering along with active outputs in the output registration phase.

The developers have also introduced a round hash of all inputs participating in the round. The new round will have different inputs, therefore a different round hash. The attacker would have to guess the correct round hash or get it in other ways. Otherwise, the coordinator will reject his output.

4.4 Man-in-the-middle attack

The Man-in-the-middle attack model refers to the attacker intercepting the traffic between the clients and the coordinator. He can either be passive and only eavesdrop or actively modify the participants' communication. Since the implementations work using the Tor network, it is challenging to accomplish them. Still, we can theoretically assume

that the attacker can intercept the messages or that the user does not use Tor.

Last year, Tor was attacked by the attackers controlling a significant portion (24%) of Tor exit nodes and forcing SSL stripping. [21] The attack led to some users of other sites losing their funds due to switching the output addresses in the transactions. Fortunately, coin theft should not be possible with CoinJoin, as we will discuss in section 4.6.

4.4.1 Deanonymizing a CoinJoin peer

CoinJoin heavily relies on the model of one user having two different identities that cannot be linked to each other, with which he communicates with the server. If the attacker can connect them by an IP address and decipher the encryption to get the actual data, the transaction becomes a false sense of privacy for the client.

Deanonymizing is achievable by intercepting the messages during the input and output registrations. Alice first sends the cleartext inputs and blinded outputs for the coordinator to sign during the input registration phase. Then the client creates another identity, Bob, to register the outputs sent in cleartext but are accompanied by the coordinator's signature.

If an attacker can link these two identities, he will gain knowledge of which inputs will correspond to which outputs. For example, suppose a client does not use Tor networking, and the attacker is sniffing packets outgoing from the client's device. In that case, the attacker can see both input and output registrations coming from one device. With these two requests, he already deanonymized the participating client.

Even if the attacker cannot link Alice and Bob, he can still succeed if the user decides to remix his coins after a successful CoinJoin. Suppose the attacker manages to intercept another Alice connection from the same machine. In that case, he could tell which outputs of the previous transactions were Alice's, as she would now register them as inputs for a new mix.

4.4.2 Denial of service

As the attacker can control the flow of the network, he could refuse any client's attempt to connect to the coordinator. However, this approach

is suspicious, and the attacked victim could realize that he is under attack.

A more covert way of denying a CoinJoin service would be modifying or replaying the coordinator messages. A stealthy denial of service attack would depend on the implementation.

With the Wasabi CoinJoin protocol, the attacker has two options. The Wasabi input registration can refuse a client without making the client think he is malicious. The attacker could do this by pretending that the input is blacklisted, as Wasabi has recently started blacklisting certain UTXOs, as we will discuss in subsection 6.2.1.

The other method is by tricking the client into thinking that the round has never started. The attacker can intercept a connection confirmation request and reply to Alice that the round has not started. The client will stay stuck in the connection confirmation phase, and the coordinator will later ban the client for not responding and start a new round.

In Samourai, there are also two possible covert ways to deny service to a client. We can do the first one during the initial input registration phase when the client waits for the coordinator to assign him a mix in the pool. By stopping the server notifications, the client will not be able to join a round and will be stuck in this phase.

The attack must be made multiple times, as we have learned that the Samourai client will try to restart the pool connection every hour or so. Therefore, it is vital to intercept the newly created connections. If the attacker can intercept every connection, this would be a preferred method.

Another method could happen after the client has already joined a mix and has confirmed the inputs. The client is waiting for a notification from the coordinator that allows peers to register outputs. Again, stopping these messages will trick the peer into thinking there still are not enough participants in the mix to start the round. Thus, the attack would seem legitimate.

4.5 Malicious coordinator

The malicious coordinator model represents an attacker that has gotten hold of the central coordinator server of the CoinJoin transaction,

which means that he has access to all data in the protocol and has the power to act on these data.

4.5.1 Sybil attack

A more feasible Sybil attack would involve the attacker being the coordinator itself. This way, the attacker could figure out which round the target is participating in and block all other peers except the attackers. This attack is possible since peers can be refused during the input registration phase. Refusal during the input registration does not reveal the fact that the tumbler could be malicious.

Nevertheless, other peers could have already registered before the target did. In this case, the attacker cannot refuse or drop other participants without getting noticed. Dropping participants in the connection confirmation phase can only be done if their input has been spent. Therefore, when a participant gets refused during that phase and does not spend his input, he can consider the coordinator malicious and will not use it anymore.

4.6 Coin theft

Coin theft is an act of manipulating a transaction's outputs in a way that is not evident to the user. CoinJoin participants have to provide their outputs, change outputs and signatures through the network to the coordinator. A man-in-the-middle attack could intercept these messages and modify the content, i.e. providing different outputs.

In the last phase of the CoinJoin protocols, clients receive the entire transaction to be able to review and sign them. This way, the user can verify whether the transaction consists of his inputs and his outputs, and only then he will sign it. If the output addresses were modified beforehand, the client would be able to spot them and refuse to sign the forged transaction.

After the transaction is signed, the attacker would not be able to modify the signed CoinJoin transaction due to the nature of digital signatures, even if the attacker was the coordinator himself. As once signed, changing the transaction would invalidate it.

Therefore, we assume that CoinJoin protocols are resistant to coin theft, and none of these attacker models can achieve such a goal.

5 Analysis of implementations

We have described the two CoinJoin implementations in detail from what information we could find online in their documentation. We can assess whether these claims and specifications hold true in their implementations. This chapter will describe the methodology, procedure and results of our assessment tests.

These tests and transactions were primarily made on Testnet, a separate instance of a Bitcoin blockchain. Testnet, as the name suggests, is used for testing and experimentation without risk to real funds or the main chain.

5.1 Setup

With all of these implementations supporting a PC environment (with a conditional exception of Samourai), we mainly used a custom-built desktop with AMD Ryzen 2600 with 32 GB of RAM running Ubuntu 20.04.4 LTS (Focal Fossa) to analyze and examine the underlying protocols of the applications.

We also used supporting machines to simulate other users of these protocols, as Testnet CoinJoins occur rarely compared to Mainnet ones. Testnet is mainly used by developers or bitcoin testers to experiment without using real money; thus, the amount of users is significantly lower than on Mainnet. Wasabi has a low limit of 2 participants to reach the goal of CoinJoin faster, while Whirlpool works only with five participants regardless of the blockchain used. These machines consisted of:

- **Lenovo Y700** - Ubuntu 20.04.4 LTS (Focal Fossa)
- **Samsung S10e** - Android 12
- **Samsung S9** - Andoird 10 (Rooted)
- **Huawei P8 Lite** - Android 6
- **Cubot X18** - Android 7
- **Virtual Google Pixel 2** - Android 8

In hindsight, a virtual emulated device (such as the Pixel 2) could replace the number of smartphones. We later discovered that there is no need for that many phones to simulate users, as we could use Whirlpool from a PC environment, needing only a pairing payload [46] from each wallet created.

The pairing payload consists of a mnemonic seed for the wallet so that the can PC client can operate with the wallet. However, the Whirlpool PC client can only participate in CoinJoins and does not support receiving and sending bitcoins. On the other hand, Sparrow Wallet, which is for desktops, supports both mixing and wallet operations.

Network-wise, we captured the traffic using WireShark and Mitmproxy on the main machine, where we executed most of the applications. The phones were connected to the home Wi-Fi router, except for the sniffed smartphone, which was connected to an access point created by the main machine.

Each Testnet wallet needed Testnet coins, at least a minimum amount for CoinJoins set by their implementation, and the mixing fees. Testnet coins can be acquired by mining blocks on the Testnet or using so-called "faucets", free services that provide Testnet coins with a specific limit. We used three, as all of them have time limits, to quickly fill out Testnet wallets with the amounts needed to proceed with our examination and analysis. These bitcoin faucets are Bitcoin Testnet Faucet [2], Testnet Faucet [38], and Coin Faucet [6]. Coin Faucet was handing out the most significant amount of testnet bitcoins, but as of the time of writing is currently empty. All Testnet bitcoins used will be returned to their respective faucets at the end of the research.

5.2 Methodology

This thesis aims to create an overview of existing protocols that increase Bitcoin transactions' privacy by using mixing techniques, particularly Wasabi CoinJoin and Samourai Whirlpool. We examined and collected the data sent and used during these protocols and compared them to the actual specifications from their creators to verify and compare their actual behaviour to that described in their specifications.

Firstly we downloaded the source codes of each implementation's last version from their repositories. Then we inspected the source code and identified the key components responsible for the CoinJoin protocol. We explored the options of activating the maximum log level. In some cases, where the logging lacked detail or volume, we had to include more logging messages to retrieve more information about the process if the captured traffic was not clear enough.

After building the source code, we started capturing network traffic using Wireshark, making sure to terminate any other process that could potentially clutter the resulting packet capture. Of course, eliminating all clutter would be more tedious as there might be discovery protocols or connectivity checks happening. As these were easy to filter out in the resulting capture, we did not attempt to disable them.

Only capturing the traffic is insufficient as the protocols are encrypted using TLS. We had to find a way to decrypt the sessions between our client and the coordinating server. We have tried the three most well-known methods:

- man-in-the-middle attack using mitmproxy
- preloading libraries
- in the case of Java, Java instrumentation API

We then ran the application with the highest log level enabled, set the option to communicate over the clearnet instead of Tor, and performed a CoinJoin. After the execution, we had a collection of application logs, captured network traffic, and dumped TLS keys for deciphering the traffic.

With the data collected, we proceeded to map the events in logs and the specification onto the captured network traffic to verify whether the reality meets expectations and check whether anything else could potentially leak information.

5.3 Wasabi

Wasabi is written in C#, and its source code resides on GitHub. [42] As of the time of writing (May 2022), Wasabi has released its new 2.0 version and a new protocol that is currently in beta, not yet working

on the Mainnet. The version we used (1.13) is still used for Mainnet transactions and currently resides in the backport branch of the repository.

5.3.1 Preparation

In order to get as much information as possible, we have to enable every logging message available in the code. In Wasabi, we can achieve maximum level logging by setting the minimum log level to trace in Logger.cs as it uses its custom logger. Despite the documentation in the code suggesting that trace level logs may include sensitive application data, Wasabi logging messages are lacking in frequency and volume.

The logging messages only announce that a particular phase has started or vaguely describe a part of the protocol, i.e. Bob has posted outputs: 1, which does not reveal much information about the protocol. The simplicity of the network communication counters the lack of descriptive logging messages. The communication is handled via HTTPS requests, clearly showing what is sent and for which phase.

5.3.2 Decrypting TLS traffic

As Wasabi uses OpenSSL, it calls on native OpenSSL libraries in the system. Thus we can try to preload our functions that can log the SSL session keys into a file. A simple LD_PRELOAD substitution does not work because the .NET runtime provides an open handle to dlsym, so dlsym needs to be wrapped.

We have used openssl-keylog [5], a library that wraps dlsym and SSL_new from which it logs the SSL session data into a log file. Specifically, it logs the client random from the ClientHello part of the TLS protocol and the cleartext master secret, allowing non-RSA SSL connections to be decrypted. [48]

Wireshark then uses these values to reconstruct the session keys and decrypt the traffic, allowing us to see what data was sent and received by Wasabi during the whole session.

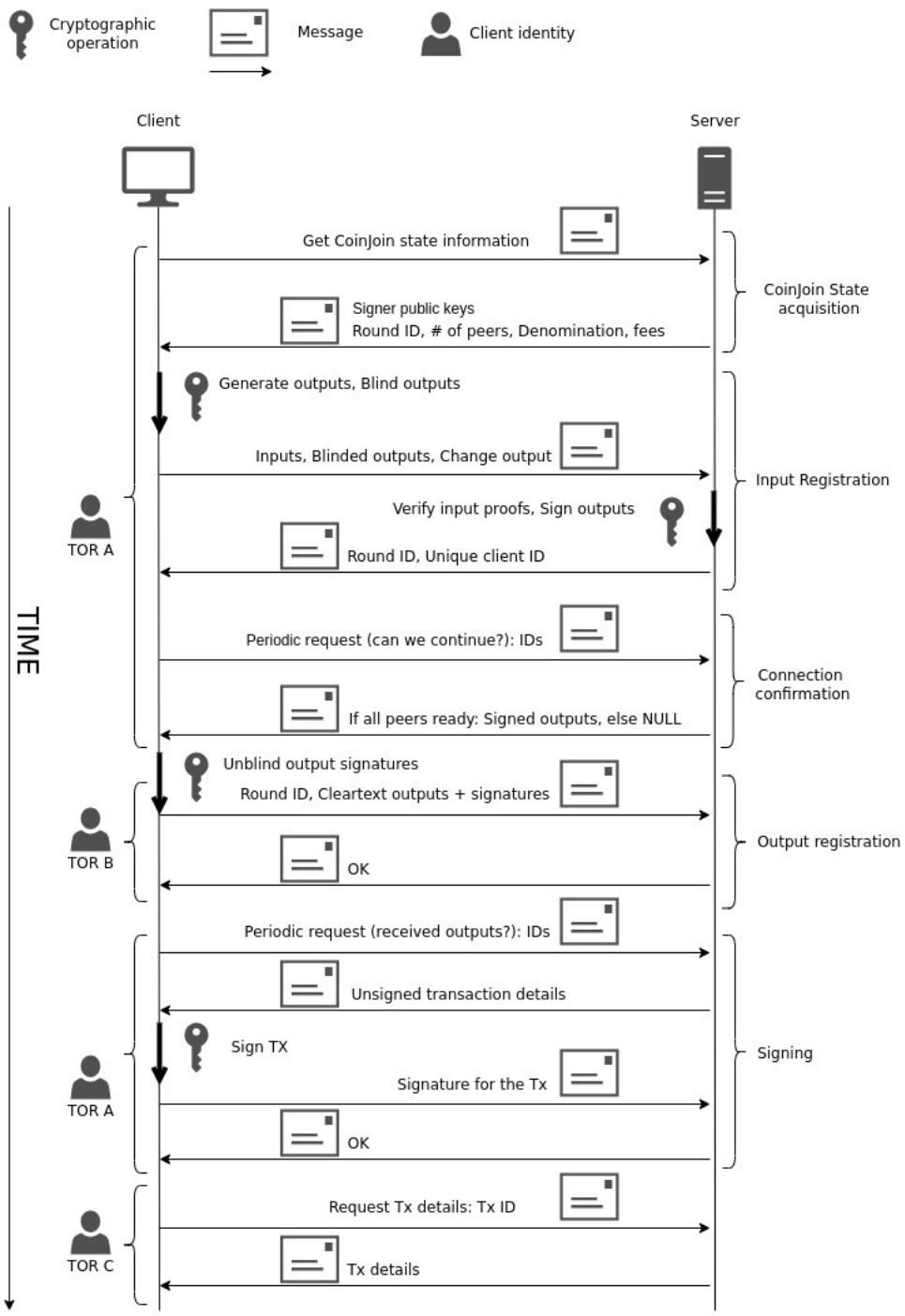

Figure 5.1: Wasabi CoinJoin Protocol

5.3.3 Protocol analysis

After decrypting the TLS protocol, we can see the underlying HTTP communication protocol between the client and the coordinating server. All these HTTP requests are documented via Swagger, and the API can be interactively tested on their website.[41] The entire protocol is preceded by an initialization phase in which the client asks the server about the CoinJoin states.

CoinJoin state acquisition

As previously stated, the client wants to know about the current state of CoinJoin happening on the server. CoinJoin state acquisition is made by sending a GET request on:

`wasabiwallet.co/api/v4/btc/chaumiancoinjoin/states/`

The server responds with a JSON object representing the state of the CoinJoin which contains basic information about the rounds and their states. The most notable information about the rounds includes:

- Round ID - 146582 in our case
- Phase - what kind of phase is the round currently in, which in our case, is input registration
- Denomination - what is the denomination of the current round - in our case, 0.0001
- Fees - both fee per input and fee per output and coordinator fee percentage
- Registered peer count - how many peers are already registered in the round
- Required peer count - how many peers are required for the round to continue
- Maximum input count per-peer - in our case, 7 (at the moment, it is a hardcoded constant)
- Registration timeout - in our case, 90

Input Registration

A new session with a client, "Alice", is created in this phase. We can observe the connection creation as a new TLS session handshake and then an HTTP POST to the API:

`api/v4/btc/chaumiancoinjoin/inputs/`

which contains these items:

- Round ID
- Blinded output - cryptographically hidden output
- CoinJoin Inputs - our coins that we put into the CoinJoin
- Proof for each input
- Cleartext change output address - in our case, it was not used

Here the implementation starts to skew from the specification.[40] At this point, according to the specification, the server should be sending a response with a unique ID and the signed blinded outputs. However, the server only returns these items:

- Round ID
- Unique ID - unique identifier for each client

The response signals the client that the round has entered the next phase of the protocol, connection confirmation.

Connection confirmation

The client periodically sends a POST without any JSON data as per the specification. Only their unique id and the round id are in the request URI. In our case, the request URI looked like this:

```
https://wasabiwallet.co/api/v4/btc/
   chaumiancoinjoin/confirmation?uniqueId=3
   df97788-b9e5-4978-851a-da53f7f2483c&roundId
   =146582
```

The specification tells us that the server should return a round hash of all the registered inputs, which is false. The actual response of the coordinator consists of a JSON object with two keys with values, which depend on the state of the conjoin.

- Blinded output signatures - in our case, NULL
- Current phase - in our case, 0

The server is withholding the signed blinded inputs for the time being. Following the stream of confirmation requests, we can arrive at the final confirmation request, which finally contains the blinded signature.

- Blinded output signature - now filled with the signed outputs
- Current phase - in our case, 1

The response also announces the end of the confirmation phase and the start of the output registration phase. We can deduce that the client is behaving in a state-oriented way when it waits until the server responds with the signatures and notifies the client to proceed with registering its outputs.

The delayed provision of the blinded signatures is a small deviation from the documentation. However, the existence of a round hash of all the inputs has been entirely scrapped by the developers. The removal was a decision in the later stages of the Wasabi Wallet and was implemented after they wrote the documentation. We will discuss the discrepancy in subsection 5.3.4describing the adherence to the specification.

Output registration

The client creates a new connection in the output registration phase, which could be seen in the capture with the new TLS handshake. The connection represents "Bob", which looks like an entirely new connection to the server. We can observe the creation of the new connection in the network traffic by a new TLS handshake. Bob sends a POST request to the server with the following JSON object containing:

- Round ID
- Cleartext output address
- Unblinded output signatures received at the end of the connection confirmation phase

The request is then proceeded by a server's 204 (No Content) status message. The output registration phase is finished for the client at this point.

Signing

After the output registration phase, Alice sends a GET request to the coordinator, with her unique id, to check the state round, just like in the connection confirmation phase. The server will respond with an entire raw unsigned transaction in hexadecimal format if the transaction is ready.

The raw transaction is followed by a POST request URI with her unique id and sending a JSON object with the signed transaction. The server then replies with a 204 HTTP code (No Content).

Post CoinJoin

After the CoinJoin, the client creates another new connection, a new Tor identity. The client directly asks the server about the completed CoinJoin transaction via a GET request on the coordinator along with the transaction id.

5.3.4 Specification adherence

Overall, Wasabi wallet implementation does not differ significantly from the ZeroLink framework or Wasabi documentation. However, there are slight deviations, which do not change functionality but affect the appearance of the traffic and participants' behaviours.

Firstly, the round hash supposed to be sent by the server, which identifies the round, was replaced in earlier stages of Wasabi. The change has happened from version 1.0 (the first stable release) to 1.1. [43] As the developers have revealed, the round hash has resulted

in being a limiting factor in various ways. The most prominent was that the coordinator was unable to launch a fallback round with the remaining number of participants when a round failed.

The reason for the round hash being introduced was described in a blog post on Adam Ficsor's website. [12] The round hash was implemented to defend against previously signed outputs described in subsection 4.3.2. The round hash functionality was replaced with the provision of a new signing key every round. Therefore, all previously signed outputs were signed with a different signing key, meaning that their signatures will not be valid in a new round, rendering the round hash redundant in that functionality.

However, this rids the client of the ability to verify the integrity of the received transaction. Specifically, the client cannot detect any changes in the inputs since the start of mixing, i.e. after the connection confirmation phase.

A malicious coordinator could potentially swap peers at his will without the peers noticing anything. However, swapping peers is possible for a malicious coordinator even during earlier stages of the protocol. Thus this only prolongs the window of opportunity when a client is unaware of such malicious activity.

Secondly, clients do not receive the signed blinded outputs at the end of the input registration phase as specified in the documentation. Instead, the server sends the signatures at the end of the connection confirmation phase. At that point, all Alices are connected to the coordinator and are ready to proceed.

To our knowledge, this change does not affect the CoinJoin protocol. There is no need to sign or serve signatures to Alices, who might drop out before all Alices connect and register their inputs. It increases the work needed for malicious clients to receive signatures for their outputs, making it a security improvement.

5.4 Samourai

Samourai Wallet is written in Java with some parts in Kotlin, and its source code resides on the Samourai team's own GitLab site.[33] The version we used for our research was 0.99. By default, the application on the Google Play store only works on Mainnet. In order to get the

wallet to operate on testnet, the application has to be side-loaded. Side-loading means installing the application from an APK straight from their GitLab site or compiling it from the source code.

5.4.1 Preparation

In order to create a successful Whirlpool, we need five separate clients that can be online at the same time. We achieved this number by using various smartphones to act as other unknown clients and only using one client that was being monitored, which was sufficient. In our case, the monitored client was on the Samsung S9 connected via USB to our desktop machine. The S9 had its activity logged and network communication sniffed by the desktop computer via ADB and Wireshark.

To decrypt traffic easier, we had also used Samourai's desktop Whirlpool client, specifically its command-line version. [45] It is a desktop application that allows Samourai users to use Whirlpool even without their phone app. To synchronize, the Whirlpool client needs a pairing payload, from which it can start using the Whirlpool to mix UTXOs without the need for the primary mobile app.

Samourai logs outperform Wasabi with their volume and description. Samourai debug logs precisely describe each step in the underlying protocol so developers and we can see exact phases and what is happening during the mix in the logs.

We can also achieve logging of the sent and received data on the samourai wallet by listening to verbose logs via ADB (Android Debug Bridge). In addition, the Samourai wallet uses the Android Stomp protocol implementation, which under verbose logging also writes both the received and sent stomp messages to the logs in cleartext.

For Whirlpool, we had to find the section of code handling the STOMP messages where we could insert logging messages. We get all incoming and outgoing traffic logged into the standard application log. The added code is in section A.1, named logging_diff. In our case, the added code is present in the Samourai Wallet by default but was not added to the Whirlpool client.

5.4.2 Decrypting TLS traffic

At first, we were unsuccessful with decrypting traffic coming to and from the S9. We searched for more straightforward methods, such as android apps that could decrypt traffic, and we found Remote PCAP. [11] Remote PCAP has proved unsuccessful as it only intercepts HTTPS traffic. Samourai uses HTTPS for wallet synchronization, but the Whirlpool protocol uses port 8081. We have attempted to include more ports, but the app's packet capture library, an integral part of the application, was not open-source.

Next, we have explored options with Java agents. [26] Java agents are a particular type of class that, by using the Java Instrumentation API, can intercept applications running on the JVM, modifying their bytecode. However, Java agents run on JVM, which Android does not use. Android has its version of instrumentation, [18] for which we did not find any implementation that dumps SSL secrets.

Where we found success after was Mimtmproxy [28] (Man in the middle proxy). Mitmproxy is a set of tools allowing the users to intercept network traffic and decrypt it. Mitmproxy can operate in different modes:

- Standard proxy
- Reverse proxy
- Socks5 proxy
- Transparent proxy
- Upstream proxy

After multiple attempts with different setups, we found the most simple, feasible one. We have rooted the S9, which allowed us to download and use an application called ProxyDroid. [20] ProxyDroid allows redirecting all traffic from an android device to a configured proxy.

We ran mitmproxy in socks5 mode on our desktop machine and routed every request to mitmproxy via ProxyDroid. Mitmproxy captures all the flows and decrypts them, unveiling the JSON objects sent between the client and the coordinating server.

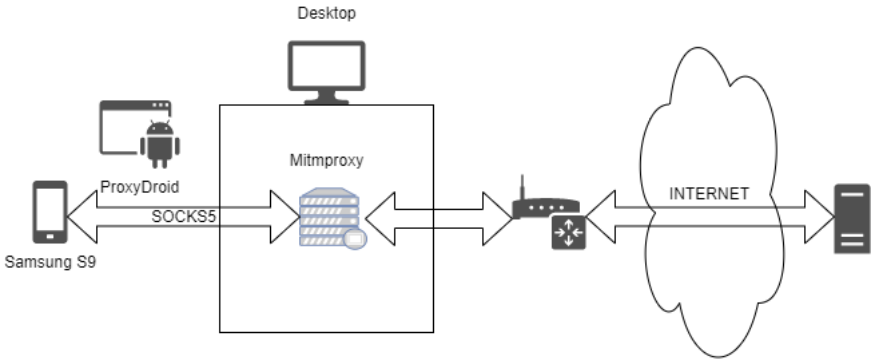

Figure 5.2: Setup for capturing Samourai network traffic on our smartphone. Smartphone traffic is tunneled through SOCKS5 to our mitmproxy that decrypts the traffic. Mitmproxy is operating in socks5 mode.

We also considered testing the desktop Whirlpool CLI client. The logs in the appendix section A.1 are captured from both the CLI client and the phone. We found it interesting to compare them if there are any differences, which we will discuss in subsection 5.4.4.

Following the same train of attempts for decrypting TLS traffic on the Samourai Wallet, we first considered using a Java agent for the Whirlpool client. Searching on the internet has provided us with jSSLKeyLog. [35] The jSSLKeyLog Java agent transforms the classes in Java SSL libraries to log the secret keys into a selected file. The final command for running the Whirlpool client looks like this:

```
java -javaagent:jSSLKeyLog.jar=whirlpool-
   client-cli/target/key.log -jar whirlpool-
   client-cli-0.10.13-run.jar --init --debug-
   client --debug | tee whirlpool_dump_log.
   txt
```

5.4.3 Protocol analysis

Unlike Wasabi, Whirlpool uses WebSocket protocol over HTTPS (WSS), with content transmitted via STOMP protocol. The mixing protocol is preceded by an initialization phase, where the client asks about the information about available pools.

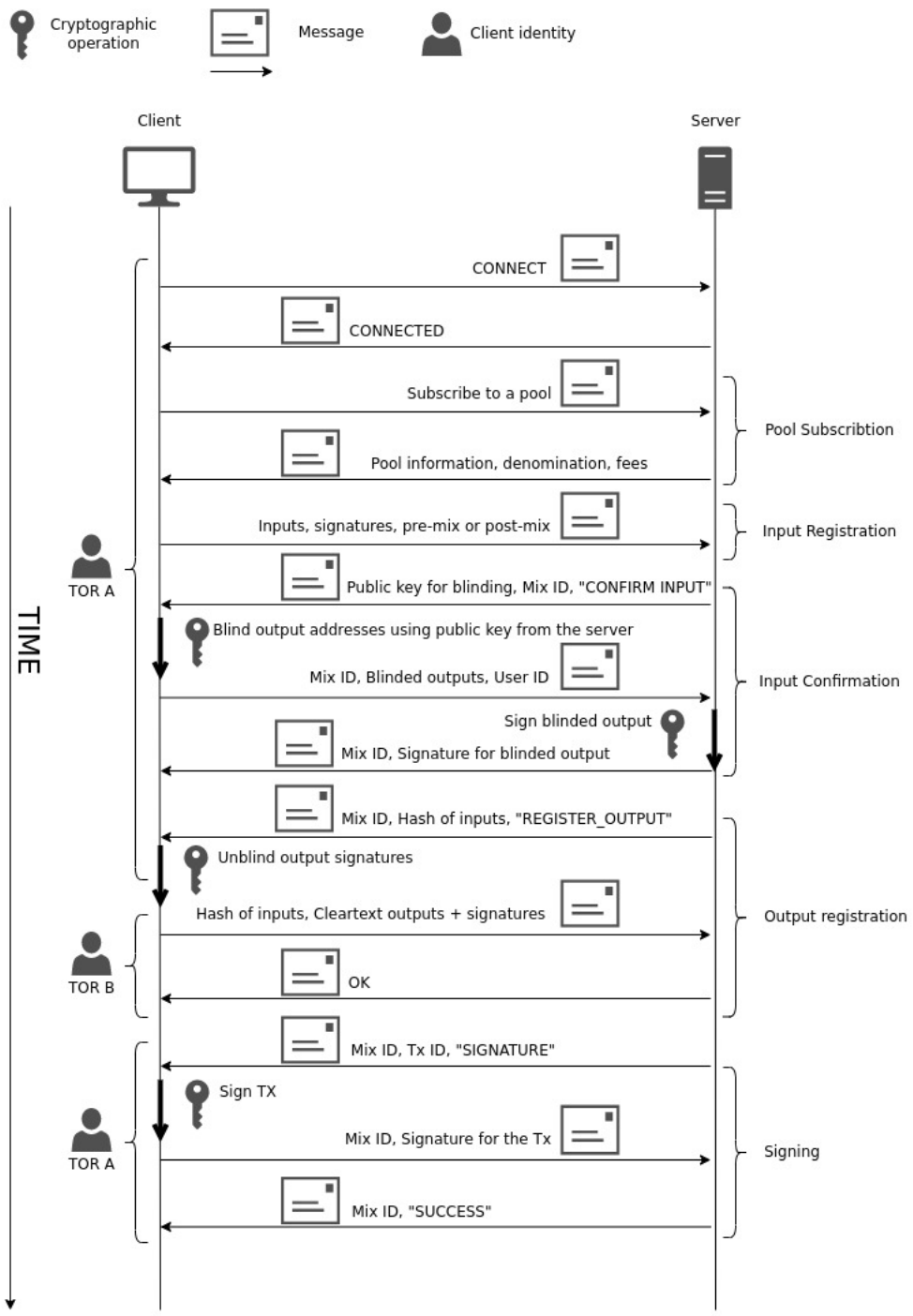

Figure 5.3: Samourai whirlpool protocol

Connection

The client first attempts to establish a connection request by executing an info request on the URL:

```
https://pool.whirl.mx:8081/ws/connect/info
```

If the connection is established, the server responds with various information about the server and communication protocol options, some of which are not important to us (e.g. Content-Type). The client then asks about the pool information that the user has chosen, in our case, the 0.001btc pool. The message also signals an intention of the client to subscribe to said pool. The server responds with:

- Network ID - in our case, "test"
- Denomination - pool size in satoshis
- mustMixBalance - minimum and maximum amount for an input to join (including fees)

Input registration

Now that the connection is established, the client can register its input. Registering input is done by sending a request to the server with a JSON object containing:

- Pool ID - 0.001btc
- Utxo hash - ID of the transaction that the input is from
- Utxo index- index of the output in said transaction
- signature - message pool id signed with UTXO's key
- liquidity - a boolean value indicating PRE-MIX or POST-MIX input

Input confirmation

When the server selects the client for the mix, the server replies with a notification, including two pieces of data.

- publicKey64 - a public key to use for blinding as key parameters
- Mix ID - unique identifier of the mix round to join
- status - in our case, "CONFIRM_INPUT"

After receiving the notification, the client uses the public key as a key parameter for blinding the generated output address and generates a server request to confirm the client's input. The request contains:

- Mix ID
- Blinded Output
- User hash - client identifier

As a reply, the server signs the blinded output and returns it to the client.

- Mix ID
- Signed output address

The client now has a confirmed input and knows he has joined a mix. The client must wait for the server to notify him to register his output once all the peers are found.

Output registration

The client receives a notification from the server, which informs him that the server is ready to receive outputs. This notification arrives in the form:

- Hash of inputs - this is the round hash of all mixed inputs
- Status - "REGISTER_OUTPUT"

- Mix ID

The client then creates a new WebSocket connection, which acts as the "Bob" entity in the ZeroLink scheme. Bob then registers the output for the client with a POST via HTTPS containing this JSON object:

- Hash of inputs
- Receiving address - cleartext output address
- Unblinded signature

Signing

After receiving the POST request from all Bobs, the server sends a notification to the clients containing the entire transaction for each Alice to check.

- Mix ID
- Status - "SIGNING"
- Transaction to verify and sign

The client then verifies the transaction. If the transaction is valid, the client sends back a signature:

- Mix ID
- Signed transaction

After which, the client gets a final reply from the server confirming the entire protocol and informing about its success:

- Mix ID
- status - "SUCCESS"

5.4.4 Specification adherence

Compared to Wasabi, the specifications are way more detailed about the architecture and stages of the protocol, down to the names of the key values. After examining the flow between the peer and the server, the communication strictly conforms to the written specification without deviation.

When comparing the captured packets from the desktop Whirlpool client with Wireshark, the captured flow from the smartphone via Mitmproxy and the specification, we cannot distinguish a difference. Both the android app and Whirlpool CLI client use the Whirlpool protocol, a separate project in Samourai's GitLab. Thus, their behaviour should be the same.

5.5 Sparrow Wallet

Sparrow Wallet is written in Java 16 and is developed by a single person. The version we used at the time of experimenting was 1.6.3.

5.5.1 Preparation

Sparrow Wallet uses Gradle to build itself, and the Sparrow developer provides scripts that run the wallet from the source. To set the wallet up for Testnet and enable logging, we only have to give command-line arguments to the prepared script:

```
./sparrow -n testnet -l TRACE
```

5.5.2 Decrypting TLS traffic

As Sparrow Wallet is written in Java 16, we could not use JSSLKeylog anymore. JSSLKeylog that we have used for Samourai could only dump SSL secrets up to Java 15.

The way jSSLKeylog worked is that it extracted the SSL secrets from the class SSLSecretDerivation, where it read the context private field. In Java 16 and onwards, this context private field was deleted. [14]

As the author noted in a Github issue, we could revert the change via instrumenting the method and the class to create the same private

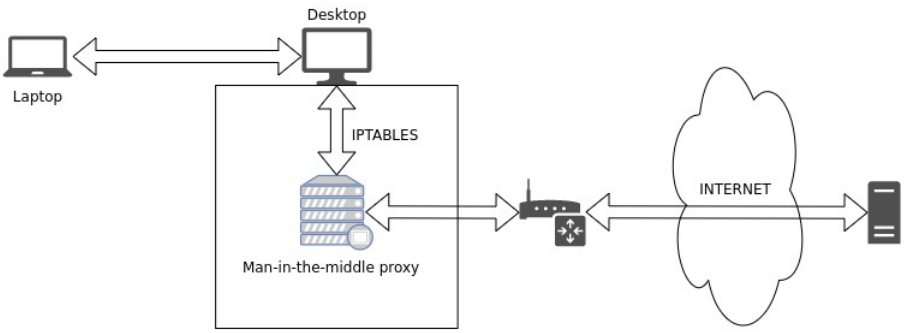

Figure 5.4: Sparrow Wallet proxy setup. Traffic from the laptop is redirected to the proxy via pre-routing tables on the gateway (desktop). Mitmproxy is operating in transparent mode.

field for context and the constructor to save the context to that private field. The altered method then allows the Java agent to work as intended.

The update allows us to dump the SSL secrets with the Java agent, but since the project gets executed via Gradle, we must modify the "build.gradle" file to add arguments to the JVM. We can add the arguments by inserting an array entry into the applicationDefaultJvmArgs variable in the script. The resulting patch looks like this:

```
-    applicationDefaultJvmArgs = ["-XX:+HeapDumpOnOutOfMemoryError",
+    applicationDefaultJvmArgs = ["-javaagent:jSSLKeyLog.jar=DUMP_PATH",
+                                 "-XX:+HeapDumpOnOutOfMemoryError",
```

However, reading WSS communication using Wireshark is tedious. Therefore, we have opted to use Mitmproxy again as it displays communication flows, extracting the relevant information from the packets. We have set up Sparrow Wallet on the laptop and connected it to the desktop via Wi-Fi. On the desktop machine, we allowed IPv4 forwarding and forwarded all incoming traffic to the wireless interface to mitmproxy's port. We appended this rule in the pre-routing table in iptables.

Sparrow Wallet has some man in the middle detection as detecting a change of certificates, but after clicking OK, it ignored the crafted certificate of our proxy.

5.5.3 Protocol

In essence, Sparrow Wallet uses Samourai's Whirlpool libraries. Therefore the traffic should be the same as in Samourai. We analyzed Trace logs, and after filtering the essential sections, we found logs being written by the Whirlpool protocol library. Regarding the CoinJoin, the logs were identical to Samourai logs.

The same occurred on both the Wireshark packet and Mitmproxy flow captures between the Whirlpool coordinator. Comparing them with the outputs of the Samourai Wallet, we have found no difference. The result is expected, as they are using the same coordinator. To communicate with the coordinator, they have to stick to the Whirlpool protocol. To our knowledge and findings, we assume that Sparrow Wallet uses Samourai's libraries and does not deviate in any way.

5.6 Mainnet assessment

We have also performed these tests on the Mainnet to discern any differences between the two blockchain networks. For these purposes, we have only used Wasabi and Samourai wallets. We will not disclose everything about the transactions to not potentially lower the anonymity sets for peers participating in the mixes with us.

In the case of Wasabi, we have participated in one CoinJoin with the minimum amount needed to join. The CoinJoin was a success, with the anonymity set being over 40. The network traffic and the logs have shown no discrepancies compared to the Testnet transactions, with the expected exceptions such as different addresses, transaction size and the Bitcoin network.

As for Samourai, we used the setup with the rooted Samsung S9 and mitmproxy. We kept the phone and the proxy running for one week and tried to get many mixes as possible. In hindsight, this was not ideal as the application was prone to freezing and lagging and did not automatically queue in inputs to the mixer. The phone had to be restarted in two instances as it froze, leaving us unable to control it.

In the end, we managed to participate in 25 mixes, including remixes. The first mixing of pre-mix inputs happened within a few hours of starting the experiment. We could not join any remixes until four days later. We believe the long waiting times were due to a

large pool of post-mix UTXOs waiting to be mixed. Generally, every Whirpool mix usually introduces four new post-mix UTXOs. Therefore, the number of post-mix UTXOs waiting for a remix strictly increases with every CoinJoin. Furthermore, every post-mix UTXO has to wait for enough pre-mix UTXOs to start mixing. Therefore it also depends on the frequency of fresh bitcoins incoming to the mixer.

After analyzing the Whirlpool transactions, we confirmed that the transactions on the Mainnet and Testnet do not show any discrepancies except the ones we accounted for, such as different addresses.

6 Comparison

Even though both Wasabi and Samourai aim to achieve the same goal, their protocols and values differ. In this chapter, we will discuss their significant differences. We will cover the protocol differences as well as the wallets, which also impact the security and privacy of the user's bitcoins.

Firstly, we collected the fundamental differences in Table 6.1 to provide a quick overview of the distinctions. The following sections will cover each difference individually in more detail.

Table 6.1: Overview of differences between Wasabi and Whirlpool

Protocol	Wasabi	Whirlpool
Centralized synchronisation?	Yes, but may use own hosted node	Yes, but may use own hosted node
Trustless syncrhonisation?	Yes	Not by default
# of participants	Up to 100	5
Minimum coins	0.1 BTC	0.001 BTC + fees
Platforms	Desktop	Android + Desktop
Networks over Tor	Yes	Yes
Blacklists UTXOs	Yes	No
Communication	HTTPS	WSS with STOMP
Fees	0.003% * Anon set	Flat for each pool

6.1 CoinJoin differences

This section will take a glance at the differences in the mixing protocols. Both Wasabi and Samourai are forks of the ZeroLink framework. Thus they are similar in functionality and architecture. Nevertheless, where Wasabi tried to follow it closely, Samourai took a different approach regarding the architecture and parameters of the protocol. The principles, however, stay the same.

6.1.1 Amount of participants

The amount of participants in a CoinJoin transaction affects the anonymity sets of the participants. Generally, the more participants in a transaction, the bigger the anonymity set gets. Wasabi and Samourai chose different approaches to this situation.

Wasabi offers a single transaction with up to one hundred participants. A large transaction ensures a quick way to boost the coin anonymity set by a large number. If the desired anonymity set defined by the user is not met, Wasabi will remix the outputs of a previous CoinJoin to meet the required set.

Samourai's whirlpool transactions have a constant amount of five participants. A higher anonymity set is reached by remixing the outputs free of charge until the desired anonymity set is met. These remixes are paid for by new UTXOs entering the mix from the pre-mix pool.

6.1.2 Minimal amount to join

In order to join a CoinJoin, the participant must have a minimum amount of bitcoins so that he can mix it with other peers. Wasabi wallet allows only one minimum amount of bitcoins to join, which is around 0.1 BTC at the time of writing and has been so for quite some time.

The reason behind this number is most prominently DoS protection and reducing the percentage of the input being paid to the miners. The miner fees stay the same regardless of the number of bitcoins sent. The actual fee is calculated depending on the size of the actual transaction. Mixing smaller amounts would mean that the client pays a larger percentage of their input to the miners.

Samourai, on the other hand, offers pools with various output amounts that have different minimum amounts to join. These pools include:

- 0.001 Pool - Minimum funds of 0.001 BTC + fees
- 0.01 Pool - Minimum funds of 0.01 BTC + fees
- 0.05 Pool - Minimum funds of 0.05 BTC + fees

- 0.5 Pool - Minimum funds of 0.5 BTC + fees

These pools have a wider range of minimum funds, which means they are potentially more accessible to users, specifically on days when bitcoin prices are higher than in previous years.

6.1.3 Mixing fees

In all cases of CoinJoin, the participant has to pay a fee. Specifically, there are two fees that a peer has to pay in order to be a part of a CoinJoin. The first fee goes to the coordinator of the CoinJoin, i.e. Wasabi or Samourai. The second fee is the actual mining fee that has to be paid to the miners to confirm a transaction. We will take a look into both fees in detail. The way Wasabi and Samourai implementation differ is not only in the amount of the fees but also in how are the fees paid.

Coordinator fees

The Wasabi coordinator and miner fees are paid for during the mix, where the coordinator address is one of the outputs in the transaction. The coordinator has to be paid a fee of 0.003% times the anonymity set. For example, if the anonymity set of a coin is 50, then a client pays 0.003% * 50, which equals 0.15% of the input intended to mix. There are also edge cases where the peer does not pay the coordinator or pays more.

For example, the smallest registrant to a round will never pay a coordinator fee. Also, when the peer is remixing, he cannot pay the full coordinator fee with his input. On the other hand, if the input is larger than the minimum and the change amount leftover is too small, it is added to the coordinator fee. Currently, the minimum change amount to be paid out is 0.3% of the base denomination (0.1 BTC).

There is also a possibility of receiving more than what the peer has put into the mix. When network fees go down between the start of the round and its end, the difference is split evenly between outputs.

In Whirlpool, the fees to the coordinator are "pre-paid", as first, you have to divide coins into a pre-mix pool. The transaction into the pre-mix pool, called *tx0*, contains a coordinator output address to

which the fees are paid. The miner fees for the CoinJoin are also paid during the mix.

As with the minimum funds to enter, Samourai fees differ based on what pool the client has chosen.

- 0.001 Pool - Fee 0.00005 BTC - 5% of pool denomination
- 0.01 Pool - Fee: 0.000425 BTC - 4.25% of pool denomination
- 0.05 Pool - Fee: 0.0014875 BTC - 2.975% of pool denomination
- 0.5 Pool - Fee: 0.014875 BTC - 2.975% of pool denomination

These fees are constant, regardless of the amount of input. For example, if we decide to mix 0.002 BTC in a 0.001 BTC pool, the coordinator fee is still 0.00005 BTC. Therefore, by mixing 0.002 BTC, we would only pay 2.5% of our input to the coordinator, which is still significantly higher than Wasabi's percentage cut. The percentage lowers, the higher the amount is. However, one should also consider mining fees, as they increase with the size of the transaction.

After one successful Whirlpool, the anonymity set of post-mix outputs is five. These post-mix UTXOs can be later remixed for free until the UTXO reaches the desired anonymity set.

Comparing fees with Wasabi, Samourai could get larger anonymity set for a smaller fee. However, it takes time as post-mix outputs do not get remixed as often.

Miner fees

In the Wasabi protocol, the participant pays the miner fees for the number of his inputs and outputs, including change. For example, if we decide to mix one input slightly above the minimum amount, we would be paying for one input and two outputs of a transaction.

Samourai uses a different architecture, where the user first splits his input and pays the fee in *tx0* and then pays for the mining fee of the mixing transaction itself.

Therefore, to mix one denomination, the client pays miner fees for tx0 in full and then a part of the mixing transaction, depending on the cycle priority. Higher priority means that pre-mix inputs pay more, and more post-mix inputs are present.

6.1.4 Communication protocols

A slight difference can be found on the network level. Each implementation uses a different network protocol to communicate with the backend server.

Wasabi uses HTTPS, a half-duplex protocol. Thus, wasabi clients have to constantly poll the coordinator for changes in the stages of the CoinJoin. There is no way for the server to notify the client when the phase of the protocol changes. Thus, Wasabi CoinJoin can generate an extensive network footprint due to regular periodic polls on the server.

Samourai, on the other hand, uses WebSockets over HTTPS (WSS) to create a connection and uses the STOMP protocol that defines the form of the data transferred. WSS is a full-duplex protocol. Therefore the server can notify the clients of Whirlpool phase changes. The notification scheme creates a smaller network footprint than Wasabi, as there is no constant polling.

6.2 Wallet differences

There are several differences between the implementations, which are not strictly CoinJoin protocol-oriented. These would include platform availability, politics and other outside factors.

6.2.1 Blacklisting

As of March 2022, Wasabi wallet has started refusing certain blacklisted UTXOs from registering to their coordinator's CoinJoins. [52] As zkSNACK's coordinator is the only one officially used by Wasabi wallet, effectively meaning that every UTXO registering to CoinJoin done by Wasabi wallet is under the company's control. The change has been done proactively to elude possible future legislation regarding mixing services. However, as Wasabi wallet is open-source, anyone can run a competing coordinator without blacklisting in place.

One of the Wasabi developers had stated in the company's defense that it was done to prevent the company from getting in trouble when illegal funds were mixed using their mixing service. [8] He also has stated that it is the company's right to refuse to do business with

specific customers. As of the time of writing, we have yet to receive an informative official statement from zkSNACKs.

The decision to ban UTXOs has been met with public backlash. However, as of the time of writing, the volumes of Bitcoin flowing through Wasabi CoinJoins have not significantly dropped compared to before the post. The change does not affect the implementation directly, and it still provides anonymization of coins.

Samourai took a stance against the banning of UTXOs in their tweet [34] as an answer to Wasabi blacklists. Samourai state that mixers like Samourai and Wasabi are message passers, not money transmitters, and therefore should not conform to such measures. With this stance, Samourai servers do not blacklist any UTXOs incoming to Whirlpool services.

6.2.2 Privacy

As explained in the third chapter, Wasabi queries the server for BIP 158 block filters to recognize which blocks contain addresses generated by the wallet. When a block filter hits, it does not matter if it is a true match or a false positive. The block will then be downloaded from a random Bitcoin P2P node with new identities. Then the block can be verified if it contains the transactions that are important to the wallet. If it was a false positive, it only adds to the privacy of the protocol, where a sniffing attacker could be thrown off.

Of course, the protocol still depends on the central backend server, as a malicious one could serve incorrect filters, resulting in the wallet not finding corresponding transactions. However, it is possible to host a local bitcoin node and connect the Wasabi Wallet to it.

Samourai users, by default, give the central server the wallet's extended public key (xpub), which is a key that can be used to derive child public keys. Anyone who knows a wallet's xpub can see every past and future transaction the owner makes. After receiving the xpub, the server will send the wallet the related blocks which contain its addresses. The user has to trust the Samourai developers or maintainers to not misuses these collected xpubs and not reveal them to third parties.

The need for implicit trust could be countered with the user hosting their node created by Samourai developers, Dojo. This approach is

under the assumption that the user has the technical knowledge and resources to run a custom full node. Everyday users who only use samourai on their smartphones might not use these measures nor know about the inherent trust in the developers.

However, Samourai Wallet is not the only wallet using Whirlpool services. Users can opt to use Sparrow Wallet instead. Instead of sending a server our xpub, Sparrow Wallet connects to public electrum servers and only asks for blocks containing transactions with specific addresses. While not as private as Wasabi, it is generally more private than Samourai as it only reveals a single address and not the entire xpub to the synchronizing server.

Even while trying to prevent giving the backend server our xpub key, we have to realize that potentially many do not do the same. Even if we use the Dojo, we will be mixing our UTXOs with users not using Dojo. Therefore, developers and server maintainers can deduce the inputs and outputs they cannot identify as ours.

6.2.3 Platform availability

Wasabi is a PC only platform. Therefore a user trying to mix their UTXOs has to have a desktop PC running the whole time until the CoinJoin completes.

Samourai is an Android-only application; therefore, mixing from the wallet is only available from users' smartphones, which have to be running and online. Due to the need for remixes to increase the anonymity set to a suitable amount, Whirlpool users would need to have their smartphones online and connected to the servers for a prolonged time, which is undesirable.

Therefore, we can either use Samourai's Whirlpool PC client or Sparrow Wallet to mix UTXOs, increasing the platform availability by PCs as well. We conclude that Whirlpool supports a broader range of platforms than Wasabi.

6.2.4 Code contribution

Development activity is a possible value that could be considered a quality identifier. We have scanned the repository contributions for all wallets using repo-contrib-graph [16]. The graphs show us how often

are these wallets maintained or developed, with more contributions having an overall positive value. When downloaded, the repo-contrib-graph did not work as expected. We had to fix two issues to make it run properly. We also created a fork and a pull request with the fixes for the creators. [30]

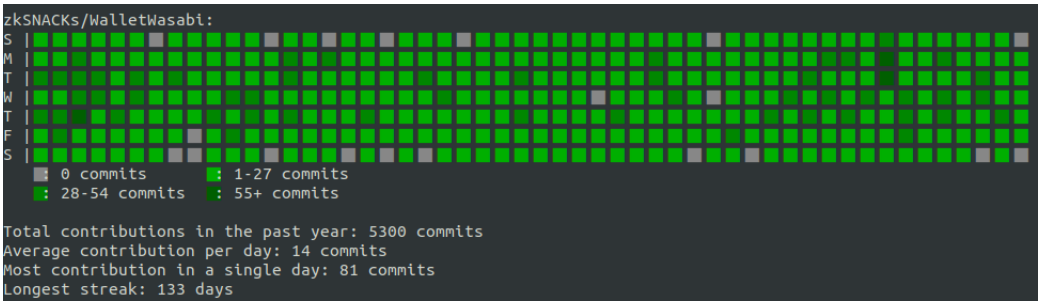

Figure 6.1: Wasabi repository contribution graph

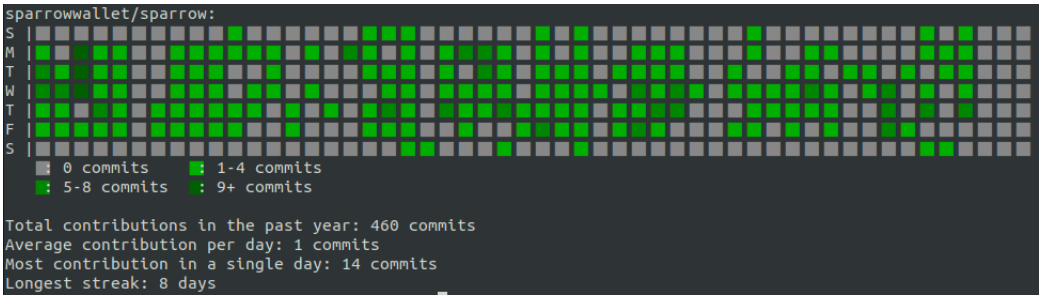

Figure 6.2: Sparrow repository contribution graph

However, we could not generate such a graph with Samourai as their codebase is on their own GitLab instance. We could get the contribution graph from GitLab analytics on the repository itself. We limited the graph only to show the last year of activity.

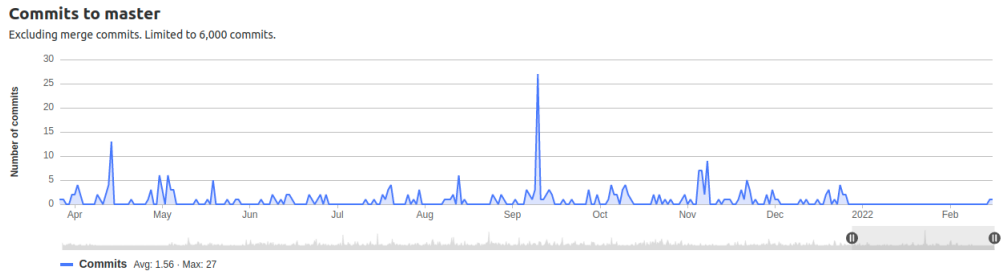

Figure 6.3: Samourai repository contribution graph

From these graphs, we see that Wasabi has the most active repository. However, it is also because Wasabi has released beta version of Wasabi 2.0, increasing the number of contributions.

6.3 Bitcoin volumes mixed

The creator of Wasabi Wallet, Adam Ficsor, has created a tool called Dumplings [10] for scanning and creating statistics comparing Samourai, Wasabi and other CoinJoin-looking transactions. It is done by scanning the blockchain and applying specific heuristics to identify a transaction based on its characteristics.

For Wasabi, these heuristics are divided into two sets. Firstly, until block 610,000, there was a constant Wasabi coordinator address, so we could solely look for it, and we knew it was a Wasabi Coinjoin. After the specified date, the Wasabi coordinator address is not reused, so we have to use different heuristics:

- Is native Segwit Only
- Has at least ten equal outputs
- Has usually more inputs than outputs
- The most frequent equal outputs must be almost the base denomination.
- It is very likely it has at leas0.t 2 unique outputs (coordinator and at least one change output)

For Samurai (named Samuri in the results), we do not have a set coordinator address, so we use these heuristics all the time:

- Is native Segwit only
- Has five inputs
- Has five outputs
- Outputs are always equal
- Constant pool sizes
- at least one and at most three post-mix inputs
- the remaining two to four inputs are between the pool size and pool size +0.0011BTC (Those pay the fees)

There is also a third heuristic for *Otheri* in the results. Transactions under the Otheri category look like CoinJoin transactions (have a common denomination output, multiple inputs and outputs) but do not fall under Wasabi or Samourai.

Regarding the majorly known JoinMarket CoinJoins, the author states that every JoinMarket transaction is an "Other CoinJoin Transaction", but not every "Other CoinJoin Transaction" is a JoinMarket transaction.

In Figure 6.4, we can see a stacked area graph of monthly bitcoin volumes being mixed by the CoinJoin protocol. As we can see, Wasabi CoinJoins are the most prominent CoinJoins from the collected transactions using Adam Ficsor's heuristics.

One of the better indicators of the popularity and spread of the software from the available statistics is the number of unmixed bitcoins CoinJoined per month. We can see a stacked graph of the volume of fresh bitcoins incoming to CoinJoins in Figure 6.5.

Since we know the total amount of bitcoins mixed and the fresh bitcoins incoming into CoinJoins, we can derive a metric that can tell us how many times was the amount of the fresh bitcoins remixed. It gives us a general idea, not the exact average number of remixes of fresh inputs, but of all inputs up to that date.

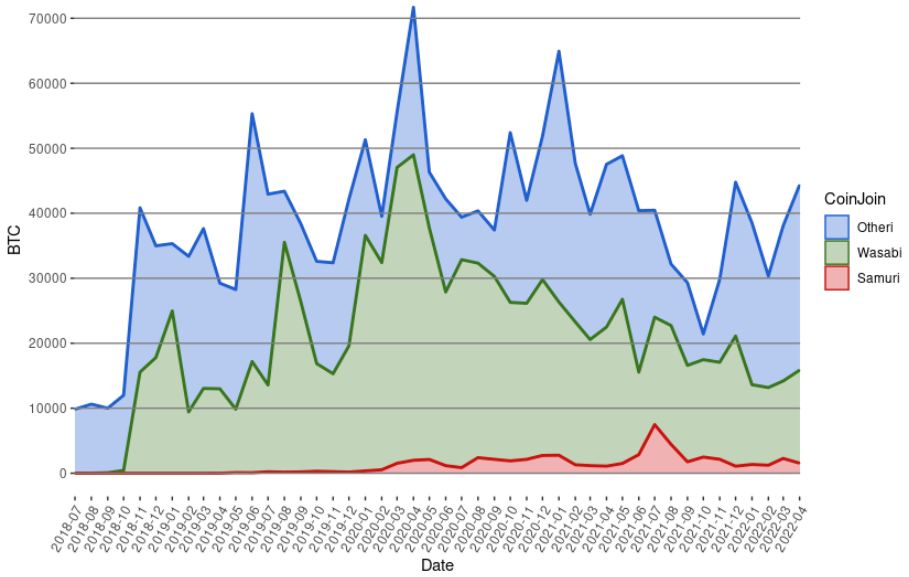

Figure 6.4: Stacked area graph of monthly volume mixed by CoinJoins

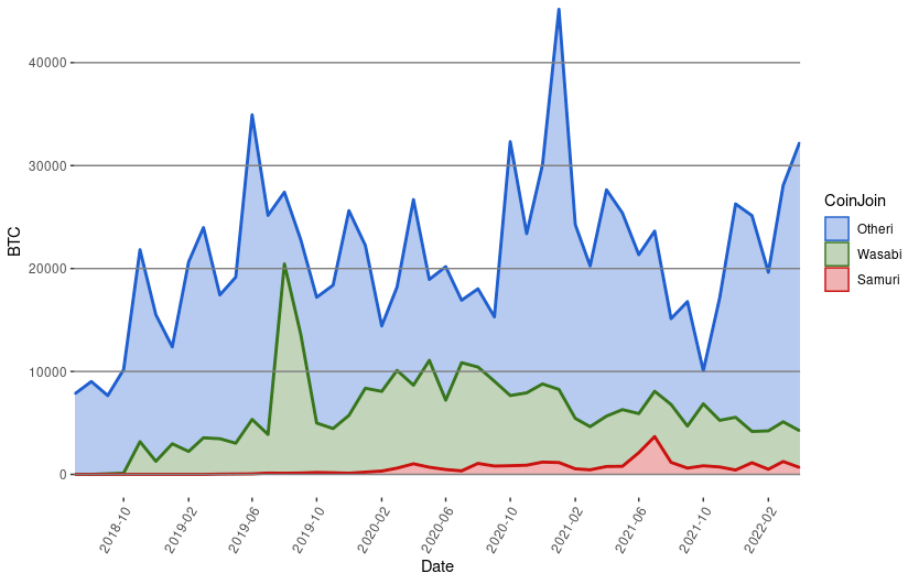

Figure 6.5: Stacked area graph of fresh bitcoins incoming to CoinJoins

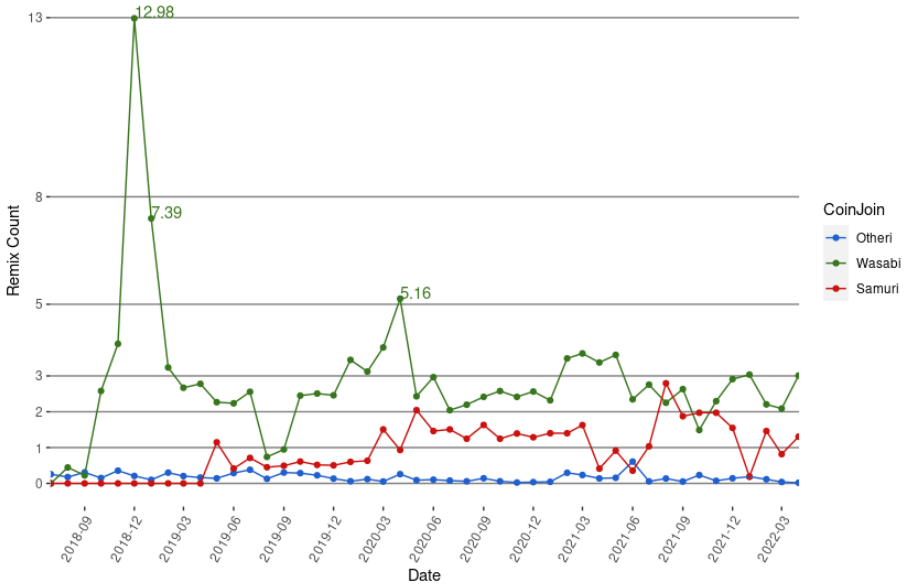

Figure 6.6: Average number of times the amount of fresh bitcoins has been remixed per month

We can see the results of this derived metric in Figure 6.6. As Wasabi usually has the highest volume of mixed bitcoins but lower fresh bitcoin volumes, we see that majority of volumes passing through wasabi are remixing.

For Samourai mixes, the amount of remixes is surprisingly low, even though they offer free remixes. The low amount of remixes could be explained by the fact that there are few pre-mix inputs in the Whirlpool, as remixing requires these fresh bitcoins.

Other transactions have a high amount of fresh bitcoins compared to the total amount of bitcoins mixes. Therefore, the remixed volume is almost zero.

7 Conclusion

The goal of the first part of the thesis was to find and study the state-of-the-art decentralized mixing services in the Bitcoin network. We focused on implementations of the ZeroLink framework and its mixing protocol, the CoinJoin. We introduced three different privacy-focused wallets: Wasabi, Sparrow and Samourai.

The second part discussed the possible security issues of CoinJoin. We considered three goals of the attacker from varying attacker models, from a blockchain observer to a malicious coordinator, describing possible attacks unique to each attacker model. We also mentioned a public advisory regarding Wasabi Wallet, written by the OXT research but denied by the Wasabi developers.

The key parts of the thesis are the actual assessment tests. We set up all three Bitcoin wallets and analyzed their behaviour by reading logs and sniffing network traffic. We then studied the difference between their specifications and their actual behaviours.

Samourai and Sparrow Wallet implementation did not differ from its specification from what we have analyzed. The specified architecture is well described and detailed down to the names of variables, which made it simple to compare the behaviour and the specification.

Regarding Wasabi, the implementation differs from its specification. To our knowledge, these changes do not compromise the overall security of CoinJoin. However, the deprecation of the round hash prevents the user from verifying if the inputs in the final transaction have changed after the mixing starts. We believe that this discrepancy between the implementation and the specification is due to not updating the specification after making changes in the software.

The last part of the thesis compares Wasabi and Whirlpool implementations from various standpoints. Concerning the protocol, the principles of the two ZeroLink fork implementations are the same. Samourai's Whirlpool differs in the architecture and parameters of the CoinJoin, such as having different pools or a fixed participant amount.

Moreover, by default, Wasabi assumes that the coordinator must not be trusted and that everyone knows what the coordinator knows. In Samourai's or Sparrow's case, there needs to be an inherent trust in the centralized server. In the case of Samourai, the clients have to

trust the developers fully, as they disclose their xpubs which reveal all their future transactions.

7.1 Future work

The main goal of this thesis was to compare the implementations of CoinJoin protocols to their specifications written by the developers. Therefore, we have only explained or briefly gone through the actual security of these implementations or the features that these wallets offer.

Addressing the security questions and features of the protocols and wallets in more detail could be a potential area of further research on CoinJoin transactions. Particularly making attempts to attack the implementations through various attack vectors as described in the theoretical part.

Another possible point of interest and research would be the now released beta of Wasabi 2.0 with the new WabiSabi protocol, which replaces the Chaumian blinding with keyed-verification anonymous credentials (KVAC) schemes. [13] Wasabi 2.0 also promises new features such as a flat 0.3% CoinJoin fee with free remixes. [51]

Literatur

[1] Sarah Meiklejohn et al. *A Fistful of Bitcoins: Characterizing Payments Among Men with No Names*. URL: https://cseweb.ucsd.edu/~smeiklejohn/files/imc13.pdf.

[2] *Bitcoin Testnet Faucet*. URL: https://bitcoinfaucet.uo1.net.

[3] *Blender.io*. URL: https://blendar.io/.

[4] Blender.io. *Blind signature*. URL: https://en.wikipedia.org/wiki/Blind_signature.

[5] Will Brown. *OpenSSL keylog*. URL: https://github.com/wpbrown/openssl-keylog.

[6] *Coinfaucet*. URL: https://coinfaucet.eu/en/btc-testnet/.

[7] *CoinJoin Sudoku*. 2014. URL: https://www.coinjoinsudoku.com/.

[8] *Developer's opinion about zkSNACK's blacklisting*. URL: https://twitter.com/BTCparadigm/status/1503154471278460932.

[9] Nicolas Dorier. *A Simple Payjoin Proposal*. URL: https://github.com/bitcoin/bips/blob/master/bip-0078.mediawiki.

[10] *Dumplings*. URL: https://github.com/nopara73/Dumplings.

[11] Andrey Egorov. "PCAP Remote Tutorial". In: (). URL: https://egorovandreyrm.com/pcap-remote-tutorial/.

[12] Adam Ficsor. *Upcoming Wasabi wallet hard fork*. URL: https://nopara73.medium.com/upcoming-wasabi-wallet-hard-fork-609f271d9c41.

[13] Adam Ficsor u. a. *WabiSabi: Centrally Coordinated CoinJoins with Variable Amounts*. URL: https://github.com/zkSNACKs/WabiSabi/releases/latest/download/WabiSabi.pdf.

[14] *JDK SSLSecretDerivation context field removal commit*. URL: https://github.com/openjdk/jdk/commit/dcf63f857872489a082be970438b5365c7a83b4a#diff-82a1cca2f87c6128fff95bf14b19922b91b18e1228d3ff519366b1fc5bfa3f69L7

[15] kefir500. *GitHub Download Stats*. URL: https://github.com/kefir500/ghstats.

[16] kylemacey. *repo-contrib-graph*. URL: https://github.com/kylemacey/repo-contrib-graph.

[17] ErgoBTC LaurentMT. *An Analysis and Disclosure Regarding the Deterministic Nature of the Wasabi Wallet CoinJoin Algorithm*. URL: https://research.oxt.me/alerts/2020/08/21/Wasabi-Wallet.

[18] Google LLC. "Instrumentation". In: (). URL: https://developer.android.com/reference/android/app/Instrumentation.

[19] Streetside Development LLC. *Samourai Wallet*. URL: https://samouraiwallet.com/.

[20] Max Lv. *ProxyDroid*. URL: https://play.google.com/store/apps/details?id=org.proxydroid&hl=en&gl=US.

[21] Riccardo Masutti. *Wasabi Wallet and Tor SSL stripping attacks*. URL: https://blog.wasabiwallet.io/wasabi-wallet-tor-ssl-stripping-attack/.

[22] Gregory Maxwell. *CoinJoin: Bitcoin privacy for the real world*. URL: https://bitcointalk.org/index.php?topic=279249.0.

[23] Monero. *BTC - XMR Atomic swaps are live!* URL: https://mobile.twitter.com/monero/status/1427671823597047811.

[24] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (). URL: https://bitcoin.org/bitcoin.pdf.

[25] Lontivero Nopara73. *zkSnacks response to OXT Research claims*. URL: https://archive.ph/tKdNL.

[26] Oracle. "Java Instrument package". In: (). URL: https://docs.oracle.com/javase/1.5.0/docs/api/java/lang/instrument/package-summary.html.

[27] *OXT Blockchain explorer*. URL: https://oxt.me/.

[28] Mitmproxy Project. "Mitmproxy introduction". In: (). URL: https://docs.mitmproxy.org/stable/.

[29] Stelios Rammos. *What are Coinjoins and how do they improve Bitcoin privacy?* URL: https://cryptotesters.com/blog/what-are-coinjoins-and-how-do-they-improve-bitcoin-privacy.

[30] *Repo-contrib-graph pull request*. URL: https://github.com/kylemacey/repo-contrib-graph/pull/27.

[31] *Reusing unprovided output*. URL: https://github.com/nopara73/ZeroLink/issues/51.

[32] *Samourai Documentation*. URL: https://docs.samourai.io/.

[33] *Samourai GitLab*. URL: https://code.samourai.io/wallet/samourai-wallet-android.

[34] *Samourai response to zkSNACK blacklisting*. URL: https://twitter.com/SamouraiWallet/status/1503389170672226308.
[35] Michael Schierl. "jSSLKeyLog - Java Agent Library to log SSL session keys to a file for Wireshark". In: (). URL: https://jsslkeylog.github.io/.
[36] Claus-Peter Schnorr. "Journal of Cryptology 4. Efficient Signature Generation by Smart Cards". In: S. 161–174.
[37] SparrowWallet. *Sparrow*. URL: https://sparrowwallet.com/.
[38] *Testnet Faucet*. URL: https://testnet-faucet.mempool.co.
[39] Wasabi Wallet. *Wasabi Wallet 2.0 Testnet Version is now available!* URL: https://twitter.com/wasabiwallet/status/1498762641812172807.
[40] Wasabi. "Wasabi CoinJoin". In: (). URL: https://docs.wasabiwallet.io/using-wasabi/CoinJoin.html.
[41] *Wasabi API*. URL: https://wasabiwallet.io/swagger/index.html.
[42] *Wasabi GitHub*. URL: https://github.com/zkSNACKs/WalletWasabi.
[43] *Wasabi round hash removal commit*. URL: https://github.com/zkSNACKs/WalletWasabi/pull/1006/commits/36d3b141dd6fcc00d37830a34b58e41
[44] *Wasabi synchronisation*.
[45] *Whirlpool Cli*. URL: https://code.samourai.io/whirlpool/whirlpool-client-cli.
[46] *Whirlpool Desktop*. URL: https://docs.samourai.io/whirlpool/desktop.
[47] Wikipedia. *Monero*. URL: https://en.wikipedia.org/wiki/Monero.
[48] Wikipedia. *Transport Layer Security*. URL: https://en.wikipedia.org/wiki/Transport_Layer_Security#Protocol_details.
[49] Pieter Wuille. *Hierarchical Deterministic Wallets*. URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.
[50] *Zerolink protocol*. URL: https://github.com/nopara73/ZeroLink.
[51] zkSnacks. *Wasabi 2.0 tweet*. URL: https://twitter.com/wasabiwallet/status/1480576794252222475.
[52] *zkSNACKs starts blacklisting*. URL: https://twitter.com/wasabiwallet/status/1503091503207432193.
[53] *zkSNACKs, unfairly private*. URL: https://zksnacks.com/.

A Attachments

A.1 Network traffic captures and logs

The file *captures.zip* contains logs and captured traffic of each implementation we tested. Namely:

- Wasabi Wallet
- Samourai Wallet
- Whirlpool CLI
- Sparrow Wallet

An excerpt from the logs and traffic can be found in the *communication* folder. Each file in the *communication* folder corresponds to a particular message exchange during the CoinJoin. There is a corresponding excerpt of the network traffic and the logged information in each file.

For Wasabi and Whirlpool CLI, the captured traffic is in pcapng format. The file can be opened with WireShark, and to decrypt the traffic, one needs to select the *keys.log* file as the master secret file for TLS preferences. The Whirlpool CLI folder also contains a git diff file changes needed to log incoming and outgoing traffic.

Samourai and Sparrow traffic was captured using mitmproxy. To read the traffic capture, one has to run mitmproxy with $-r$ option and give it the name of the file.

```
./mitmproxy -r PATH_TO_FILE
```

A.2 CoinJoin volumes CSV files and R scripts

The *volume_flows.zip* contains CSV files with derived data from Dumplings and R scripts that generate graphs in section 6.3.