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

**Mimblewimble** The Mimblewimble protocol was introduced in 2016 by an anonymous entity named Jedusor, Tom Elvis [Jed16]. 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 [Poe16]. 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 [FOS19, BCL$^+$19].
Compared to Bitcoin, there are some differences in Mimblewimble:

- Use of Pedersen commitments instead of plaintext transaction values
- No addresses. Coin ownership is given by the knowledge of the opening of the coins Pedersen commitment.
- Spend outputs are purged from the ledger such that only unspent transaction outputs remain.
- No scripting features.

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.

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

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. 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 [FOS19].

CHAPTER 2

# Motivation & Objectives

TODO

CHAPTER 3

# Preliminaries

## 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 $p$. $\mathbb{Z}_p^*$ is $\mathbb{Z}_p$
$\{0\}$. $g,\ h$ are adjacent generators in $\mathbb{G}$, whereas adjacent means the discrete logarithm of $h$ in regards to $g$ is not known. Exponention stands for repeated application of the group operation. For two scalars $a$ and $b$ multiplication is defined as $a \ \cdot \ b$. We define addition between two curve points as $g^a \ \cdot \ g^{g^b} \ = \ g^{a \ \cdot \ b}$.

**Definition 3.1** (Hard Relation)**.** 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: [AEE+20]

1. $\mathsf{GenR}\mathit{1}^n$ 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 neglible.

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 $\mathbb{G}$.

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

$$\Phi = (\mathsf{Setup},\ \mathsf{Sign},\ \mathsf{Verf})$$

$\mathsf{Setup}$ 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 $\mathsf{Sign}_m(sk)$ procedure to create a signature $\sigma$ over the message $m$. Parties knowing $pk$ can then test the validity of the signature by calling $\mathsf{Verf}_m(pk,\ \sigma)$ 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 fullfill two security properties

- Correctness: For all messages $m$ and valid keypairs $(sk, pk)$ the following must hold $\mathsf{Verf}_m(pk,\ \mathsf{Sign}_m(sk)) \ = \ 1$
- Unforgability: Note that there are different levels of Unforgability: [GMR88]
  - 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.

**Definition 3.3** (Cryptographic Hash Function)**.** A cryptographic hash function $\mathsf{h}$ is defined as $\mathsf{h}(I) \rightarrow \{0,1\}^n$ for some fixed number $n$ and some input $I$. A secure hashing function has to fullfill the following security properties: [AKDB11]

- 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 $\mathsf{h}$ 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 $\mathsf{h}$ and an input $I$, it should be infeasible to find a different input $I'$ such that $I \ \neq \ I'$ and $\mathsf{h}(I) \ = \ \mathsf{h}(I')$.

**Definition 3.4** (Commitment Scheme)**.** [BBB+18] A cryptographic Commitment is defined by a pair of functions $(\mathsf{Setup}(1^n),\ \mathsf{Commit}(I,k))$. $\mathsf{Setup}$ 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 $\mathsf{Commit}$ 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 $\mathsf{Commit}(I,k) = \mathsf{Commit}(I',k')$ 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 $\frac{1}{2} + \mathsf{negl}(P)$.

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

$$\mathsf{Commit}(I_1,k_1) \cdot \mathsf{Commit}(I_2,k_2) = \mathsf{Commit}(I_1 + I_2, k_1 + k_2)$$

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

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

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

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

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

## 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

### 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

- For two adjacent elliptic curve generators $g$ and $h$ a coin in Mimblewimble is of the form $\mathcal{C} := g^v \cdot h^k,\ \pi$. $\mathcal{C}$ is a so called Pedersen Commitment [Ped91] to the value $v$ with blinding factor $k$. $\pi$ is a range proof attesting to the fact that $v$ is in a valid range in zero-knowledge.

- 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 $\sum v'_i - \sum v_i = 0$ so the sum of all input values has to be 0. (Not taking transaction fees into account)
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:

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

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:

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

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 [FOS19].

**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 [Max13] 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

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 [Poe16].

### 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.

### The Ledger

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: [Jed16]
Assume Alice wants to transfer coins of value $p$ to Bob.

1. Alice first selects input coins $\mathcal{C}_{inp}$ of total value $v \geq p$ that she controls. She than 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 $\sigma$ with the combined secret key $x + \sum x'_i$ and 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 **??** depicts the original transaction flow.
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:

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

| **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 $\pi$ 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 $\sigma$ with $x_{shared}$ |
| | | Create range-proof $\pi$ for $\mathcal{C}_{out}^{B}$ |
| | | Finalize transaction $tx$ |

Figure 3.1: Original transaction building process

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

$$x_{rv} \; := \; -x \; + \; \sum x_i'$$

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 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. [FOS19] 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.

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

Figure 3.2: Salvaged tranction protocol by Fuchsbauer et al. [FOS19]

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

## 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 defined in 3.5, which is specifically tailored for the use in an Atomic Swap scenario in which (at least one side of the swap) uses a two-party protocol to generate transaction signatures and keypairs are fixed beforehand. We will start by defining the general two-party signature creation protocol as it is currently implemented in the Grin Cryptocurrency. We reduce the generated signatures to the general case [Sch89] and thereby prove its security. From this 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 4.2 and finally prove its security in 4.3.

## 4.1 Definitions

**Definition 4.1** (Two Party Signature Scheme)**.** A two-party signature scheme wrt. a hard relation $R$ is an extension of a signature scheme as defined in 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 $pk_{comp} := pk_A + pk_B$ without having to reveal their secret keys to each other. For this we add three procedures to our signature scheme:

$$\Phi_{MP} = (\Phi \mid\mid \mathsf{setupPt},\ \mathsf{signPt},\ \mathsf{vrfPt},\ \mathsf{finSig})$$

- $\mathsf{setupPt}$ takes as an input $1^n$ with $n$ as the security parameter and randomly generates a nonce $k$ as well as a commitment $R$ to the nonce which has to be distributed between Alice and Bob. Note that in this protocol, we assume Alice keypair $(sk_A, pk_A)$, as well as Bob's keypair $(sk_B, pk_B)$ to be fixed so they can't be chosen arbitrarily. This circumstance is important because otherwise, the protocol would be vulnerable to a Rogue-Key Attack [HMP95]. As we have seen in section 3.3.3, the public keys in the Mimblewimble protocol to which the parties are signing are the so-called Kernel elements $K$ which are calculated from transaction elements rather then chosen.

- signPt takes as input a to be signed message $m$ and generates a partial signature with Alice's key $sk_A$ and her nonce $k_A$. This partial signature should only be verifiable by the second party and not necessarily has to be a fully valid signature already on its own.

- vrfPt lets Alice verify Bob's partial signature and vice versa.

- finSig will take the two partial signatures as well as the randomness exchanged between the parties and create a valid signature which can be verified with the regular Verf function under the composite public key $pk_{comp} := pk_A \cdot pk_B$.

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

$$\Phi_{Apt} := (\Phi_{MP} \,||\, \mathsf{setupApt},\ \mathsf{signPtPreSig},\ \mathsf{vrfPtPreSig},\ \mathsf{extWit})$$

- setupApt generates randomly a secret witness $x$, which has to be kept private and is revealed to the other party by receiving the final composite signature. $X$ is distributed to the other party. In an Atomic Swap scenario between Alice and Bob (which we will describe in 5) Bob receiving $X$ should convince himself of its validity. (For example by verifying that there are indeed funds available to him if he gets the secret $x$).

- signPtPreSig allows Alice (knowing $x$) to create a partial pre signature (similiar to as defined in 4.1) which additionally hides the secret $x$ without immediately revealing it. It outputs the partial pre signature, which can be verified to contain $x$ by Bob (knowing $X$).

- vrfPtPreSig makes it possible to verify the validity of a partial pre signature plus that it indeed contains the secret witness of $X$.

- finAptPreSig creates a final valid signature under the composite public key $pk_{comp} := pk_A \cdot pk_B$ by adapting Bobs partial pre signature into Bobs final partial Signature which which we can then construct the final composite one.

- extWit lets Alice extract the secret witness $x$ from the final composite signature, her own partial signature $\sigma_A$ and Bob's partial pre signature $\sigma^B_{apt}$. She does so by first calculating Bob's adapted partial signature in which the witness is already removed. Knowing Bobs adapted partial signature as well as the presignature one, she can subtract those two to receive the secret $x$.

**Definition 4.3** (Secure Adaptor Signature Scheme)**.** As defined by Aumayr et al. in [AEE+20], a secure adaptor signature scheme needs four security properties to be fulfilled:

1. *pre-signature correctness*
2. *adapted existential unforgeability under choosen message attack*
3. *pre-signature adaptability*
4. *witness extractability*

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

**Definition 4.4** (Pre-signature correctness)**.** Similiar to how it is defined in [AEE+20] additionally to *correctness* require our signature scheme to satisfy *pre-signature correctness.*
This property is given if a presignature is generated by signPtPreSig can be completed into a final (partial) signature for all pairs $(x,\ X)$, from which it will be possible to extract the witness computing extWit with the required parameters.

More formally *pre-signature correctness* is given if for every security parameter $n \in \mathbb{N}$, message $m \in \{0,1\}^*$, fixed public keys $(sk_A, pk_A)$, $(sk_B, pk_B)$ witth their composite public key $pk_{comp} := pk_A \cdot pk_B$ and every statement/witness pair $X,\ x)$ in a relation $R$ it must hold that:

$$\Pr\left[\begin{array}{c|l} 1.\ \mathsf{Verf}_{pk_{comp}}(m,\ \sigma_{fin}) = 1 & (k_A,\ R_A) \leftarrow \mathsf{setupPt}(1^n) \\ \wedge & ((k_B,\ R_B),\ (x,\ X)) \leftarrow \mathsf{setupApt}(1^n) \\ 2.\ \mathsf{vrfPtPreSig}_{sk_A,k_A,pk_B,R_B,\sigma_B}(m,\ X) = 1 & \sigma^B_{apt} \leftarrow \mathsf{signPtPreSig}_{sk_B,k_B,pk_A,R_A,x}(m) \\ \wedge & \sigma_A \leftarrow \mathsf{signPt}_{sk_A,k_A,pk_B,R_B}(m) \\ 3.\ (X,\ x' \in R) & \sigma_{fin} \leftarrow \mathsf{finAptPreSig}_{R_A,R_B,x}(\sigma_A,\ \sigma^B_{apt}) \\ & x' \leftarrow \mathsf{extWit}(\sigma_{fin},\ \sigma_A,\ \sigma^B_{apt}) \end{array}\right] = 1.$$

**Definition 4.5** ($\mathsf{aEUF-CMA}$)**.** Additionally to the regular definition of *existential unforgeability under chosen message attacks* as defined for example in [Vau06] we additionally require that it is hard to produce a forged partial signature $\sigma_{prt}$ if the adversary $\mathcal{A}$ gets to know a valid pre signature $\sigma_{prt\_apt}$ w.r.t. some message $m$ and a statement $X$. For the definition of $\mathsf{aEUF-CMA}$-security we define the experiment $\mathsf{forgeAptSig}_{\mathcal{A}}$ for a $PPT$ adversary $\mathcal{A}$ with a fixed keypair $(sk_{\mathcal{A}},\ pk_{\mathcal{A}})$ and an adapted signature scheme $\Phi_{Apt}$ as follows:

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

$\mathbb{S} := \emptyset$

$(k_{\mathcal{A}},\ R_{\mathcal{A}}) \leftarrow \mathsf{setupPt}(1^n)$

$m \leftarrow \mathcal{A}^{\mathcal{O}_{s,sk,pk}(\cdot)}(pk_{\mathcal{A}},\ R_{\mathcal{A}})$

$\sigma_{prt\_apt} \leftarrow \mathsf{signPtPreSig}_{sk,k,pk_{\mathcal{A}},R_{\mathcal{A}},x}(m)$

$\sigma_{prt} \leftarrow \mathcal{A}^{\mathcal{O}_{s,sk,pk}(\cdot)}(\sigma_{prt\_apt})$

$\sigma_{fin} \leftarrow \mathsf{finAptPreSig}_{x,R,R_{\mathcal{A}}}(\sigma_{prt},\ \sigma_{prt\_apt})$

**return** $(m \notin \mathbb{S} \ \wedge\ \mathsf{Verf}_{pk_{\mathcal{A}}\ \cdot\ pk}(m,\ \sigma_{fin}))$

$\mathcal{O}_{s,pk_{\mathcal{A}},R_{\mathcal{A}}}(m)$

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

$((k,\ R),\ (x,\ X)) \leftarrow \mathsf{setupApt}(1^n)$

**return** $\mathsf{signPtPreSig}_{sk,k,pk_{\mathcal{A}},R_{\mathcal{A}},x}(m)$

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

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

**Definition 4.6** (Pre-signature adaptability)**.** Informally *pre-signature adaptability* means that any valid partial pre-signature together with a valid corresponding partial signature from the other party can always be completed into a valid final composite signature by using the finAptPreSig procedure.
Formally for any security parameter $n$, message $m \in \{0,1\}^*$, valid fixed keypairs $(sk_A, pk_A)$, $(sk_B, pk_B)$ and the composite public key $pk_{comp} := pk_A \cdot pk_B$, valid partial pre-signature $\sigma^B_{apt}$ and valid partial signature $\sigma_A$ the following must hold:

$$\Pr\left[\mathsf{Verf}_{pk_{comp}}(m,\ \mathsf{finAptPreSig}_{x,R_A,R_B}(\sigma_A,\ \sigma^B_{apt})) = 1 \ \middle|\ \begin{array}{l} (k_A,\ R_A) \leftarrow \mathsf{setupPt}(1^n) \\ (k_B,\ R_B),\ (x,\ X) \leftarrow \mathsf{setupApt}(1^n) \\ \mathsf{vrfPt}_{sk_B,k_B,pk_A,R_A}(m,\ \sigma_A) = 1 \\ \mathsf{vrfPtPreSig}_{m,sk_A,k_A,pk_B,R_B}(m,\ X)\sigma^B_{apt} \end{array}\right] = 1.$$

This definition might seem very similar to *pre-signature correctness* defined in definition 4.4; however it is slightly stronger as it requires that all partial signatures and partial pre-signatures can be completed into a valid signature, as long as they are valid themselves. This would also include a partial signature that was created maliciously.

**Definition 4.7** (Witness extractability)**.** Informally the *witness extractability* property holds for an adapted signature scheme $\Phi_{Apt}$ computed for the statement $X$ when we can always extract the witness $(x,\ X)$ from the final signature $\sigma_{fin}$, partial presignature $\sigma^{\mathcal{A}}_{prt}$ and corresponding partial signature $\sigma_B$. To formalize this statement we describe and experiment $\mathsf{aExtrWit}_{\mathcal{A}}$ for a $PPT$ adversary $\mathcal{A}$ with a fixed keypair $(sk_{\mathcal{A}},\ pk_{\mathcal{A}})$ and the fixed keypair $(sk_B, pk_B)$ of the second party.

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

$\mathbb{S} := \emptyset$

$(k_{\mathcal{A}},\ R_{\mathcal{A}}) \leftarrow \mathsf{setupPt}(1^n)$

$(m,\ X) \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot),\mathcal{O}_{s,pk_{\mathcal{A}},R_{\mathcal{A}}}(\cdot)}(pk_{\mathcal{A}},\ R_{\mathcal{A}})$

$\sigma^{\mathcal{A}}_{prt} \leftarrow \mathsf{signPtPreSig}_{sk_{\mathcal{A}},k_{\mathcal{A}},pk_B,R_B,x}(m)$

$(\sigma_B) \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot),\mathcal{O}_{s,pk_{\mathcal{A}},R_{\mathcal{A}}}(\cdot)}(pk_{\mathcal{A}},\ R_{\mathcal{A}},\ \sigma^{\mathcal{A}}_{prt})$

$\sigma_{fin} \leftarrow \mathsf{finAptPreSig}_{R_{\mathcal{A}},R_B,x}(\sigma^{\mathcal{A}}_{prt},\ \sigma_B)$

$x' \leftarrow \mathsf{extWit}(\sigma_{fin},\ \sigma_B,\ \sigma^{\mathcal{A}}_{prt})$

$\textbf{return}\ (m \notin \mathbb{S}\ \wedge\ (X,\ x') \notin R\ \wedge\ \mathsf{Verf}_{pk_{\mathcal{A}}\ \cdot\ pk_B}(m,\ \sigma_{fin}))$

$\underline{\mathcal{O}_k(1^n)}$

$(k_B,\ R_B) \leftarrow \mathsf{setupPt}(1^n)$

$\textbf{return}\ R_B$

$\underline{\mathcal{O}_{s,pk_{\mathcal{A}},R_{\mathcal{A}}}(m)}$

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

$\textbf{return}\ \mathsf{signPt}_{sk_B,k_B,pk_{\mathcal{A}},R_{\mathcal{A}}}(m)$

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 $p$, $\mathsf{h}$ is a secure hash function as defined in 3.3 and $m$ is a publicly known message.

- $\mathsf{Setup}$ creates a keypair $(sk, pk)$, the public key can be distributed to the verifier(s) and the secret key has to be kept private.

- $\mathsf{Sign}$ creates a signature consisting of a variable $s$ and generator $g$ raised to the nonce used during the signing process $g^k$.

- $\mathsf{Verf}$ allows a verifier knowing the signature $\sigma$ and the provers public key $pk$ to verify the signatures validity.

| $\mathsf{Setup}(1^n)$ | $\mathsf{Sign}_{sk}(m)$ | $\mathsf{Verf}_{pk}(m,\ \sigma)$ |
|---|---|---|
| 1 : $x \leftarrow\$\ \mathbb{Z}_p^*$ | 1 : $k \leftarrow\$\ \mathbb{Z}_p^*$ | 1 : $s\ :=\ \sigma.s$ |
| 2 : **return** $(sk\ :=\ x,\ pk\ :=\ g^x)$ | 2 : $R\ :=\ g^k$ | 2 : $R\ :=\ \sigma.R$ |
| | 3 : $e\ :=\ \mathsf{h}(m\ \vert\vert\ R\ \vert\vert\ pk)$ | 3 : $e\ :=\ \mathsf{h}(m\ \vert\vert\ R\ \vert\vert\ pk)$ |
| | 4 : $s\ :=\ k\ +\ e\ \cdot\ sk$ | 4 : **return** $g^s\ =\ R^e\ \cdot\ pk$ |
| | 5 : **return** $\sigma\ :=\ (s, R)$ | |

Figure 4.1: Schnorr Signature Scheme as defined in [Sch89]

| $\mathsf{setupPt}(1^n)$ | $\mathsf{signPt}_{sk_A,k_A,pk_B,R_B}(m)$ |
|---|---|
| 1 : $k \leftarrow\$\ \mathbb{Z}_p^*$ | 1 : $e\ :=\ \mathsf{h}(m\ \vert\vert\ R_A\ \cdot\ R_B\ \vert\vert\ pk_A\ \cdot\ pk_B)$ |
| 2 : $R\ :=\ g^R$ | 2 : $s\ :=\ k_A\ +\ e\ \cdot\ sk_A$ |
| 3 : **return** $(k,\ R)$ | 3 : **return** $\sigma_A\ :=\ s$ |

$\mathsf{vrfPt}_{sk_A,k_A,pk_B,R_B}(m,\ \sigma_B)$

1 : $e\ :=\ \mathsf{h}(m\ ||\ R_A\ \cdot\ R_B\ ||\ pk_A\ \cdot\ pk_B)$
2 : $s\ :=\ \sigma_B.s$
3 : **return** $g^s\ =\ R_B^e\ \cdot\ pk_B$

$\mathsf{finSig}_{R_A,R_B}(\sigma_A,\ \sigma_B)$

1 : $s_A\ :=\ \sigma_A.s$
2 : $s_B\ :=\ \sigma_B.s$
3 : **return** $\sigma_{fin}\ :=\ (s\ :=\ s_A\ +\ s_B,\ R\ :=\ R_A\ \cdot\ R_B)$

Figure 4.2: Two Party Schnorr Signature Scheme

The concrete implementation can be seen in figure 4.1. The signature scheme is called schnorr signature scheme, first defined in [Sch89] 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 4.1 in figure 4.2. Note that this two-party variant of the scheme is what is currently implemented in the Grin Mimblewimble Cryptocurrency and will provide a basis from which we will build our adapted scheme.

| **Alice** | | **Bob** |
|---|---|---|
| | Public Parameters: $m,\ 1^n$ <br> Fixed Parameters: $(sk_A, pk_A),\ (sk_B, pk_B)$ | |
| $(k_A,\ R_A) \leftarrow \mathsf{setupPt}(1^n)$ | | |
| | $\xrightarrow{(pk_A,\ R_A)}$ | |
| | | $(k_B,\ R_B) \leftarrow \mathsf{setupPt}(1^n)$ <br> $\sigma_B \leftarrow \mathsf{signPt}_{sk_B, k_B, pk_A, R_A}(m)$ |
| | $\xleftarrow{(\sigma_B,\ pk_B,\ R_B)}$ | |
| $\mathsf{vrfPt}_{sk_A, k_A, pk_B, R_B}(m,\ \sigma_B) = 1$ <br> $\sigma_A \leftarrow \mathsf{signPt}_{sk_A, k_A, pk_B, R_B}(m)$ | | |
| | $\xrightarrow{(\sigma_A)}$ | |
| | | $\sigma_{fin} \leftarrow \mathsf{finSig}_{R_A, R_B}(\sigma_A,\ \sigma_B)$ |

Figure 4.3: Two Party Schnorr Signature Scheme Interaction

We further explain in figure 4.3 how Alice and Bob can cooperate to produce a final signature which fulfills *Correctness* as defined in 3.2. We outline an interaction between Alice and Bob using the procedures defined in figure 4.2 to build the final composite signature $\sigma_{fin}$.

The final signature is a valid signature to the message $m$ with the composite public key $pk_{comp} := pk_A \cdot pk_B$. A verifier knowing the signed message $m$, the final signature $\sigma_{fin}$ and the composite public key $pk_{comp}$ can now verify the signature using the $\mathsf{Verf}$ procedure. The challenge $e$ will be the same because

$$\mathsf{h}(m \mid\mid R \mid\mid pk_{comp}) = \mathsf{h}(m \mid\mid R_A \cdot R_B \mid\mid pk_A \cdot pk_B)$$

*Correctness* is proven by showing that for partial signatures $\sigma_A$ and $\sigma_B$:

$$\mathsf{Verf}_{pk_A \cdot pk_B}(m,\ \mathsf{finSig}_{R_A,R_B}(\sigma_A,\ \sigma_B)) = 1$$

*Proof.* The proof is by showing equality of the equation checked by the verifier by continous substitutions in the left side of equation:

$$g^s = R^e \cdot pk_{comp} \tag{4.1}$$
$$g^{s_A} \cdot g^{s_B} \tag{4.2}$$
$$g^{k_A + e \cdot sk_A} \cdot g^{k_B + e \cdot sk_B} \tag{4.3}$$
$$g^{k_A \cdot e} \cdot g^{sk_A} \cdot g^{k_B \cdot e} \cdot g^{sk_B} \tag{4.4}$$
$$R_A^e \cdot pk_A \cdot R_B^e \cdot pk_B \tag{4.5}$$
$$R^e \cdot pk_{comp} = R^e \cdot pk_{comp} \tag{4.6}$$

**$\mathsf{setupApt}(1^n)$**

$(k,\ R) \leftarrow \mathsf{setupPt}(1^n)$
$x \leftarrow\$\ \mathbb{Z}_p^*$
$X\ :=\ g^x$
**return** $((k,\ R),\ (x, X))$

**$\mathsf{signPtPreSig}_{sk_A, k_A, pk_B, R_B, x}(m)$**

$\sigma_B\ \leftarrow\ \mathsf{signPt}_{sk_A, k_A, pk_B, R_B}(m)$
$s\ :=\ \sigma_B.s\ +\ x$
**return** $\sigma^A_{apt}\ :=\ (s,\ X)$

**$\mathsf{vrfPtPreSig}_{sk_A, k_A, pk_B, R_B, X}(m,\ \sigma^B_{apt})$**

$e\ :=\ \mathsf{h}(m \mid\mid pk_A\ \cdot\ pk_B \mid\mid R_A\ \cdot\ R_B)$
**return** $g^{\sigma^B_{apt}}\ =\ R^e_B\ \cdot\ pk_B\ \cdot\ g^x$

**$\mathsf{finAptPreSig}_{R_A, R_B, x}(\sigma_A,\ \sigma^B_{apt})$**

$\sigma_B\ :=\ \sigma^B_{apt}\ -\ x$
**return** $\mathsf{finSig}_{R_A, R_B}(\sigma_A,\ \sigma^B_{apt})$

**$\mathsf{extWit}(\sigma_{fin},\ \sigma_A,\ \sigma^B_{apt})$**

$\sigma_B\ :=\ \sigma_{fin}\ -\ \sigma_A$
$x\ :=\ \sigma^B_{apt}\ -\ \sigma_B$
**return** $(x)$

Figure 4.4: Fixed Witness Adaptor Schnorr Signature Scheme

□

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

Again in figure 4.5 we show an interaction between Alice and Bob creating a signature $\sigma_{fin}$ for the composite public key $pk_B\ :=\ pk_A\ \cdot\ pk_B$ while Bob will hide his secret $x$ which Alice can extract after the signing process has completed.

## 4.3 Security

We now show that the outlined instantiation is secure with regards to the regular signature scheme definitions 3.2 and the adaptor signature scheme 4.2. We start by proving *correctness* of the scheme by showing that for a partial signature $\sigma_A$ and a corresponding pre-signature $\sigma^B_{apt}$ the following must hold

$$\mathsf{Verf}_{pk_A\ \cdot\ pk_B}(m,\ \mathsf{finAptPreSig}_{R_A, R_B, x}(\sigma_A,\ \sigma^B_{apt}))\ =\ 1$$

| **Alice** | | **Bob** |
|---|---|---|
| | Public Parameters: $m,\ 1^n$<br>Fixed Parameters: $(sk_A, pk_A),\ (sk_B, pk_B)$ | |
| $(k_A,\ R_A) \leftarrow \mathsf{setupPt}(1^n)$ | | |
| | $\xrightarrow{(pk_A,\ R_A)}$ | |
| | | $((k_B,\ R_B),\ (x,\ X)) \leftarrow \mathsf{setupApt}(1^n)$<br>$\sigma^B_{apt} \leftarrow \mathsf{signPtPreSig}_{sk_B,k_B,pk_A,R_A,x}(m)$ |
| | $\xleftarrow{(\sigma_B,\ pk_B,\ R_B,\ X)}$ | |
| $\mathsf{vrfPtPreSig}_{sk_A,k_A,pk_B,R_B,\sigma_B}(m,\ X) = 1$<br>$\sigma_A \leftarrow \mathsf{signPt}_{sk_A,k_A,pk_B,R_B}(m)$ | | |
| | $\xrightarrow{(\sigma_A)}$ | |
| | | $\sigma_{fin} \leftarrow \mathsf{finAptPreSig}_{R_A,R_B,x}(\sigma_A,\ \sigma^B_{apt})$ |

Figure 4.5: Fixed Witness Adaptor Schnorr Signature Interaction

*Proof.* Again the proof is by continous substitutions in the equation checked by the verifier:

$$
\begin{aligned}
g^s &= R^e \cdot pk_{comp\_apt} && (4.7)\\
g^{s_A + s_B - x} & && (4.8)\\
g^{s_A} \cdot g^{s_B - x} & && (4.9)\\
g^{k_A + e \cdot sk_A} \cdot g^{k_B + e \cdot sk_B + x - x} & && (4.10)\\
g^{k_A} \cdot g^{e \cdot sk_A} \cdot g^{k_B} \cdot g^{e \cdot sk_B} & && (4.11)\\
R^e_A \cdot pk_A \cdot R^e_B \cdot pk_B & && (4.12)\\
R^e \cdot pk_{comp} &= R^e \cdot pk_{comp} && (4.13)
\end{aligned}
$$

□

Next we provide a proof that in addition to regular *correctness* also *pre-signature correctness* holds. Note that we have 3 statements to prove, we have already proven that $\mathsf{Verf}_{pk_A \cdot pk_B}(m,\ \sigma_{fin}) = 1$ holds in our instantiation of the signature scheme in the correctness proof 4.3. It remains to prove that $\mathsf{vrfPtPreSig}_{sk_A,k_A,pk_B,R_B,\sigma_B}(m,\ X) = 1$ and $(X,\ x' \in R)$.

*Proof.* For this prove we assume the setup already specified in 4.4. First we prove that the following statement:

$$\mathsf{vrfPtPreSig}_{sk_A,k_A,pk_B,R_B,X}(m,\ \sigma^B_{apt}) = 1$$

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

$$
\begin{aligned}
g^{\sigma^B_{apt}} &= R_B^e \cdot pk_B \cdot X && (4.14)\\
g^{\sigma_B + x} & && (4.15)\\
g^{k_B + e \cdot sk_B + x} & && (4.16)\\
g^{k_B \cdot e} \cdot g^{sk_B} + g^x & && (4.17)\\
R_B^e \cdot pk_B \cdot X &= R_B^e \cdot pk_B \cdot X && (4.18)\\
1 &= 1 && (4.19)
\end{aligned}
$$

We now continue to prove the last equation required:

$$(X,\ x' \in R)$$

To prove correctness we show that $x$ is calculated correctly in $\mathsf{extWit}$:

$$
\begin{aligned}
x &:= \sigma^B_{apt} - (\sigma_{fin} - \sigma_A) && (4.20)\\
& \sigma^B_{apt} - ((s_A + s_B) - s_A) && (4.21)\\
& s_B + x - (s_B) && (4.22)\\
x &:= x && (4.23)\\
& && (4.24)
\end{aligned}
$$

□

TODO Proof for pre-signature adaptability, $\mathsf{aEUF} - \mathsf{CMA}$ and witness extractability.

CHAPTER 5

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

### 5.0.1 Construction Bitcoin side

### 5.0.2 Construction Grin side

CHAPTER 6

# Implementation

## 6.1 Implementation Bitcoin side

## 6.2 Implementation Grin side

## 6.3 Performance Evaluation

CHAPTER 7

# Implementation Security and Privacy Evaluation

## 7.1 Security Evaluation

## 7.2 Privacy Evaluation

CHAPTER 8

# Related and Future Work

## 8.1 Payment Channel Networks on Grin

## 8.2 Payment Channel Networks on Monero

## 8.3 Atomic Swaps With Related Cryptocurrencies

## 8.4 Tumbler Based Atomic Swaps

CHAPTER 9

# Conclusion

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography


[AEE+20] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized bitcoin-compatible channels. Cryptology ePrint Archive, Report 2020/476, 2020. `https://eprint.iacr.org/2020/476`.

[AKDB11] Saif Al-Kuwari, James H Davenport, and Russell J Bradford. Cryptographic hash functions: recent design trends and security notions. *IACR Cryptology ePrint Archive*, 2011:565, 2011.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[BCL+19] Gustavo Betarte, Maximiliano Cristiá, Carlos Luna, Adrián Silveira, and Dante Zanarini. Towards a formally verified implementation of the mimblewimble cryptocurrency protocol. *arXiv preprint arXiv:1907.01688*, 2019.

[FOS19] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: a cryptographic investigation of mimblewimble. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 657–689. Springer, 2019.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[HMP95] Patrick Horster, Markus Michels, and Holger Petersen. Meta-multisignature schemes based on the discrete logarithm problem. In *Information Security—the Next Decade*, pages 128–142. Springer, 1995.

[Jed16] Tom Elvis Jedusor. Mimblewimble, 2016.

[Max13] Greg Maxwell. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*, 2013.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[Poe16] Andrew Poelstra. Mimblewimble, 2016.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[Vau06] Serge Vaudenay. *A classical introduction to cryptography: Applications for communications security*. Springer Science & Business Media, 2006.