TU WIEN Informatics

Adaptor Signature Based Atomic Swaps Between Bitcoin and a Mimblewimble Based Cryptocurrency

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering & Internet Computing

by

Jakob Abfalter, BSc
Registration Number 01126889

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Matteo Maffei
Assistance: Dr. Pedro Moreno-Sanchez

Vienna, 6th April, 2020

Jakob Abfalter	Matteo Maffei

Erklärung zur Verfassung der Arbeit

Jakob Abfalter, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. April 2020

Jakob Abfalter

Acknowledgements

Enter your text here.

Abstract

Enter your text here.

Contents

CHAPTER 1

Introduction

Pedro: We need to discuss a structure for the introduction. Proposal:
- Introduce why coin exchanges are interesting
- Explain why atomic swaps protocols (e.g., one could use a trusted server for this and problem solved, right?)
- Why coin exchanges between Bitcoin and Mimblewimble?
- Why what you are proposing in this thesis is challenging?
- What are the main contributions of these thesis?
- What do you think is an interesting future research direction?

Mimblewimble The Mimblewimble protocol was introduced in 2016 by an anonymous entity named Jedusor, Tom Elvis [**jedusor2016mimblewimble**]. The author's name, as well as the protocols name, are references to the Harry Potter franchise. [1] In Harry Potter, Mimblewimble is a tongue-typing curse which reflects the goal of the protocol's design, which is improving the user's privacy. Later, Andrew Poelstra took up the ideas from the original writing and published his understanding of the protocol in his paper [**poelstra2016mimblewimble**]. The protocol gained increasing interest in the community and was implemented in the Grin [2] and Beam [3] Cryptocurrencies, which both launched in early 2019. In the same year, two papers were published, which successfully defined and proved security properties for Mimblewimble [**fuchsbauer2019aggregate**, **betarte2019towards**].

Pedro: I would not add a line break at the end of each paragraph. The template should do that

[1] `https://harrypotter.fandom.com/wiki/Tongue-Tying_Curse`
[2] `https://grin.mw/`
[3] `https://beam.mw/`

Pedro: If you are going to compare to Bitcoin, you need to introduce Bitcoin before

Compared to Bitcoin, there are some differences in Mimblewimble:

- Use of Pedersen commitments instead of plaintext transaction values

  Pedro: The reader does not know what Pedersen commitments are at this point. Perhaps say transaction values are hidden from a blockchain observer while this is not the case in Bitcoin

- No addresses. Coin ownership is given by the knowledge of the opening of the coins Pedersen commitment.

  Pedro: This is also unclear. Could one see the commitment as the "address" in Mimblewimble? Perhaps you want to say that there is no scripting language supported?

- Spend outputs are purged from the ledger such that only unspent transaction outputs remain.

- No scripting features.

Pedro: Use "we" for contributions that you do in the thesis and "they" for parts that are borrowed from other works

Pedro: An intuition of these two terms is required here

Pedro: another sentence that shows that you need to explain before how Bitcoin works (the basics)

By utilizing Pedersen commitments in the transactions, we hide the amounts transferred in a transaction, improving the systems user privacy, but also requiring additional range proofs, attesting to the fact that actual amounts transferred are in between a valid range. Not having any addresses enables transaction merging and transaction cut through, which we will explain in section 3.3.3. However, this comes with the consequence that building transactions require active interaction between the sender and receiver, which is different than in constructions more similar to Bitcoin, where a sender can transfer funds to any address without requiring active participation by the receiver. Through transaction merging and cut-through and some further protocol features, which we will see later in this section, we gain the third mentioned property of being able to delete transaction outputs from the Blockchain, which have already been spent before. This ongoing purging in the Blockchain makes it particularly space-efficient as the space required by the ledger only grows in the number of UTXOs, in contrast to Bitcoin, in which space requirement increases with the number of overall mined transactions. Saving space is especially relevant for Cryptocurrencies employing confidential transactions because the size of the range proofs attached to outputs can be significant.

Pedro: What comes next is hard to read. It requires better organization: Advantages of Mimblewimble are: (i) .., (ii)...; Disadvantages are: (i)..., (ii),...).

Another advantage of this property is that new nodes joining the system do not have to verify the whole history of the Blockchain to validate the current state, making it much easier to join the network. Another limitation of Mimblewimble- based Cryptocurrencies is that at least the current construction does not allow scripts, such as they are available in Bitcoin or similar systems. Transaction validity is given solely by a single valid signature plus the balancedness of inputs and outputs. This shortcoming makes it challenging to realize concepts such as multi signatures or conditional transactions which are required for Atomic Swap protocols. However, as we will see in 3.4 there are ways we can still construct the necessary transactions by merely relying on cryptographic primitives [**fuchsbauer2019aggregate**].

CHAPTER 2

Motivation & Objectives

TODO

CHAPTER 3

Preliminaries

Pedro: Although not strictly required, IMO it is nice to have some text here introducing what the reader should expect in the rest of the section. For instance:
In this section, we first introduce the notation and definitions used hereby in this thesis. Then, we ...... Finally, we introduce.....

3.1 General Notation and Definitions

Notation We first define the general notation used in the following chapters to formalize procedures and protocols. Let $\mathbb{G}$ denote a cyclic group of prime order p and $\mathbb{Z}_p$ the ring of integers modulo p with identity element 1_p. $\mathbb{Z}_p^*$ is $\mathbb{Z}_p \setminus \{0\}$. $g,\ h$ are adjacent generators in $\mathbb{G}$, where adjacent means the discrete logarithm of h in regards to g is not known. Exponentiation stands for repeated application of the group operation. We define the group operation between two curve points as $g^a\ \cdot\ g^{g^b}\ \stackrel{?}{=}\ g^{a\ +\ b}$.

Pedro: I think macro \ was broken here. I have updated to use \ instead. Please check that this is what you expected

Pedro: We normally do not use the tilde to add spaces in math mode

Definition 3.1 (Hard Relation[**aumayr2020bitcoinchannels**])**.** Given a language $L_R\ :=\ \{A \mid \exists a \text{ s.t. } (A, a) \in R\}$ then the relation R is considered hard if the following three properties hold:

1. $\mathsf{genRel}((1^n))$ is a PPT sampling algorithm which outputs a statement/witness of the form $(A, a) \in R$.
2. Relation R is poly-time decidable.
3. For all PPT adversaries $\mathcal{A}$ the probability of finding a given A is negligible.

Pedro: I would include these two relations below as your own definitions because I imagine that you would like to refer to them afterwards in the thesis

In this thesis we find two types of hard relations:

1. The output of a secure hash function (as defined in 3.3) and it's input $(I, \mathsf{H}(I))$.
2. The discrete logarithm x of g^x in the group $\mathbb{G}$.

Pedro: Link to paper/book where you got this definition from is missing

Pedro: What is valid here? You have not defined it before

Definition 3.2 (Signature Scheme)**.** A valid Signature Scheme must provide three procedures:

Pedro: I would write this sentence as: A signature scheme Φ is a tuple of algorithms $(\mathsf{keyGen},\ \mathsf{sign},\ \mathsf{verf})$ defined as follows:

$$\Phi = (\mathsf{keyGen},\ \mathsf{sign},\ \mathsf{verf})$$

Pedro: write the API of the algorithms in bullet points

$(sk, pk) \leftarrow \mathsf{keyGen}(1^n)$ takes as input a security parameter 1^n and outputs a keypair (sk, pk), consisting of a secret key sk and a public key pk, whereas the secret key has to be kept private and the public key is shared with other parties. sk can be used together with a message m to call the $\mathsf{sign}(sk,\ m)$ procedure to create a signature σ over the message m. Parties knowing pk can then test the validity of the signature by calling $\mathsf{verf}(pk,\ \sigma,\ m)$ with the same message m. The procedure will only output 1 if the message was indeed signed with the correct secret key sk of pk and therefore proves the possesion of sk by the signer . A valid signature scheme have to fulfill two security properties

Pedro: "proving that the sender had the sk" is a property that no all signature schemes may have

Pedro: Choose one unforgeability form, the one that you require later in the thesis

- Correctness: For all messages m and valid keypairs (sk, pk) the following must hold $\mathsf{verf}(pk,\ \mathsf{sign}(sk,\ m),\ m) \stackrel{?}{=} 1$

Pedro: Except with negligible probability

- Unforgeability: Note that there are different levels of Unforgability: [**goldwasser1988digital**]
  - Universal Forgery: The ability to forge signatures for any message.
  - Selective Forgery: The ability to fogre signatures for messages of the adversary's choice.
  - Existential Forgery: The ability to forge a valid signature / message pair not previously known to the adversary.

Pedro: nice that you have defined the three properties. I would keep the one that you need later

Definition 3.3 (Cryptographic Hash Function)**.** A cryptographic hash function $\mathsf{H}$is defined as $\mathsf{H}(I) \to \{0,1\}^n$ for some fixed number n and some input I. A secure hashing function has to fullfill the following security properties: [**al2011cryptographic**]

Pedro: minor thing: we normally use capital H

- Collision-Resistence (CR): Collision-Resistance means that it is computationally infeasible to find two inputs I_1 and I_2 such that $\mathsf{H}(I_1) := \mathsf{H}(I_2)$ with $I_1 \neq I_2$.
- Pre-image Resistence (Pre): In a hash function H that fulfills Pre-image Resistance it is infeasible to recover the original input I from its hash output $\mathsf{H}(I)$. If this security property is achieved, the hash function is said to be non-invertible.
- 2nd Pre-image Resistence (Sec): This property is similar to Collision-Resistance and is sometimes referred to as *Weak Collision-Resistance*. Given such a hash function H and an input I, it should be infeasible to find a different input I^* such that $I \neq I^*$ and $\mathsf{H}(I) \stackrel{?}{=} \mathsf{H}(I^*)$.

Pedro: I think here the Open operation of the commitment is missing, which you need later for the binding property

Definition 3.4 (Commitment Scheme [**bunz2018bulletproofs**])**.** A cryptographic Commitment Scheme COM is defined by a pair of functions $(\mathsf{keyGen}(1^n),\ \mathsf{commit}(I,k))$. keyGen is the setup procedure, it takes as input a security parameter 1^n and outputs public parameters PP. Depending on PP we define a input space $\mathbb{I}_{PP}$, a randomness space $\mathbb{K}_{PP}$ and a commitment space $\mathbb{C}_{PP}$.

The function commit takes an arbitrary input $I \in \mathbb{I}_{PP}$, and a random value $k \in \mathbb{K}_{PP}$ and generates an output$C \in \mathbb{C}_{PP}$.

Secure commitments must fullfill the *Binding* and *Hiding* security properties:

- *Binding:* If a Commitment Scheme is binding it must hold that for all PPT adversaries $\mathcal{A}$ given a valid input $I \in \mathbb{I}_{PP}$ and randomness $k \in \mathbb{K}_{PP}$ the probabilty of finding a $I^* \neq I$ and a k^* with $\mathsf{commit}(I,k) \stackrel{?}{=} \mathsf{commit}(I^*,k^*)$ is negligible.
- *Hiding:* For a PPT adversary $\mathcal{A}$, commitment inputs $I \in \mathbb{I}_{PP},\ k \in \mathbb{K}_{PP}$ and a commitment output $C := \mathsf{commit}(I,k,)$ the probabilty of the adversary choosing the correct input $\{I, I^*\}$ must not be higher then $\frac{1}{2} + \mathsf{negl}(P)$.

Pedro: this definition seems wrong to me? Where does I^* come from?

Pedro: Add reference from where you took this definition. You may want to add that it is an "Additive" Homomorphic Commitment

Definition 3.5 (Homomorphic Commitment)**.** If a Commitment Scheme as defined in 3.4 is homomorphic then the following must hold

$$\mathsf{commit}(I_1, k_1) \cdot \mathsf{commit}(I_2, k_2) \stackrel{?}{=} \mathsf{commit}(I_1 + I_2, k_1 + k_2)$$

First, a Pedersen Commitment is an instance of Commitment Scheme as in Def 3.4; Second it has the homomorphic property as in 3.5. Clarify that, for instance, by explaining exactly how the algorithms are implemented, as you did below.

Definition 3.6 (Pedersen Commitment)**.** A Pedersen Commitment is an instantiation of a Homomoprhic Commitment Scheme as defined in 3.5:

$$\mathbb{C}_{PP} := \mathbb{G}$$

of order p, $\mathbb{I}_{PP}$, $\mathbb{K}_{PP} := \mathbb{Z}_p$. the procedures ($\mathsf{keyGen}, \mathsf{commit}$) are then instantiated as:

$$\mathsf{keyGen}(1^n) := g, h \leftarrow\$ \mathbb{G}$$

$$\mathsf{commit}(I, k) := g^k h^I$$

3.2 Bitcoin

3.2.1 Bitcoin Transaction Protocol

3.2.2 Bitcoin Scaling and Layer Two Solutions

3.3 Privacy-enhancing Cryptocurrencies

3.3.1 Zero Knowledge Proofs

3.3.2 Range Proofs

Definition 3.7 (Rangeproof System)**.** A Rangeproofs system $\Pi[COM]$ with regards to a homomorphic commitment scheme COM consists of a tupel of functions ($\mathsf{ranPrfSetup}, \mathsf{ranPrf}, \mathsf{vrfRanPrf}$).

- $(lb, ub) \leftarrow \mathsf{ranPrfSetup}(1^n, i, j)$: The rangeproof setup algorithm takes as input a security paramter 1^n as well as two numbers i and j which are treated as exponents of 2 to define the lower and upper bound of the rangeproof protocol.
- $\pi \leftarrow \mathsf{ranPrf}(C, v, r)$: The proof algorithm is a DPT function which takes as input a commitment C a value v and a blinding factor r. It will output a proof π attesting to the statement that the value v of commitment C is in between the range $\langle lb, ub \rangle$ as defined during the $\mathsf{ranPrfSetup}$ function.
- $\{1, 0\} \leftarrow \mathsf{vrfRanPrf}(\pi, C)$: The proof verification algorithm is a DPT function which verifies the validity of the proof π with regards to the commitment C. It will output 1 upon a successfull verification or 0 otherwise.

Definition 3.8 (Multiparty Rangeproof System)**.** A Multiparty Rangeproof Sytem $\Pi_{mp}[COM]$ with regards to a homomorphic commitment scheme COM is an extension of the regular Rangeproof Sytem with the following distributed protocol $\mathsf{dRanPrf}$.

- $\pi \leftarrow \mathsf{dRanPrf}\langle(C, v, r_A), (C, v, r_B)\rangle$: The distributed proof protocol allows two parties Alice and Bob, each owning a share of the commitment C to cooperate in order to produce a valid range proof π without a party learning the blinding factor share from the other party.

For MP proofs [**klinec2020privacy**]

3.3.3 Mimblewimble

In this section we will outline the fundamental properties of the protocols employed in Mimblewimble which are relevant for the thesis and particularily the construction of the Atomic Swap protocol defined in 5.

Transaction Structure

Pedro: I think that throughout this section, you have nice explanations of the different parts of the transaction. It would be also possible to add definitions for the different things that you use

- For two adjacent elliptic curve generators g and h a coin in Mimblewimble is a tuple of the form $(\mathcal{C}, \pi)$, where $\mathcal{C} := g^v \cdot h^k$ a Pedersen Commitment [**pedersen1991non**] to the value v with blinding factor k. π is a range proof attesting to the statement that v is in a valid range in zero-knowledge.

  Pedro: not sure whether the point below is required

Pedro: you might want to specify what range is used here. Also I rewrote some part, so please check.

- As already pointed out, there are now addresses in Mimblewimble. Ownership of a coin is equivalent to the knowledge of its opening, so the blinding factor takes the role of the secret key.
- A transaction consists of $\mathcal{C}_{inp} := (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ input coins and $\mathcal{C}_{out} := (\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ output coins.

A transaction is considered valid iff $\sum v'_i - \sum v_i \stackrel{?}{=} 0$ so the sum of all input values has to be 0. (Not taking transaction fees into account)

Pedro: doesn't need to check the range proofs as well?

From that we can derive the following equation:

$$\sum \mathcal{C}_{out} - \sum \mathcal{C}_{inp} := \sum (h^{v'_i} \cdot g^{k'_i}) - \sum (h^{v_i} \cdot g^{k_i})$$

So if we assume that a transaction is valid then we are left with the following so called excess value:

$$\mathcal{E} \;:=\; g^{(\sum k_i' \;-\; \sum k_i)}$$

Pedro: You mean knowledge of the exponent of $\mathcal{E}$?

Knowledge of the opening of all coins and the validity of the transaction implies knowledge of $\mathcal{E}$. Directly revealing the opening to $\mathcal{E}$ would leak too much information, an adversary knowing the openings for input coins and all but one output coin, could easily calculate the unknown opening given $\mathcal{E}$. Therefore knowledge of $\mathcal{E}$ instead is proven by providing a valid signature for $\mathcal{E}$ as public key. Coinbase transactions (transactions creating new money as part of a miners reward) additionally include the newly minted money as supply s in the excess equation:

$$\mathcal{E} \;:=\; g^{(\sum k_i' \;-\; \sum k_i)} \;-\; h^s$$

Finally a Mimblewimble transaction is of form:

$$tx \;:=\; (s,\; \mathcal{C}_{inp},\; \mathcal{C}_{out},\; K) \text{ with } K \;:=\; (\{\pi\},\; \{\mathcal{E}\},\; \{\sigma\})$$

Pedro: the $\mathcal{E}$ is a single value? or a set?

where s is the transaction supply amount, $\mathcal{C}_{inp}$ is the list of input coins, $\mathcal{C}_{out}$ is the list of output coins and K is the transaction Kernel. The Kernel consists of $\{\pi\}$ which is a list of all output coin range proofs, $\{\mathcal{E}\}$ a list of excess values and finally $\{\sigma\}$ a list of signatures [**fuchsbauer2019aggregate**].

Transaction Merging

An essential property of the Mimblewimble protocol is that two transactions can easily be merged into one, which is essentially a non-interactive version of the CoinJoin protocol on Bitcoin [**maxwell2013coinjoin**] Assume we have the following two transactions:

$$tx_0 \;:=\; (s_0,\; \mathcal{C}^0_{inp},\; \mathcal{C}^0_{out},\; (\{\pi_0\},\; \{\mathcal{E}_0\},\; \{\sigma_0\}))$$

$$tx_1 \;:=\; (s_1,\; \mathcal{C}^1_{inp},\; \mathcal{C}^1_{out},\; (\{\pi_1\},\; \{\mathcal{E}_1\},\; \{\sigma_1\}))$$

Then we can build a single merged transaction:

$$tx_m \;:=\; (s_0 \;+\; s_1,\; \mathcal{C}^0_{inp} \;||\; \mathcal{C}^1_{inp},\; \mathcal{C}^0_{out} \;||\; \mathcal{C}^1_{out},\; (\{\pi_0\} \;||\; \{\pi_1\}),\; \{\mathcal{E}_0\} \;||\; \{\mathcal{E}_1\},\; \{\sigma_0\} \;||\; \{\sigma_1\})$$

We can easily deduce that if tx_0 and tx_1 are valid, it follows that tx_m also has to be valid: If tx_0 and tx_1 are valid that means $\mathcal{C}^0_{inp} \;-\; \mathcal{C}^0_{out} \;-\; h^{s_0} \;:=\; \mathcal{E}_0$, $\{\pi_0\}$ contains valid range proofs for the outputs $\mathcal{C}^0_{out}$ and $\{\sigma_0\}$ contains a valid signature to $\mathcal{E}_0 \;-\; h^{s_0}$ as public key, the same must hold for tx_1.

By the rules of arithmetic it then must also hold that

$$\mathcal{C}^0_{inp} \;||\; \mathcal{C}^1_{inp} \;-\; \mathcal{C}^0_{out} \;||\; \mathcal{C}^1_{out} \;-\; h^{s_0 \;+\; s_1} \;:=\; \mathcal{E}_0 \;\cdot\; \mathcal{E}_1,\; \{\pi_0\} \;||\; \{\pi_1\}$$

must contain valid range proofs for the output coins and $\{\sigma_0\} \;||\; \{\sigma_1\}$ must contain valid signatures to the respective Excess points, which makes tx_m a valid transaction.

Subset Problem

Pedro: I think the content below is not fully clear yet. If needed for the rest, we need to clarify (e.g., add an example?)

A subtle problem arises with the way transactions are merged in Mimblewimble. From the shown construction, it is possible to reconstruct the original separate transactions from the merged one, which can be a privacy issue. Given a set of inputs, outputs, and kernels, a subset of these will recombine to reconstruct one of the valid transaction which were aggregated since Kernel Excess values are not combined. (which would invalidate the signatures and therefore break the security of the system) This problem has been mitigated in Cryptocurrencies implementing the protocol by including an additional variable in the Kernel, called offset value. The offset is randomly chosen and needs to be added back to the Excess values to verify the sum of the commitments to zero.

$$\sum \mathcal{C}_{out} \; - \; \sum \mathcal{C}_{inp} \; - \; h^s \; := \; \mathcal{E}^o$$

Every time two transactions are merged, the offset values are combined into a single value. If offsets are picked truly randomly, and the possible range of values is broad enough, the probability of recovering the uncombined offsets from a merged one becomes negligible, making it infeasable to recover original transactions from a merged one [**poelstra2016mimblewimble**].

Cut Through

From the way transactions are merged together, we can now learn how to purge spent outputs securely. Let's assume $\mathcal{C}_i$ appears as an output in tx_0 and as an input in tx_1, which are being merged. Remembering the equation for transaction balancedness, $\mathcal{C}_{inp} \; - \; \mathcal{C}_{out} \; := \; \mathcal{E}$ if $\mathcal{C}_i$ appears both in the inputs and outputs, and we erase it on both sides, the equation will still hold. Therefore every time a transaction spends an output, it can be virtually forgotten to improve transaction unlinkability as well as yielding saving space.

Pedro: This requires further explanation and maybe an example?

The Ledger

Pedro: Do we need this subsection?

The ledger of the Mimblewimble protocol itself is a transaction of the already discussed form. Initially, the ledger starts empty, and transactions are added and aggregated recursively.

- Only transactions in which input coins are contained in the output coins of the ledger will be valid.
- The supply of the ledger is the sum of the supplies of all transactions added so far. Therefore we can easily read the total circulating supply from the ledger state.

- Due to cut through, the input coin list of the ledger is always empty, and the output list is the set of UTXOs.

Transaction Building

As already pointed out, building transactions in Mimblewimble is an interactive process between the sender and receiver of funds. Jedusor, Tom Elvis originally envisioned the following two-step process to build a transaction: [**jedusor2016mimblewimble**]

Assume Alice wants to transfer coins of value p to Bob.

Pedro: we need a way to express this clearer

1. Alice first selects input coins $\mathcal{C}_{inp}$ of total value $v \geq p$ that she controls. She then creates change coin outputs $\mathcal{C}^A_{out}$ (could be multiple) of total value $v - p$ and then sends $\mathcal{C}_{inp}$, $\mathcal{C}^A_{out}$, a valid range proofs for $\mathcal{C}^A_{out}$, plus the opening $(-p,\ x)$ of $\sum \mathcal{C}^A_{out} - \sum \mathcal{C}_{inp}$ to Bob.
2. Bob creates himself additional output coins $\mathcal{C}^B_{out}$ plus range proofs of total value p with keys (x_i^*) and computes a signature σ with the combined secret key $x + \sum x_i^*$ and finalizes the transaction as

$$tx := (0,\ \mathcal{C}_{inp},\ \mathcal{C}^A_{out} \mid\mid \mathcal{C}^B_{out},\ (\pi,\ \mathcal{E} := \sum \mathcal{C}^A_{out} \cdot \sum \mathcal{C}^B_{out} - \sum \mathcal{C}_{inp},\ \sigma))$$

and publishes it to the network.

Figure 3.1 depicts the original transaction flow.

Pedro: For security, privacy or both?

This protocol however turned out to be vulnerable. The receiver can spend the change coins $\mathcal{C}^A_{out}$ by reverting the transaction. Doing this would give the sender his coins back, however as the sender might not have the keys for his spent outputs anymore, the coins could then be lost.

In detail this reverting transaction would look like:

$$tx_{rv} := (0,\ \mathcal{C}^A_{out} \mid\mid \mathcal{C}^B_{out},\ \mathcal{C}_{inp},\ (\pi_{rv},\ \mathcal{E}_{rv},\ \sigma_{rv}))$$

Again remembering the construction of the Excess value of this construction would look like this:

$$\mathcal{E}_{rv} := \sum \mathcal{C}^A_{out} \mid\mid \mathcal{C}^B_{out} - \mathcal{C}_{inp}$$

The key x originally sent by Alice to Bob is a valid opening to $\sum \mathcal{C}_{inp} - \sum \mathcal{C}^A_{out}$. With the inverse of this key x_{inv} we get the opening to $\sum \mathcal{C}^A_{out} - \mathcal{C}_{inp}$. Now all Bob has to do is add his keys $\sum x_i^*$ to get:

$$x_{rv} := -x + \sum x_i^*$$

Pedro: Why range proof is not correct here in the first place?

which is the opening to $\mathcal{E}_{rv}$. Furthermore obtaining a valid range proofs is trivial, as it once was a valid output the ledger will cointain a valid proof for this coin already.

This means Bob spends the newly created outputs and sends them back to the original input coins, chosen by Alice. It might at first seem unclear why Bob would do that. An

Alice		**Bob**
Select $\mathcal{C}_{inp}$ of value $v \geq p$		
Create $\mathcal{C}_{out}^{A}$ of value $v - p$		
$\mathcal{E}_A := \sum \mathcal{C}_{out}^{A} - \sum \mathcal{C}_{inp}$		
$(-p,\ x)$ opening to $\mathcal{E}_A$		
Create range-proof π for $\mathcal{C}_{out}^{A}$		
	$\xrightarrow{(-p,\ x),\ \pi}$	
		Create $\mathcal{C}_{out}^{B}$ with value p and keys (x_i^*)
		$x_{shared} := x + \sum x_i^*$
		Create σ with x_{shared}
		Create range-proof π for $\mathcal{C}_{out}^{B}$
		Finalize transaction tx

Figure 3.1: Original transaction building process

Pedro: Really nice that you created this protocol :)

example situation could be if Alice pays Bob for some good which Bob is selling. Alice decides to pay in advance, but then Bob discovers that he is already out of stock of the good that Alice ordered. To return the funds to Alice, he reverses the transaction instead of participating in another interactive process to build a new transaction with new outputs. If Alice already deleted the keys to her initial coins, the funds are now lost. The problem was solved in the Grin Cryptocurrency by making the signing process itself a two-party process which will be explained in more detail in chapter 4.

Fuchsbauer et al. [**fuchsbauer2019aggregate**] proposed the following alternative way to build transactions which would not be vulnerable to this problem.

1. Alice constructs a full-fledged transaction tx_A spending her input coins $\mathcal{C}_{inp}$ and creates her change coins $\mathcal{C}_{out}^{A}$, plus a special output coin $\mathcal{C}_{out}^{sp} := h^p \cdot g^{x_{sp}}$, where p is the desired value which should be transferred to Bob and x_{sp} is a randomly choosen key. She proceeds by sending tx_A as well as $(p,\ x_{sp})$ and the necessary range proofs to Bob.

2. Bob now creates a second transaction tx_B spending the special coin $\mathcal{C}_{out}^{sp}$ to create an output only he controls $\mathcal{C}_{out}^{B}$ and merges tx_A with tx_B into tx_m. He then broadcasts tx_m to the network. Note that when the two transactions are merged the intermediate special coin $\mathcal{C}_{out}^{sp}$ will be both in the coin output and input list of the transaction and therfore will be discarded.

The only drawback of this approach is that we have two transaction kernels instead of

Alice		**Bob**
Select $\mathcal{C}_{inp}$ of value $v \geq p$		
Create $\mathcal{C}^{A}_{out}$ of value $v - p$		
$x_{sp} \leftarrow\$ \mathbb{Z}^{*}_{p}$		
Create $\mathcal{C}^{sp}_{out} := h^{p} + g^{x_{sp}}$		
Construct and sign tx_A with $\mathcal{C}_{inp}$, $\mathcal{C}^{A}_{out}$, $\mathcal{C}^{sp}_{out}$		
	$\xrightarrow{tx_A,\ p,\ x_{sp}}$	
		Create $\mathcal{C}^{B}_{out}$ with value p
		Create tx_B spending $\mathcal{C}^{sp}_{out}$ to $\mathcal{C}^{B}_{out}$
		Merge tx_A and tx_B to tx_m
		Publish tx_m

Figure 3.2: Salvaged tranction protocol by Fuchsbauer et al. [**fuchsbauer2019aggregate**]

Pedro: put labels inside captions

just one because of the merging step, making the transaction slightly bigger. A figure showing the protocol flow is depicted in Figure 3.2.

3.4 Scriptless Scripts

3.5 Adaptor Signatures

3.5.1 Schnorr Signature Construction

3.5.2 ECDSA Signature Construction

CHAPTER 4

Two Party Fixed Witness Adaptor Signatures

In this chapter, we will define a variant of the adaptor signature scheme as explained in section 3.5. The main difference in the protocol outlined in this thesis is that one of the two parties does know the fixed secret witness before the start of the protocol. The aim of the protocol will then be that the other person is able to extract the witness from the final signature. This feature can then be leveraged to build an Atomic Swap protocol as we will show in 5.

First we will define the general two-party signature creation protocol as it is currently implemented in Mimblewimble-based Cryptocurrencies. We reduce the generated signatures to the general case [**schnorr1989efficient**] and prove its correctness. From this two-party protocol, we then derive the adapted variant, which allows hiding a fixed witness value in the signature, which can be revealed only by the other party after attaining the final signature.

We start by defining our extended signature scheme in section 4.1, proceed by providing a schnorr-based instantiation of the protocol in section 4.2 and finally prove its security in section 4.2.1.

4.1 Definitions

A two-party signature scheme is an extension of a signature scheme as defined in definition 3.2, which allows us to distribute signature generation for a composite public key shared between two parties Alice and Bob. Alice and Bob want to collaborate to generate a signature valid under the composite public key $pk := pk_A \cdot pk_B$ without having to reveal their secret keys to each other.

Definition 4.1 (Two Party Signature Scheme)**.** A *two party signature scheme* Φ_{MP} extends a signature scheme Φ with a tuple of protocols and algorithms $(\mathsf{dKeyGen}, \mathsf{signPrt}, \mathsf{vrfPt}, \mathsf{finSig})$ defined as follows:

- $((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n\rangle$: The distributed key generation protocol takes as input the security parameter from both Alice and Bob and returns the tuple $(sk_A, pk_A, k_A, \Lambda)$ to Alice (similar to Bob) where (sk_A, pk_A) is a pair of private and corresponding public keys, k_A a secret nonce and Λ is the signature context containing parameters shared between Alice and Bob. We introduce Λ for the participants to share as well as update parameters with each other during the protocol execution.

- $(\tilde{\sigma_A}) \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$: The partial signing algorithm is a DPT function that takes as input the message m and the share of the secret key sk_A and nonce k_A (similiar for Bob) as well as the shared signature context Λ. The procedure outputs $(\tilde{\sigma_A})$, that is, a share of the signature to a participant.

- $\{1, 0\} \leftarrow \mathsf{vrfPt}(\tilde{\sigma_A},\ m,\ pk_A)$: The share verification algorithm is a DPT function that takes as input a signature share $\tilde{\sigma_A}$, a message m, and the other participants public key pk_A (similiar pk_B for Bobs partial signature). The algorithm returns 1 if the verification was successfull or 0 otherwise.

- $\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$: The finalize signature algorithm is a DPT function that takes as input two shares of the signatures and combines them into a final signature valid ander the shared public key pk.

We require the two party signature scheme to be correct as well as unforgeable against chosen message attacks (EUF-CMA). EUF-CMA for a two-party signature scheme was defined for instance in [**lindell2017fast**]. For the correctness of the distributed key-generation protcol $\mathsf{dKeyGen}$, special care needs to be taken to gurantee a uniformly random distribution of generated keys as pointed out by Lindell and Yehuda in [**lindell2017fast**].

Definition 4.2 (Two Party Fixed Witness Adaptor Schnorr Signature Scheme)**.** From the definition 4.1, we now derive an adapted signature scheme Φ_{Apt}, which allows one of the participants to hide the discrete logarithm x of a statement $X := g^x$ chosen at the beginning of the protocol. Again we extend our previously defined signature scheme with new functions:

$$\Phi_{Apt} := (\Phi_{MP} \mid\mid \mathsf{adaptSig} \mid\mid \mathsf{verifyAptSig} \mid\mid \mathsf{extWit})$$

- $\hat{\sigma_B} \leftarrow \mathsf{adaptSig}(\tilde{\sigma_B},\ x)$: The adapt signature algorithm is a DPT function that takes as input a partial signature $\tilde{\sigma}$ and a secret witness value x. The procedure will output a adapted partial signature $\hat{\sigma}$ which can be verified to contain x using the $\mathsf{verifyAptSig}$ function, without having to reveal x.

- $\{1,0\} \leftarrow \mathsf{verifyAptSig}(\hat{\sigma_A},\ m,\ pk_A,\ X)$: The verification algorithm is a DPT function that takes as input an adapted partial signature $\hat{\sigma}$, the other participants public keys and a statement X. The function will verify the partial signature's validity as well that it contains the secret witness x.

- $x \leftarrow \mathsf{extWit}(\sigma_{fin},\ \tilde{\sigma_A},\ \hat{\sigma_B})$: The witness extraction algorithm is a DPT function that lets Alice extract the secret witness x from the final composite signature. Note that to extract the witness x the partial signatures shared between the participants beforehand and the statement X needs to be provided as inputs. This means that for executing this function one needs to first learn the partial signatures exchanged between the parties.

Note that our definition of the adaptor signature scheme is different then the definition seen in 3.5. This has the reason that we require our scheme to be able to hide a secret chosen before the signing protocol has been started. One of the participants will be able to hide this secret during the distributed signing protocol which the other party can extract after completion of the protocol. This feature is a requirement for our signature scheme such that we can build the atomic swap protocol which will be layed out in 5.3.

Definition 4.3 (Secure Adaptor Signature Scheme). As defined by Aumayr et al. in [**aumayr2020bitcoinchan** a secure adaptor signature scheme needs two security properties to be fulfilled:

1. aEUF − CMA
2. Witness Extractability

We proceed by redefining these properties as well as adapted correctness for our adapted two-party fixed witness signature scheme defined in definition 4.2:

Similar to how it is defined in [**aumayr2020bitcoinchannels**] additionally to Correctness as defined in 3.2 we require our signature scheme to satisfy Adaptor Signature Correctness . This property is given when every adapted partial signature generated by adaptSig can be completed into a final signature for all pairs $(x,\ X) \in R$, from which it will be possible to extract the witness computing extWit with the required parameters.

Definition 4.4 (Adaptor Signature Correctness). More formally Adaptor Signature Correctness is given if for every security parameter $n \in \mathbb{N}$, message $m \in \{0,1\}^*$, keypairs $((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n\rangle$ with their composite public key $\Lambda.pk$ and every statement/witness pair $(X,\ x)\mathsf{genRel}(1^n)$ in a relation R it must hold that:

$$\Pr\left[\begin{array}{c|l} \mathsf{verf}(m,\ \sigma_{fin},\ \Lambda.pk)\ =\ 1 & \tilde{\sigma_A}\ \leftarrow\ \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda) \\ \wedge & \tilde{\sigma_B}\ \leftarrow\ \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda) \\ \mathsf{verifyAptSig}(\hat{\sigma_B},\ m,\ ,\ pk_B)X\ =\ 1 & \hat{\sigma_B}\ \leftarrow\ \mathsf{adaptSig}(\tilde{\sigma_B},\ x) \\ \wedge & \sigma_{fin}\ \leftarrow\ \mathsf{finSig}(\tilde{\sigma_A},\ \hat{\sigma_B}) \\ (X,\ x^*) \in R & x^*\ \leftarrow\ \mathsf{extWit}(\sigma_{fin},\ \tilde{\sigma_A},\ \hat{\sigma_B}) \end{array}\right] = 1.$$

Additionally to the regular definition of *existential unforgeability under chosen message attacks* as defined for example in [**lindell2017fast**] or [**vaudenay2006classical**] we require that it is hard to produce a forged partial signature $\tilde{\sigma}$ if the adversary $\mathcal{A}$ gets to know a valid adapted signature $\hat{\sigma}$ w.r.t. some message m and a statement X.

Definition 4.5 ($\mathsf{aEUF}-\mathsf{CMA}$)**.** For the definition of $\mathsf{aEUF}-\mathsf{CMA}$ -security we define the experiment $\mathsf{forgeAptSig}_{\mathcal{A}}$ for a PPT adversary $\mathcal{A}$ with a keypair $(sk_A,\ pk_A)$, meaning the attacker plays the role of Alice in the protocol as follows:

$\mathsf{forgeAptSig}_{\mathcal{A}}(n)$

$\mathbb{S} := \emptyset$

$((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n\rangle$

$m \leftarrow \mathcal{A}^{\mathcal{O}_{ps}(\cdot,\cdot,\cdot,\cdot)}(pk_B)$

$(x,\ X) \leftarrow \mathsf{genRel}(1^n)$

$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$

$\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)$

$\hat{\sigma_B} \leftarrow \mathsf{adaptSig}(\tilde{\sigma_B},\ x)$

$\tilde{\sigma_A}^* \leftarrow \mathcal{A}^{\mathcal{O}_{ps}(\cdot,\cdot,\cdot,\cdot)}(\tilde{\sigma_A}, \hat{\sigma_B})$

$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A}^*,\ \hat{\sigma_B})$

$\textbf{return } ((m) \notin \mathbb{S} \ \wedge\ \tilde{\sigma_A}^* \neq \tilde{\sigma_A} \ \wedge\ \mathsf{verf}(m,\ \sigma_{fin},\ \Lambda.pk))$

$\mathcal{O}_{ps}(m, pk_A, pk_B, \Lambda)$

$(x,\ X) \leftarrow \mathsf{genRel}(1^n)$

$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$

$\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)$

$\hat{\sigma_B} \leftarrow \mathsf{adaptSig}(\tilde{\sigma},\ x)$

$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \hat{\sigma_B})$

$\mathbb{S} := \mathbb{S} \cup \{m\}$

$\textbf{return } (\sigma_{fin},\ X)$

The adapted signature scheme Φ_{Apt} is called $\mathsf{aEUF}-\mathsf{CMA}$ -secure if

$$\Pr[\mathsf{forgeAptSig}_{\mathcal{A}}(n) = 1] \leq \mathsf{negl}(n)$$

Definition 4.6 (Witness Extractability)**.** Informally the Witness Extractability property holds for an adapted signature scheme Φ_{Apt} computed for the statement X when we can always extract the witness $(x,\ X)$ from the final signature σ_{fin}, given the partial signatures of the participants. To formalize this statement we describe an experiment $\mathsf{aExtrWit}_{\mathcal{A}}$ for a *PPT* adversary $\mathcal{A}$ with a keypair (sk_B, pk_B), meaning the attacker plays the role of Bob in the protocol.

$\underline{\mathsf{aExtrWit}_{\mathcal{A}}(n)}$

$$\begin{aligned}
&\mathbb{S} := \emptyset\\
&((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n\rangle\\
&(m,\ (x, X) \in R) \leftarrow \mathcal{A}^{\mathcal{O}_{ps}(\cdot,\cdot,\cdot,\cdot)}(pk_A)\\
&\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)\\
&\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)\\
&(\hat{\sigma_B},\ \sigma_{fin}) \leftarrow \mathcal{A}^{\mathcal{O}_{ps}(\cdot,\cdot,\cdot,\cdot)}(pk_B)\\
&x^* \leftarrow \mathsf{extWit}(\sigma_{fin},\ \tilde{\sigma_A},\ \hat{\sigma_B})\\
&\textbf{return } (m \notin \mathbb{S} \wedge (X,\ x^*) \notin R \wedge \mathsf{verf}(m,\ \sigma_{fin},\ \Lambda.pk))
\end{aligned}$$

$\underline{\mathcal{O}_{ps}(m, pk_A, pk_B, \Lambda)}$

$$\begin{aligned}
&\mathbb{S} := \mathbb{S} \cup m\\
&\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)\\
&\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)\\
&\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})\\
&\textbf{return } \sigma_{fin}
\end{aligned}$$

In order to satisfy witness extractability the following must hold:

$$\Pr[\mathsf{aExtrWit}_{\mathcal{A}}(n) = 1] \leq \mathsf{negl}(n)$$

4.2 Schnorr-based instantiation

We start by providing a general instantiation of a signature scheme (see definition 3.2): We assume we have a group $\mathbb{G}$ with prime p, H is a secure hash function as defined in definition 3.3 and $m \in \{0,1\}^*$ is a message.

- $(sk, pk) \leftarrow \mathsf{keyGen}(1^n)$: The keygen function creates a keypair (sk, pk), the public key can be distributed to the verifier(s) and the secret key has to be kept private.

- $\sigma \leftarrow \mathsf{sign}(m,\ sk)$: The signing function creates a signature consisting of a variable s and R which is a commitment to the secret nonce k used during the signing process. As an input it takes a message m and the secret key sk of the signer.

$\mathsf{keyGen}(1^n)$	$\mathsf{sign}(m,\ sk)$	$\mathsf{verf}(m,\ \sigma,\ pk)$
1 : $x \leftarrow\$\ \mathbb{Z}_p^*$	1 : $k \leftarrow\$\ \mathbb{Z}_p^*$	1 : $(s,\ R)\ \leftarrow\ \sigma$
2 : $\textbf{return}\ (sk\ :=\ x,\ pk\ :=\ g^x)$	2 : $R\ :=\ g^k$	2 : $e\ :=\ \mathsf{H}(m\ \|\|\ R\ \|\|\ pk)$
	3 : $e\ :=\ \mathsf{H}(m\ \|\|\ R\ \|\|\ pk)$	3 : $\textbf{return}\ g^s \stackrel{?}{=} R\ \cdot\ pk^e$
	4 : $s\ :=\ k\ +\ e\ \cdot\ sk$	
	5 : $\textbf{return}\ \sigma\ :=\ (s, R)$	

Figure 4.1: Schnorr Signature Scheme as first defined in [**schnorr1989efficient**]

- $\{1, 0\}\ \leftarrow\ \mathsf{verf}(m,\ \sigma,\ pk)$: The verification function allows a verifier knowing the signature σ, message m and the provers public key pk to verify the signatures validity.

A concrete implementation can be seen in figure 4.1. The signature scheme is called schnorr signature scheme, first defined in [**schnorr1989efficient**] and is widely employed in many cryptography systems. Correctness of the scheme is easy to derive. As s is calculated as $k\ +\ e\ \cdot\ sk$, when generator g is raised to s, we get $g^{k\ +\ e\ \cdot\ sk}$ which we can transform into $g^k\ \cdot\ g^{sk\ \cdot\ e}$, and finally into $R\ \cdot\ pk^e$ which is the same as the right side of the equation.

From the regular schnorr signature we now provide an instantiation for the two-party case defined in definition 4.1. Note that this two-party variant of the scheme is what is currently implemented in the mimblewimble-based cryptocurrencies and will provide a basis from which we will build our adapted scheme.

First we define a auxiliary function $\mathsf{setupCtx}$ to use for the instantion:

$\mathsf{setupCtx}(\Lambda,\ pk_A,\ R_A)$
1 : $\Lambda.pk\ :=\ \Lambda.pk\ \cdot\ pk_A$
2 : $\Lambda.R\ :=\ \Lambda.R\ \cdot\ R_A$
3 : $\textbf{return}\ \Lambda$

This function helps the particpents to setup and update the signature context shared between them. In figure 4.2 we show a concrete instantiation of the protocol and functions. In $\mathsf{dKeyGen}$ Alice and Bob will each randomly chose their secret key and nonce. They further require to create a zero-knowledge proof attesting to the fact that they have generated their key before any message was exchanged. This is essential for the scheme to achieve EUF-CMA as described by Lindell et al. [**lindell2017fast**].

In dKeyGen Alice will initially setup the signature context and send it to Bob, together with her public and zk-proof. Bob verifies the proof (and exits if it is invalid). He will proceed by adding his parameters to the signature context and send it back to Alice, together with his public key and zk-proof, which Alice will verify.

signPrt and vrfPt are generally similiar to the instantiation of the normal schnorr signature scheme. Note however that for computing the schnorr challenge e the input into the hash function will be the combined public key pk and combined nonce commitment R, which the participants can read from the context object Λ. This has the effect that the partial signature itself are not yet a valid signature (neither under pk nor under pk_A or pk_B). This is because to be valid under pk the partial signatures are missing the s values from the other participants. They are also not valid under the partial public keys pk_A or pk_B because the schnorr challenge is computed already with the combined values. There we have to introduce the slightly adjusted vrfPt to be able to verify specifically the partial signatures.

We further show in figure 4.3 how Alice and Bob can cooperate to produce a final signature which fulfills Correctness as defined in definition 3.2.

The final signature is a valid signature to the message m with the composite public key $pk := pk_A \cdot pk_B$. A verifier knowing the signed message m, the final signature σ_{fin} and the composite public key pk can now verify the signature using the regular verf procedure.

In figure 4.4 we further provide a schnorr-based instantiation for the fixed witness adapted signature scheme as defined in definition 4.2.

adaptSig will add the secret witness x to the s value of the signature, this means we will not be able to verify the adapted signature using vrfPt anymore. Therefore we introduce verifyAptSig which takes as additional parameter the statement X which will be included in the verifiers equation. Now the function verifies not only validity of the portial signature, but also that it indeed has been adapted with the witness value x, being the discrete logarithm of X. After obtaining σ_{fin}, we can then cleverly unpack the secret x, which is shown in the $\mathsf{aExtrWit}_{\mathcal{A}}$ function.

Again in figure 4.5 we show another example interaction between Alice and Bob creating a signature σ_{fin} for the composite public key $pk := pk_A \cdot pk_B$ while Bob will hide his secret x which Alice can extract after the signing process has completed. One thing to note is that in this protocol only Bob is able to call finSig to create the final signature. This is because the function requires Bobs unadapted partial signature $\tilde{\sigma_B}$ as input, which Alice does not know. (She only knows Bobs adapted variant). Therefore one further interaction is needed to send the final signature to Alice.

$\mathsf{dKeyGen}\langle 1^n,\ 1^n \rangle$

Alice		Bob
$sk_A \leftarrow\$\ \mathbb{Z}_p^*$		$sk_B \leftarrow\$\ \mathbb{Z}_p^*$
$k_A \leftarrow\$\ \mathbb{Z}_p^*$		$k_B \leftarrow\$\ \mathbb{Z}_p^*$
$pk_A := g^{sk_A}$		$pk_B := g^{sk_B}$
$R_A := g^{k_A}$		$R_B := g^{k_B}$
$st_A := \exists sk_A\ s.t.\ g^{sk_A} = pk_A$		$st_B := \exists sk_B\ s.t.\ g^{sk_B} = pk_B$
$\pi_A \leftarrow \mathsf{P}_{NIZK}(sk_A, st_A)$		$\pi_B \leftarrow \mathsf{P}_{NIZK}(sk_B, st_B)$
$\Lambda := \langle pk := 1_p,\ R := 1_p \rangle$		
$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ pk_A,\ R_A)$		
	$\xrightarrow{\Lambda, pk_A, \pi_A}$	
		if $\mathsf{V}_{NIZK}(\pi_A) = 0$
		return $\perp$
		$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ pk_B,\ R_B)$
	$\xleftarrow{\Lambda, pk_B, \pi_B}$	
if $\mathsf{V}_{NIZK}(\pi_B) = 0$		
return $\perp$		
return $(sk_A, pk_A, k_A, \Lambda)$		**return** $(sk_B, pk_B, k_B, \Lambda)$

$\mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$

$(R,\ pk) \leftarrow \Lambda$
$R_A := g^{k_A}$
$e := \mathsf{H}(m \mid\mid R \mid\mid pk)$
$s_A := k_A + sk_A \cdot e$
return $\tilde{\sigma_A} := (s_A, R_A, \Lambda)$

$\mathsf{vrfPt}(\tilde{\sigma_A},\ m,\ pk_A)$

$(s_A,\ R_A,\ \Lambda) \leftarrow \tilde{\sigma_A}$
$(pk,\ R) \leftarrow \Lambda$
$e := \mathsf{H}(m \mid\mid R \mid\mid pk)$
return $g^{s_A} \stackrel{?}{=} R_A \cdot pk_A^e$

$\mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$

$(s_A,\ R_A,\ \Lambda) \leftarrow \tilde{\sigma_A}$
$(s_B,\ R_B,\ \Lambda) \leftarrow \tilde{\sigma_B}$
$(pk,\ R) \leftarrow \Lambda$
$s := s_A + s_B$
$\sigma_{fin} := (s, R)$
return σ_{fin}

Figure 4.2: Two Party Schnorr Signature Scheme

Alice		**Bob**
	Public Parameters: $m,\ 1^n$	
	$((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n \rangle$	
$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$		$\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)$
	$\xleftarrow{\quad\tilde{\sigma_B}\quad}$	
	$\xrightarrow{\quad\tilde{\sigma_A}\quad}$	
$\mathsf{vrfPt}(\tilde{\sigma_B},\ m,\ pk_B) \stackrel{?}{=} 1$		$\mathsf{vrfPt}(\tilde{\sigma_A},\ m,\ pk_A)$
$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$		$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$

Figure 4.3: Two Party Schnorr Signature Scheme Interaction

$\underline{\mathsf{adaptSig}(\tilde{\sigma},\ x)}$

1 : $(s,\ R_A,\ \Lambda) \leftarrow \tilde{\sigma}$
2 : $s^* := s + x$
3 : **return** $\hat{\sigma} := (s^*,\ R_A,\ \Lambda)$

$\underline{\mathsf{verifyAptSig}(\hat{\sigma_A},\ m,\ pk_A,\ X)}$

1 : $(s_A,\ R_A,\ \Lambda) \leftarrow \hat{\sigma_A}$
2 : $(pk,\ R) \leftarrow \Lambda$
3 : $e := \mathsf{H}(m\ ||\ R\ ||\ pk)$
4 : **return** $g^{s_A} \stackrel{?}{=} R_A \cdot pk_A^e \cdot X$

$\underline{\mathsf{extWit}(\sigma_{fin},\ \tilde{\sigma_A},\ \hat{\sigma_B})}$

1 : $(s,\ R) \leftarrow \sigma_{fin}$
2 : $(s_A,\ R_A,\ \Lambda) \leftarrow \tilde{\sigma_A}$
3 : $(\hat{s_B},\ R_B,\ \Lambda) \leftarrow \hat{\sigma_B}$
4 : $s_B := s - s_A$
5 : $x := \hat{s_B} - s_B$
6 : **return** (x)

Figure 4.4: Fixed Witness Adaptor Schnorr Signature Scheme

Alice		**Bob**
	Public Parameters: $m,\ 1^n$ $((sk_A, pk_A, k_A, \Lambda), (sk_B, pk_B, k_B, \Lambda)) \leftarrow \mathsf{dKeyGen}\langle 1^n,\ 1^n\rangle$	
$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$		$(x,\ X) \leftarrow \mathsf{genRel}(1^n)$ $\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)$ $\hat{\sigma_B} \leftarrow \mathsf{adaptSig}(\tilde{\sigma_B},\ x)$
	$\xleftarrow{\quad \hat{\sigma_B}, X \quad}$	
$\mathsf{verifyAptSig}(\hat{\sigma_B},\ m,\ pk_B,\ X) \stackrel{?}{=} 1$		$\mathsf{vrfPt}(\tilde{\sigma_A},\ m,\ pk_A) \stackrel{?}{=} 1$
	$\xrightarrow{\quad \tilde{\sigma_A} \quad}$	
		$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \hat{\sigma_B})$
	$\xleftarrow{\quad \sigma_{fin} \quad}$	
$x \leftarrow \mathsf{extWit}(\sigma_{fin},\ \tilde{\sigma_A},\ \hat{\sigma_B})$		

Figure 4.5: Fixed Witness Adaptor Schnorr Signature Interaction

4.2.1 Correctness & Security

We now prove that the outlined schnorr-based instantiation is correct, i.e. Adaptor Signature Correctness holds, as well as secure with regards to the definition 4.3.

4.2.2 Adaptor Signature Correctness

To prove that Adaptor Signature Correctness holds we have 3 statements to prove, first we prove that $\mathsf{verf}(m,\ \sigma_{fin},\ \Lambda.pk) \stackrel{?}{=} 1$ holds in our schnorr-based instantiation of the signature scheme, whereas Λ is setup such that $pk = pk_A \cdot pk_B$.

Proof. For this prove we assume the setup already specified in definition 4.4. The proof is by showing equality of the equation checked by the verifier of the final signature by continuous substitutions in the left side of equation:

$$
\begin{aligned}
g^s &= R \cdot pk^e && (4.1)\\
g^{s_A} \cdot g^{s_B} & && (4.2)\\
g^{k_A + e \cdot sk_A} \cdot g^{k_B + e \cdot sk_B} & && (4.3)\\
g^{k_A} \cdot pk_A^e \cdot g^{k_B} \cdot pk_B^e & && (4.4)\\
R_A \cdot pk_A^e \cdot R_B \cdot pk_B^e & && (4.5)\\
R \cdot pk^e &= R \cdot pk^e && (4.6)\\
1 &= 1 && (4.7)
\end{aligned}
$$

It remains to prove that with the same setup $\mathsf{verifyAptSig}(\hat{\sigma_B},\ m,\ pk_B,\ X) \stackrel{?}{=} 1$ and $(X,\ x^*) \in R$ hold.

$$\mathsf{verifyAptSig}(\hat{\sigma_B},\ m,\ pk_B,\ X) \stackrel{?}{=} 1$$

The proof is by continuous substitutions in the equation checked by the verifier:

$$\begin{align} g^{\hat{\sigma_B}} &= R_B \cdot pk_B^e \cdot X \tag{4.8}\\ g^{\tilde{\sigma_B} + x} & \tag{4.9}\\ g^{k_B + sk_B \cdot e + x} & \tag{4.10}\\ g^{k_B} \cdot g^{sk_B \cdot e} + g^x & \tag{4.11}\\ R_B \cdot pk_B^e \cdot X &= R_B \cdot pk_B^e \cdot X \tag{4.12}\\ 1 &= 1 \tag{4.13} \end{align}$$

We now continue to prove the last equation required:

$$((X,\ x^*) \in R)$$

We do this by showing that x is calculated correctly in extWit:

$$\begin{align} x &:= \hat{s} - (s - s_A) \tag{4.14}\\ &\quad \hat{s} - ((s_A + s_B) - s_A) \tag{4.15}\\ &\quad s_B + x - (s_B) \tag{4.16}\\ x &:= x \tag{4.17}\\ & \tag{4.18} \end{align}$$

□

4.2.3 Secure Adaptor Signature Scheme

In order to prove the security of the scheme we need to provide a proof that both $\mathsf{aEUF-CMA}$ and Witness Extractability hold in our instantiation. To perform the proof we must first recall the regular definition of $\mathsf{EUF-CMA}$ given for regular schnorr signatures. [**schnorr1989efficient**] For that we define the game $\mathsf{forgeSig}_{\mathcal{A}}$:

$\mathsf{forgeSig}_{\mathcal{A}}(n)$	$\mathcal{O}_s(m, pk)$
1 : $\mathbb{S} \leftarrow \emptyset$	1 : $\sigma \leftarrow \mathsf{sign}(m,\ sk)$
2 : $(sk, pk) \leftarrow \mathsf{keyGen}(1^n)$	2 : $\mathbb{S} := \mathbb{S} \cup \{m\}$
3 : $(m,\ \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_s(\cdot,\cdot)}$	3 : **return** σ
4 : **return** $((m) \notin \mathbb{S} \wedge \mathsf{verf}(m,\ \sigma,\ pk))$	

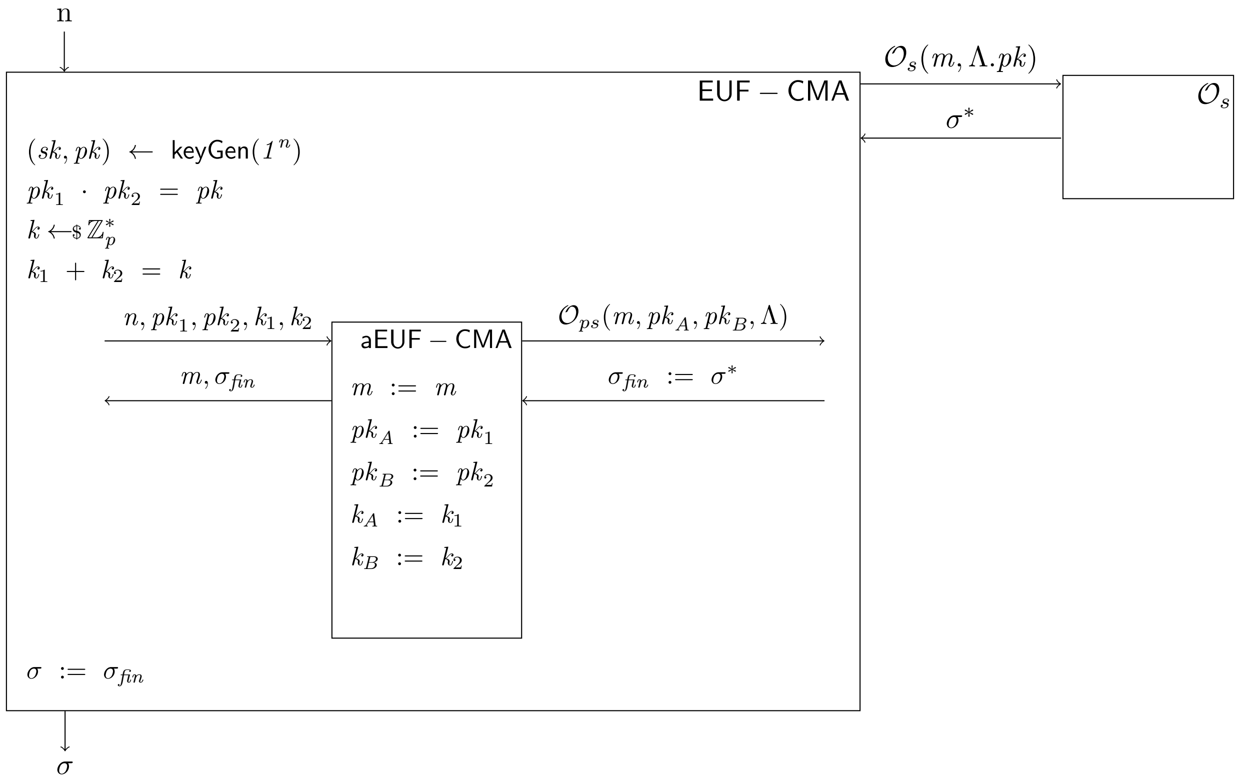

Figure 4.6: Reduction from aEUF − CMA to EUF − CMA

EUF − CMA holds if $\Pr[\mathsf{forgeSig}_{\mathcal{A}}(n) = 1] \leq \mathsf{negl}(n)$.

Proof. We proof aEUF − CMA holds by providing a black box reduction from aEUF − CMA to EUF − CMA of schnorr signatures. Intuitively if we suppose there exists a PPT adversary $\mathcal{A}$ that wins the $\mathsf{forgeAptSig}_{\mathcal{A}}$ game with probability 1, then $\mathcal{A}$ will also be able to win the $\mathsf{forgeSig}_{\mathcal{A}}$ game with the same probability, which leads us to a contradiction. This can be achieved by splitting up the public key chosen by the challenger in the $\mathsf{forgeSig}_{\mathcal{A}}$ game into pk_1 and pk_2 and then running the $\mathsf{forgeAptSig}_{\mathcal{A}}$ game using the two split keys. Since the signing oracle in aEUF − CMA and Witness Extractability both provide a signature valid unter the composite of the two public keys, we can simulate the oracle queried by the adversary $\mathcal{A}$ simply by forwarding the query to the EUF − CMA oracle with the original unsplit version of the public key. The output of the $\mathsf{forgeAptSig}_{\mathcal{A}}$ game will be a forged final signature valid under the combined public key of pk_1 and pk_2 which we can then use to win the $\mathsf{forgeSig}_{\mathcal{A}}$ game. See figure 4.6 for the black-box reduction.

In a very similiar way we can provide a reduction from Witness Extractability to EUF − CMA . Again if we suppose there exists a PPT adversary $\mathcal{A}$ able to win the $\mathsf{aExtrWit}_{\mathcal{A}}$ game with probability 1, then $\mathcal{A}$ will always be able to win the $\mathsf{forgeSig}_{\mathcal{A}}$, leading to a contradiction. Similiar to the previous proof the adversary $\mathcal{A}$ splits up the secret key pk computed during the $\mathsf{forgeSig}_{\mathcal{A}}$ game into pk_1 and pk_2 to use them in the $\mathsf{aExtrWit}_{\mathcal{A}}$. The forged final signature σ_{fin} can then be used to win the $\mathsf{forgeSig}_{\mathcal{A}}$ game. As the black-box reduction is the same as before we again refer to figure 4.6 to see the details.

$\square$

CHAPTER 5

Adaptor Signature Based Atomic Swaps Between Bitcoin and a Mimblewimble Based Cryptocurrency

In this section, we will define Mimblewimble transactions and required security properties, similiar to those found by Fuchsbauer et al. in [**fuchsbauer2019aggregate**]. We will only focus on the creation of new transfer transactions (transferring value from one or many parties to one or many parties), the notions of transaction aggregation, coin minting and transaction publishing discussed in [**fuchsbauer2019aggregate**] will not be topic of this formalization, as they are not relevant building blocks for the proposed Atomic Swap protocol.

However, as an extension to the regular transaction protocol transferring value from one sender to a receiver we will define two further protocols. The first of them titled Extended Mimblewimble Transaction Scheme will provide additional functions to create and spend coins owned by two parties instead of just one. The second extended definition is called Adapted Extended Mimblewimble Transaction Scheme and will allow the receiver of a coin to hide a secret witness value x in his signature, in a way that the sender (or the senders) can redeem this secret after the protocol has completed.

We will proceed by providing an instantiation of the three transactions schemes which can be implemented and deployed on a Mimblewimble based Cryptocurrency such as Beam or Grin. Furthermore we provide proofs that the schemes are correct and secure with regards to the defined security properties.

Finally, we define a Atomic Swap protocol from these building blocks, allowing two parties to securely and trustlessly swap funds from a Mimblewimble blockchain with those on another Blockchain, such as Bitcoin.

5.1 Definitions

As we have already discussed in section 3.3.3 for the creation of a transaction, it is immanent that both the sender and receiver collaborate and exchange messages via a secure channel. To construct the transaction protocol we assume that we have access to a two-party signature scheme Φ_{MP} as defined in definition 4.1, a zero-knowledge Range-proofs system Π such as Bulletproofs, as described in section 3.3.2 and a homomorphic commitment scheme COM as defined in definition 3.5 such as Pedersen Commitments 3.6.

Fuchsbauer et al. have defined three procedures Send, Rcv and Ldgr with regards to the creation of a transaction. Send called by the sender will create a pre-transaction, Rcv takes the pre-transaction and adds the receivers output and Ldgr (again called by the sender) publishes the final transaction to the Blockchain ledger. As we already pointed out in this thesis we won't discuss the transaction publishing phase therefore we will not cover the functionality of the Ldgr procedure, instead we indroduce two functions finSig and verfTx. finSig can be called by the transaction sender to finalize a pre-transaction into final valid transaction, which then could be broadcast with a node connected to the Blockchain. The verfTx function is called by nodes (acting as public verifiers) on the blockchain verifying the validity of the transaction, before including them to a block.

Definition 5.1 (Mimblewimble Transaction Scheme)**.** A Mimblewimble transaction scheme $MW[COM, \Phi_{MP}, \Pi]$ consist of the following procedures:

$$MW[COM, \Phi_{MP}, \Pi] := (\mathsf{spendCoins}, \mathsf{recvCoins}, \mathsf{finTx}, \mathsf{verfTx})$$

- $(ptx, (sk_A, k_A)) \leftarrow \mathsf{spendCoins}([\mathcal{C}_{inp}], [r], p, v)$: The spendCoins algorithm is a DPT function called by the sending party to initiate the spending of some input coins. As input it takes a list of coins $[\mathcal{C}_{inp}]$ which should be spent, the respective keys $[r]$ to the input coins and a value p which should be transferred to the receiver as well as v which is the total value stored in the input coins. It outputs a pre-transaction ptx which can be sent to a receiver, as well as the senders signing key and nonce (sk_A, k_A).

- $ptx \leftarrow \mathsf{recvCoins}(ptx, p)$: The receiveCoins algorithm is a DPT function called by the receiver and takes as input a pre-transaction ptx and a fund value p and will output a modified pre-transaction ptx.

- $tx \leftarrow \mathsf{finTx}(ptx, sk_A, k_A)$: The finalize algorithm is a DPT function again called by the transaction sender that takes as input a pre-transaction ptx and the senders signing key sk_A. The function will output a finalized transaction tx, which can be published to the blockchain.

- $\{1, 0\} \leftarrow \mathsf{verfTx}(tx)$: The transaction verification algorithm is a DPT function which can be called by a public verifier and takes as input a transaction tx. It outputs either 1 on verification success or 0 otherwise.

Definition 5.2 (Extended Mimblewimble Transaction Scheme)**.** An extended Mimblewimble transaction scheme $MW_{ext}[COM, \Phi_{MP}, \Pi]$ is an extension to MW with the following two procedures:

$$MW_{ext}[COM, \Phi_{MP}, \Pi] \ := \ MW[COM, \Phi_{MP}, \Pi] \ || \ (\mathsf{dSpendCoinsId}, \mathsf{dRecvCoins})$$

- $((ptx, sk), (ptx, sk)) \leftarrow \mathsf{dSpendCoinsId}\langle([\mathcal{C}_{inp}], [r_A], p, v), \ ([\mathcal{C}_{inp}], [\mathcal{C}_{inp}], p, [r_B])\rangle$: The distributed coin spending algorithm takes as input a list of input coins, as well as a list of blinding factors from each Alice and Bob, and the to be transferred value p and total value of input coins v. Note that for each provided input coin $\mathcal{C}_{inp}$ the blinding factor is composed by combining the shares from Alice and Bob like $r \ := \ r_A \ + \ r_B$.

- $(ptx, ptx) \leftarrow \mathsf{dRecvCoins}\langle(ptx, p), \ (ptx, p)\rangle$: The distributed coin receive procedure takes as input a pre-transaction ptx and a value p which should be transferred. The distributed algorithm will generate a output coin owned by both Alice and Bob. (each owning a share of the key). The output will be similiar to the single party version a updated pre-transaction ptx.

Definition 5.3 (Adapted Extended Mimblewimble Transaction Scheme)**.** The adapted version of the extended Mimblewimble Transaction Scheme updates the Extended Mimblewimble Transaction Schme by providing a modified version of the single party receive function and the finalize transaction function.

$$MW_{apt}[COM, \Phi_{MP}, \Pi] \ := \ MW_{ext}[COM, \Phi_{MP}, \Pi] \ || \ \mathsf{aptSpendCoins}, \mathsf{dAptFinTx}$$

- $(ptx, X) \leftarrow \mathsf{aptSpendCoins}(ptx, p, x)$: The adapted variant of the receive function takes an additional input a secret witness value x which will be hidden in the transactions signature and extractable by the other party after the protocols completion. Additionally to the updated pre-transaction ptx it returns the statement x which can be sent to the other party.

- $((tx, x), tx) \leftarrow \mathsf{dAptFinTx}\langle(ptx, sk_A, k_A, X), (ptx, \tilde{\sigma_B})\rangle$: The adapted variant of the finalize transaction algorithm is a distributed protocol between the sender and receiver. Additionally to the pre-transaction ptx Alice (the sender) needs to input her signing key and the statement X received from Bob. Bob (the receiver) needs to provide the secret witness x as input. The protocol will output the final transaction as well as x to Alice.

We first define a security property which informally states that a transactions output value can only be less or equal to the value of its input coins (if less then the miner of the transaction can extract fees). In other words a transaction in our definition shall only transfer existing but never generate any new value. We call this security property *Inflation Resistance*. In order to define this property we first have to define a cryptographic game inflate which takes as input a security parameter 1^n and a value v which the Adversary $\mathcal{A}$ tries to inflate.

We define the game inflate as follows, whereas a challenger creates a input coin with value v given as a parameter to the game. The adversary then chooses a value v^* with $v^* > v$ and creates a new output coin. The adversary wins if he can construct a valid transaction spending the challengers input coin $\mathcal{C}_{inp}$ to $\mathcal{C}_{out}$ and thereby creating new value $v^* - v$.

$\mathsf{inflate}(1^n, v)$

$r \leftarrow\$ \mathbb{Z}^*_*$

$(\mathcal{C}_{inp}, \pi) \leftarrow \mathsf{createCoin}(v, r)(r^*, v^*) \leftarrow \mathcal{A}(\mathcal{C}_{inp}, v)$

$(\mathcal{C}_{out}, \pi) \leftarrow \mathsf{createCoin}(v^*, r)$

$tx\mathcal{A}(\mathcal{C}_{inp}, v, \mathcal{C}_{out})$

$\textbf{return } v^* > v \wedge \mathsf{verfTx}(tx) = 1 \wedge tx.out \stackrel{?}{=} [\mathcal{C}_{out}] \wedge tx.inp \stackrel{?}{=} [\mathcal{C}_{inp}]$

Definition 5.4 (Inflation Resistence)**.** A Mimblewimbe Transaction Scheme is called inflation resistent if for any value v in a valid range (as defined by the public parameters of the ledger), security paramter 1^n and a polytime adversary $\mathcal{A}$ the following holds:

$$\Pr[\mathsf{inflate}(1^n, v) = 1] \leq \mathsf{negl}(n)$$

In a Mimblewimble transaction scheme a coins ownership is given by the knowledge of its blinding factor r. To spend the coin the sender would also have to know the coins value v in addition to the blinding factor, however as the possible value v is restriced by the ledgers parameters, it is trivial to guess. Therefore we assume that knowledge of the blinding factor r alone implies ownership of the coin.

$\mathsf{stealCoin}(1^n, \mathcal{C}_{inp})$

$(ptx, sk, p) \leftarrow \mathcal{A}(\mathcal{C}_{inp})$

$ptx^* \leftarrow \mathsf{recvCoins}(ptx, p)$

$tx \leftarrow \mathcal{A}(ptx^*)$

$\textbf{return } \mathsf{verfTx}(tx) = 1 \wedge \mathcal{C}_{inp} \in tx.inp$

Definition 5.5 (Theft-resistance)**.**

Definition 5.6 (Transaction indistinguishability)**.** TODO

5.2 Mimblewimble instantiation

In this section we will provide an instantiation of the transaction scheme definitions found in 5.1, 5.2 and 5.3. The instantiations can be implemented in a Cryptocurrency based on the Mimblewimble protocol such as Beam and Grin.

5.2.1 Mimblewimble Transation Scheme

First we provide an instantiation of the simplest form of a transaction in which a sender wants to transfer some value p to a receiver. For the execution of the protocol we assume to have access to a homomorphic commitment scheme such as Pedersen Commitment COM as defined in definition 3.6. Furthermore we require a Rangeproof system Π as defined in 3.3.2 and a two-party signature scheme Φ_{MP} as defined in 4.1.

The make the pseudocode for the transaction protocol easier we first introduce two auxiliary functions createCoin and createTx. The coin creation function will take as input a value v and a blinding factor r, it will create and output a new coin $\mathcal{C}$ together with a range proof π attesting to the statement that the coins value v is within the valid range as defined for the blockchain. The transaction creation algorithm createTx takes as input a message m, a list of input coins $[\mathcal{C}_{inp}]$, a list of output coins $[\mathcal{C}_{out}]$, a list of rangeproofs $[\pi]$, a signature context Λ, a list of nonce commitments C and a signature σ and will collect the input data into a transaction object.

$\mathsf{createCoin}(v, r)$	$\mathsf{createTx}(m, [\mathcal{C}_{inp}], [\mathcal{C}_{out}], [\pi], \Lambda, [C], \sigma)$
$\mathcal{C} \leftarrow \mathsf{commit}(v, r)$	**return** (
$\pi \leftarrow \mathsf{ranPrf}(\mathcal{C}, v, r)$	$m := m,$
return $(\mathcal{C}, \pi)$	$inp := [\mathcal{C}_{inp}],$
	$out := [\mathcal{C}_{out}],$
	$\Pi := [\pi],$
	$\Lambda := \Lambda,$
	$com := [C],$
	$\sigma := \sigma$
	)

In figure 5.1 we provide an instantiation of the Mimblewimble Transction Scheme using the auxiliary functions provided before.

In the spendCoins function the sender creates his change output coin, which is the difference between the value stored in his input coins and the value which should be transferred to a receiver. He sets up the signature context with his parameters and gets a pretransaction ptx, as well as a signing key sk_A as output. The pretransaction can then be sent to a receiver. Note that this instantiation differs from the one described by Fuchsbauer et al. [**fuchsbauer2019aggregate**] in that the sender does not yet sign the transaction during spendCoins. This has the reason that in our definition of the Two-Party Signature Scheme 4.1 the signature context Λ requires to be fully setup before a partial signature can be created, therfore signing can only start at the recievers turn, after the signature context has been completed. In the paper by Fuchsbauer et al. signing already at the senders turn works, because instead of using the notion of a two party signing protocol, they instead rely on a aggregateable signature scheme. The sender and receiver both will create their signatures which will then be aggregated into the final one. However, we find that by relying upon a two-party signature scheme instead we are closer to what is implemented in practice[1] . Furthermore by starting the signing process at the receivers turn we avoid a potential problem: If an adversary learns the already signed pre-transaction *ptx* and the value *p* before the intended receiver, the adversary would be able to steal the coins by creating his malicious output coin together with his signature, which he could then aggregate to the senders pre-transaction.

In recvCoins the receiver of a pre-transaction will verify the senders proof π_B, create his outputcoin $\mathcal{C}_{out}^{B}$, add his parameters to the signature context and then create his partial signature $\tilde{\sigma_B}$. The function returns an updated version of the pre-transaction *ptx* which can be sent back to the sender.

Now in finTx the original sender will validate the updated pre-transcation *ptx* sent to him by the receiver. If he finds it as valid, he will only now create his partial signature and finally finalize the two partial signatures in the final composite one, with which he can then build the final transaction.

In verfTx a public verifier will verify the rangeproofs for the transactions output coins. Then he will compute the so-called Excess value $\mathcal{E}$ from the difference between output and input coins and use it as the public key for validating the transation signature. If the signature is now also found to be correct the verfiier can deduce that the transaction is well-formed and valid.

5.2.2 Extended Mimblewimble Transaction Scheme

First of all we define a protocol dSign which is a protocol between two-parties running the partial signature creation outlined in section 4.2, which will be used to simplify the pseudocodes of the following protocols. Note that we assume that the secret keys as well as nonces are already given as a paramter (which is the case during the transaction protocol) therefore we don't need an additional call to dKeyGen.

[1] **https://medium.com/@brandonarvanaghi/grin-transactions-explained-step-by-step-fdceb905a853**.

$\mathsf{spendCoins}([\mathcal{C}_{inp}], [r_A], p, v)$

$m := \{0,1\}^*$
$(r_A^*, k_A) \leftarrow\$\ \mathbb{Z}_p^*$
$(\mathcal{C}_{out}^A, \pi_A) \leftarrow \mathsf{createCoin}(v - p, r_A^*)$
$sk_A := r_A^* - \sum[r_A]$
$\Lambda := \{pk := 1_p, R := 1_p\}$
$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_A},\ g^{k_A})$
$ptx \leftarrow \mathsf{createTx}(m, [\mathcal{C}_{inp}], [\mathcal{C}_{out}^A], [\pi_A], \Lambda, [g^{sk_A}], \emptyset)$
$\mathbf{return}\ (ptx, (sk_A, k_A))$

$\mathsf{recvCoins}(ptx, p)$

$(m, inp, out, \Pi, \Lambda, com, \emptyset) \leftarrow ptx$
$\mathbf{if}\ \mathsf{vrfRanPrf}(\Pi[0], out[0]) = 0$
$\quad \mathbf{return}\ \perp$
$(r_B^*, k_B) \leftarrow\$\ \mathbb{Z}_p^*$
$(\mathcal{C}_{out}^B, \pi_B) \leftarrow \mathsf{createCoin}(p, r_B^*)$
$sk_B := r_B^*$
$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_B},\ g^{k_B})$
$\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ \Lambda.pk,\ \Lambda.R)$
$ptx \leftarrow \mathsf{createTx}(m, inp, out \mid\mid \mathcal{C}_{out}^B, \Pi \mid\mid \pi_B, \Lambda, com \mid\mid g^{k_B}, \tilde{\sigma_B})$
$\mathbf{return}\ ptx$

$\mathsf{finTx}(ptx, sk_A, k_A)$

$(m, inp, out, \Pi, \Lambda, com, \tilde{\sigma_B}) \leftarrow ptx$
$\mathbf{if}\ \mathsf{vrfRanPrf}(\Pi[1], out[1]) = 0$
$\quad \mathbf{return}\ \perp$
$\mathbf{if}\ \mathsf{vrfPt}(\tilde{\sigma_B},\ m,\ com[1]) = 0$
$\quad \mathbf{return}\ \perp$
$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$
$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$
$tx \leftarrow \mathsf{createTx}(m, inp, out, \Pi, \Lambda, com, \sigma_{fin})$
$\mathbf{return}\ tx$

$\mathsf{verfTx}(tx)$

$(m, inp, out, \Pi, \Lambda, com, \sigma) \leftarrow tx$
$\mathbf{foreach}\ out\ as\ (i => \mathcal{C}_{out})$
$\quad \mathbf{if}\ \mathsf{vrfRanPrf}(\Pi[i], \mathcal{C}_{out}[i]) = 0$
$\qquad \mathbf{return}\ 0$
$pk := \sum out - \sum inp$
$\mathbf{return}\ \mathsf{verf}(\sigma,\ m,\ pk)$

Figure 5.1: Instantiation of Mimblewimble Transaction Scheme.

$\mathsf{dSign}\langle(m, sk_A, k_A), (m, sk_B, k_B)\rangle$

Alice		*Bob*
$\Lambda := \{pk := 1_p, R := 1_p\}$		
$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_A},\ g^{k_A})$		
	$\xrightarrow{\Lambda, pk_A\ :=\ g^{sk_A}}$	
		$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_B},\ g^{k_B})$
		$\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ k_B,\ \Lambda)$
	$\xleftarrow{\tilde{\sigma_B}, \Lambda, pk_B\ :=\ g^{sk_B}}$	
if $\mathsf{vrfPt}(\tilde{\sigma_B},\ m,\ pk_B) = 0$		
return $\perp$		
$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m,\ sk_A,\ k_A,\ \Lambda)$		
	$\xrightarrow{\tilde{\sigma_A}}$	
		if $\mathsf{vrfPt}(\tilde{\sigma_A},\ m,\ pk_A) = 0$
		return $\perp$
$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$		$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A},\ \tilde{\sigma_B})$
return σ_{fin}		**return** σ_{fin}

Figure 5.2 shows an instantiation of the dSpendCoinsId function of the Extended Mimblewimble Transaction Scheme. We have one array of input coins which keys are shared between two parties A and C, which should be spent. We use C here to not confuse this party with the receiver, which we previously called B. Although C and B could be the same person, they not necessarily have to be.
The protocol starts with both Alice and Carol creating her spend outputs with values v_A and v_C. Alice then creates the initial pre-transaction *ptx* and sends it to Carol who verifies Alice's output, adds her outputs and paramters and sends back *ptx*, which Alice verifies. The protocol returns *ptx* to both parties, which can then be transmitted to the receiver by any of the two parties, as well as the secret signing information (sk_A, k_A), (sk_C, k_C). Note that when using this protocol to spend coins that finTx also turns into a protocol, using dSign instead of signPrt for signature generation. (Apart from that finTx remains unchanged)

Figure 5.3 shows an instantiation of the recvCoins function of the Extended Mimblewimble Transaction Scheme. Calling this protocol two receivers B and C want to create a receiving shared coin $\mathcal{C}_{out}^{sh}$ with value p and key shares (sk_A, sk_B). The protocol starts by both receivers verifing the senders output(s). Bob starts by creating a coin with fund value p and his share of the newly create blinding factor and sends it over to Carol. Carol finalizes the shared coin by adding a commitment to her blinding factor to the coin and

$\mathsf{dSpendCoinsId}\langle([\mathcal{C}_{inp}], [r_A], p, v_A),\ ([\mathcal{C}_{inp}], [r_C], p, v_C)\rangle$

A		C
$m := \{0,1\}^*$		
$(r_A^*, k_A) \leftarrow\!\$\ \mathbb{Z}_p^*$		$(r_C^*, k_C) \leftarrow\!\$\ \mathbb{Z}_p^*$
$(\mathcal{C}_{out}^A, \pi_A) \leftarrow \mathsf{createCoin}(v_A, r_A^*)$		$(\mathcal{C}_{out}^C, \pi_C) \leftarrow \mathsf{createCoin}(v_C, r_C^*)$
$sk_A := r_A^* - \sum[r_A]$		$sk_C := r_C^* - \sum[r_C]$
$\Lambda := \{pk := 1_p, R := 1_p\}$		
$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_A},\ g^{k_A})$		
$ptx \leftarrow \mathsf{createTx}(m, [\mathcal{C}_{inp}], [\mathcal{C}_{out}^A], [\pi_A], \Lambda, [g^{k_A}], \emptyset)$		
	$\xrightarrow{ptx}$	
		$(m, inp, out, \Pi, \Lambda, com) \leftarrow ptx$
		if $\mathsf{vrfRanPrf}(\Pi[0], out[0]) = 0$
		return $\perp$
		$\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_C},\ g^{k_C})$
		$ptx \leftarrow \mathsf{createTx}(m, inp, out \mid\mid \mathcal{C}_{out}^C, \pi \mid\mid \pi_C, \Lambda, com \mid\mid g^{k_C}, \emptyset)$
	$\xleftarrow{ptx}$	
if $\mathsf{vrfRanPrf}(ptx.\Pi[1], ptx.out[1]) = 0$		
return $\perp$		
return $(ptx, (sk_A, k_A))$		**return** $(ptx, (sk_C, k_C))$

Figure 5.2: Extended Mimblewimble Transaction Scheme - $\mathsf{dSpendCoinsId}$

sends it back, together with the commitment. Bob verifies validity of the updated shared coin after which the two parties engange in two-party protocols to create their partial signature and coin rangeproof. Finally they create the updated pre-transaction *ptx* which can be sent back to the sender.

5.2.3 Adapted Extended Mimblewimble Transaction Scheme

Figure 5.4 shows an instantiation of the aptSpendCoins algorithm. Before updating the pre-transaction *ptx* Bob adapts his partial signature with the witness value x. The procedure then returns the pre-transaction *ptx* containing Bobs adapted partial signature, and the statement X which is a commitment to the witness value x.

In figure 5.5 we show the updated distributed version of the transaction finalization protocol. Again Alice verifies the pre-transaction *ptx* received by Bob and then proceeds by building her own partial signature. Note that at this point Alice is not able to finalize the signature (and consequently the transaction) as she only knows Bobs adapted partial signature, but not the original one, which is needed for the finSig function. Therefore in another round of interaction Alice sends her partial signature to Bob, who will verify Alice partial signature and finally calculate the final signature, needed for the transaction. He will send over σ_{fin} which lets both parties construct the valid transaction as well as Alice call extWit to extract the secret witness x.

5.3 Atomic Swap protocol

$\mathsf{dRecvCoins}\langle(ptx, p),\ (ptx, p)\rangle$

Bob		$Carol$
$(m, inp, out, \Pi, \Lambda, com, \tilde{\sigma_B}) \leftarrow ptx$		$(m, inp, out, \Pi, \Lambda, com, \tilde{\sigma_B}) \leftarrow ptx$
foreach out as $(i => \mathcal{C}_{out})$		**foreach** out as $(i => \mathcal{C}_{out})$
if $\mathsf{vrfRanPrf}(\Pi[i], \mathcal{C}_{out}[i]) = 0$		**if** $\mathsf{vrfRanPrf}(\Pi[i], \mathcal{C}_{out}[i]) = 0$
return $\perp$		**return** $\perp$
$(r_B^*, k_B) \leftarrow\!\$\, \mathbb{Z}_p^*$		
$(\mathcal{C}_{out}^{sh}, \cdot) \leftarrow \mathsf{createCoin}(p, r_B^*)$		
$sk_B := r_B^*$		
	$\xrightarrow{\mathcal{C}_{out}^{sh}}$	
		$(r_C^*, k_C) \leftarrow\!\$\, \mathbb{Z}_p^*$
		$sk_C := r_C^*$
		${\mathcal{C}_{out}^{sh}}' := \mathcal{C}_{out}^{sh} \cdot g^{sk_C}$
	$\xleftarrow{{\mathcal{C}_{out}^{sh}}', g^{sk_C}}$	
		if ${\mathcal{C}_{out}^{sh}}' \neq \mathcal{C}_{out}^{sh} \cdot g^{sk_C}$
		return $\perp$
	$(\tilde{\sigma_{BC}}, pk_{BC}, \Lambda^*) \leftarrow \mathsf{dSign}\langle(m, sk_B, k_A), (m, sk_B, k_B)\rangle$	
	$\Lambda' \leftarrow \mathsf{setupCtx}(\Lambda,\ \Lambda^*.pk,\ \Lambda^*.k)$	
	$\pi_{BC} \leftarrow \mathsf{dRanPrf}\langle({\mathcal{C}_{out}^{sh}}', p, sk_A), ({\mathcal{C}_{out}^{sh}}', p, sk_B)\rangle$	
	$ptx \leftarrow \mathsf{createTx}(m, inp, out \mid\mid {\mathcal{C}_{out}^{sh}}', \Pi \mid\mid \pi_{BC}, \Lambda', com \mid\mid pk_{BC}, \tilde{\sigma_{BC}})$	
return ptx		**return** ptx

Figure 5.3: Extended Mimblewimble Transaction Scheme - dRecvCoins

$\mathsf{aptSpendCoins}(ptx, p, x)$

$$
\begin{array}{l}
(m, inp, out, \Pi, \Lambda, com, \emptyset) \leftarrow ptx \\
\textbf{if } \mathsf{vrfRanPrf}(\Pi[0], out[0]) = 0 \\
\quad \textbf{return } \bot \\
(r_B^*, k_B) \leftarrow\!\$\, \mathbb{Z}_p^* \\
(\mathcal{C}_{out}^B, \pi_B) \leftarrow \mathsf{createCoin}(p, r_B^*) \\
sk_B := r_B^* \\
\Lambda \leftarrow \mathsf{setupCtx}(\Lambda,\ g^{sk_B},\ g^{k_B}) \\
\tilde{\sigma_B} \leftarrow \mathsf{signPrt}(m,\ sk_B,\ \Lambda.pk,\ \Lambda.R) \\
\hat{\sigma_B} \leftarrow \mathsf{adaptSig}(\tilde{\sigma_B},\ x) \\
ptx \leftarrow \mathsf{createTx}(m, inp, out \mid\mid \mathcal{C}_{out}^B, \Pi \mid\mid \pi_B, \Lambda, com \mid\mid g^{k_B}, \hat{\sigma_B}) \\
\textbf{return } (ptx, X)
\end{array}
$$

Figure 5.4: Adapted Extended Mimblewimble Transaction Scheme - aptSpendCoins.

$\mathsf{dAptFinTx}\langle(ptx, sk_A, k_A, X), (ptx, x)\rangle$

Alice		*Bob*
$(m, inp, out, \Pi, \Lambda, com, \hat{\sigma_B}) \leftarrow ptx$		
if $\mathsf{vrfRanPrf}(\Pi[1], out[1]) = 0$		
return $\bot$		
if $\mathsf{verifyAptSig}(\tilde{\sigma_B}, m, com[1], X) = 0$		
return $\bot$		
$\tilde{\sigma_A} \leftarrow \mathsf{signPrt}(m, sk_A, k_A, \Lambda)$		
	$\xrightarrow{\tilde{\sigma_A}}$	
		$\mathsf{vrfPt}(\tilde{\sigma_A}, m, com[0])$
		$\sigma_{fin} \leftarrow \mathsf{finSig}(\tilde{\sigma_A}, \tilde{\sigma_B})$
	$\xleftarrow{\sigma_{fin}}$	
$tx \leftarrow \mathsf{createTx}(m, inp, out, \Pi, \Lambda, com, \sigma_{fin})$		$tx \leftarrow \mathsf{createTx}(m, inp, out, \Pi, \Lambda, com, \sigma_{fin})$
$x \leftarrow \mathsf{extWit}(\sigma_{fin}, \tilde{\sigma_A}, \hat{\sigma_B})$		
return (tx, x)		**return** tx

Figure 5.5: Adapted Extended Mimblewimble Transaction Scheme - dAptFinTx.

List of Figures

List of Tables

List of Algorithms