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

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.

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 $\mathbb{A}$ 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:

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

Gen 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 fullfill two security properties

- Correctness: For all messages m and valid keypairs (sk, pk) the following must hold $\mathsf{Verf}(pk,\ \mathsf{Sign}(sk, m),\ m)\ =\ 1$
- 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 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: [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 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 $\mathsf{h}(I)\ =\ \mathsf{h}(I')$.

Definition 3.4 (Commitment Scheme)**.** [BBB⁺18] A cryptographic Commitment is defined by a pair of functions $(\mathsf{Commit}(1^n),\ \mathsf{Commit}(I, r))$. 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 $\mathbb{R}_{PP}$ and a commitment space $\mathbb{C}_{PP}$.
The function Commit takes an arbitrary input $I \in \mathbb{I}_{PP}$, and a random value $r \in \mathbb{R}_{PP}$ 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 $\mathbb{A}$ given a valid input $I \in \mathbb{I}_{PP}$ and randomness $r \in \mathbb{R}_{PP}$ the probabilty of finding a $I' \neq I$ and a r' with $\mathsf{Commit}(I, r) = \mathsf{Commit}(I', r')$ is negligible.

- *Hiding:* For a PPT adversary $\mathbb{A}$, commitment inputs $I \in \mathbb{I}_{PP}$, $r \in \mathbb{R}_{PP}$ and a commitment output $C := \mathsf{Commit}(I, r,)$ 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

$$\mathsf{Commit}(I_1, r_1) + \mathsf{Commit}(I_2, r_2) = \mathsf{Commit}(I_1 + I_2, r_1 + r_2)$$

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

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

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

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 $\mathcal{C} := g^v + h^r, \pi$. $\mathcal{C}$ is a so called Pedersen Commitment [Ped91] to the value v with blinding factor r. π 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:

$$\sum \mathcal{C}_{out} - \sum \mathcal{C}_{inp} := \sum h^{v'_i} + g^{r'_i} - \sum h^{v_i} + g^{r_i}$$

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

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

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:

$$\mathcal{E} := g^{(\sum r'_i - \sum r_i)} - h^s$$

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:

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

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

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

must contain valid range proofs for the output coins and $\{\sigma_0\} \mid\mid \{\sigma_1\}$ 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 σ with the combined secret key $x + \sum x'_i$ and and finalizes the transaction as
$$tx := (0, \mathcal{C}_{inp}, \mathcal{C}^{A}_{out} \mid\mid \mathcal{C}^{B}_{out}, (\pi, \mathcal{E} := \sum \mathcal{C}^{A}_{out} + \sum \mathcal{C}^{B}_{out} - \sum \mathcal{C}_{inp}, \sigma))$$
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 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

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

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]

creates her change coins $\mathcal{C}_{out}^{A}$, plus a special output coin $\mathcal{C}_{out}^{sp} := h^p + g^{x_{sp}}$, 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 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 seen 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. We will start by explaining the general two-party signature creation protocol as it 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 then proof its security by showing that all security definitions defined in [AEE+20] hold. In chapter 5 we will then utilize this scheme to build the Atomic Swap protocol.

We start by defining a instantiation of Signature Scheme (see definition 3.2)) which is currenlty employed in Grin, a Mimblewimble based Cryptocurrency. We assume we have a group $\mathbb{G}$ with prime p, h is a secure hash function as defined in 3.3 and m is a publicly known message.
Gen creates a keypair (sk, pk), the public key can be distributed to the verifier(s) and the secret key has to be kept private.
Sign creates a signature consisting of a variable s and generator g raised to the nonce used during the signing process g^k.
Verf allows a verifier knowing the signature σ and the provers public key pk to verify the signatures validity.
Correctness of the Scheme is easy to derive. As s is calculated as $k + e * sk$, when generator g is raised to s, we get $g^{k + e * sk}$ which we can transform into $g^k + e * g^{sk}$, and finally into $r + e * pk$ which is the same as the right side of the equation.

```
Gen($1^n$)
1 : $x \leftarrow\$ \mathbb{Z}_p^*$
2 : return $(sk := x,\ pk := g^x)$
```

```
Sign($m, sk$)
1 : $k \leftarrow\$ \mathbb{Z}_p^*$
2 : $r := g^k$
3 : $e := \mathsf{h}(m \mid\mid r \mid\mid pk)$
4 : $s := k + e * sk$
5 : return $\sigma := (s, r)$
```

```
Verf($m,\ \sigma,\ pk$)
1 : $s := \sigma.s$
2 : $r := \sigma.r$
3 : $e := \mathsf{h}(m \mid\mid r \mid\mid pk)$
4 : return $g^s = r + e * pk$
```

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

Definition 4.1 (Two Party Signature Generation)**.** We now define a multiparty Signature Scheme wrt. to a hard relation R as an extension of the outlined Signature Scheme, which allows us to distribute signature generation for a single public key shared between two parties Alice and Bob. We assume both Alice and Bob generated a keypair $(sk_A, pk_A) \mid\mid (sk_B, pk_B)$ using the Setup procedure. Alice and Bob want to collaborate to generate a signature valid unter 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:

$$\Phi_{MP} = (\Phi \mid\mid \mathsf{setupPt},\ \mathsf{genPtSig},\ \mathsf{vrfPtSig},\ \mathsf{finSig})$$

- setupPt takes as input 1^n with n as the security parameter and randomly generates a nonce k as well as r which has to be distributed between Alice and Bob.
- genPtSig takes as input a to be signed message m, Alice's key sk_A and nonce k_A. as well as pk_B and r_B as provided by Bob. In contrast to the regular case we only have one output paramter instead of two, as the nonce used was already generated by the setupPt procedure. Note that Bob will also call this procedure with his secrets and the parameters provided to him by Alice.
- vrfPtSig lets Alice verify Bobs partial signature and vice verca. Note that the equation the verifier checks is identical to what we have already proven to be correct in the regular case. The only difference is that e is computed differently.
- finSig will take the two partial signatures as well as the randomness exchanged and creates a valid signature which can be verified with Verf under the public key $pk_A + pk_B$.

The final signature is a valid signature to the message m with the composite public key $pk_{comp} := g^{x_{p1}+x_{p2}}$. A verifier knowing the signed message m, the final signature σ_{fin} and the composite public key pk_{comp} can now verify the signature using the Verf procedure. The challenge e will be the same because $\mathsf{h}(m \mid\mid r \mid\mid pk_{comp}) = \mathsf{h}(m \mid\mid r_A + r_B \mid\mid pk_A pk_B)$. Correctness of the final equality check of the Verf procedure is given as follows:

```
setupPt($1^n$)
1 :  $k \leftarrow\$ \mathbb{Z}_p^*$
2 :  $r := g^r$
3 :  return $(k, r)$

genPtSig($m,\ sk_A,\ k_A,\ pk_B,\ r_B$)
1 :  $e := \mathsf{h}(m \mid\mid r_A + r_B \mid\mid pk_A + pk_B)$
2 :  $s := k_A + e * sk_A$
3 :  return $\sigma_A := s$

vrfPtSig($m,\ sk_A,\ k_A,\ pk_B,\ r_B,\ \sigma_B$)
1 :  $e := \mathsf{h}(m \mid\mid r_A + r_B \mid\mid pk_A + pk_B)$
2 :  $s := \sigma_B.s$
3 :  return $g^s = r_B + e * pk_B$

finSig($\sigma_A,\ \sigma_B,\ r_A,\ r_B$)
1 :  $s_A := \sigma_A.s$
2 :  $s_B := \sigma_B.s$
3 :  return $\sigma_{fin} := (s := s_A + s_B,\ r := r_A + r_B)$
```

Figure 4.2: Two Party Schnorr Signature Scheme

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

$$g^s = r + e * pk_{comp} \tag{4.1}$$
$$g^{s_A} + g^{s_B} \tag{4.2}$$
$$g^{k_A + e * sk_A} + g^{k_B + e * sk_B} \tag{4.3}$$
$$g^{k_A} + e * g^{sk_A} + g^{k_B} + e * g^{sk_B} \tag{4.4}$$
$$r_A + e * pk_A + r_B + e * pk_B \tag{4.5}$$
$$r + e * pk_{comp} = r + e * pk_{comp} \tag{4.6}$$

□

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 curve point g^x choosen at the beginning of the protocol. Again we extend our previously defined Signature Scheme with new functions:

$$\Phi_{Apt} := (\Phi_{MP} \mid\mid \mathsf{getAptPtSig}, \mathsf{vrfAptSig}, \mathsf{extWit})$$

- 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. g^x is distributed to the other party. In a Atomic Swap scenario between Alice and Bob (which we will describe in 5) Bob receiving g^x can now convince himself of its validity. (For example by verifying that there are indeed funds available to him if he receives the secret x).

- getAptPtSig allows Alice (knowing x) to create a adapted partial signature (as defined in 4.1) which hides the secret x without immediately revealing it. It outputs the adapted partial signature, which can be verified to contain x by Bob (knowing g^x).

- vrfAptSig makes it possible to verify the validity of an adapted partial signature plus that it indeed contains the secret witness of g^x. Assume Bob provided a adapted partial signature to Alice, Alice calculates the Schnorr challenge e as previously defined, but when comparing equality of the provided partial signature and her computed values she will add g^x to additionally verify the signature containing witness x.

- extWit lets Alice extract the secret witness x from a final composite signature, her own partial signature σ_A and Bobs adapted partial signature σ^B_{apt}. She does so by first calculating Bobs partial signature without the added witness by simply substracting her partial signature from the composite one. Knowing Bobs adapted partial signature as well as the unadapted one she can simply substract the two values to receive the secret x.

Figure **??** shows the concrete construction of the functions. The final signature is created again by calling finSig with the partial signatures computes by Alice and Bob. (Note that the signature Bob provides is adapted)

We proof the correctness of this extended scheme by showing that a verifier calling Verf will be able to verfiy the signature under the composite adapted public key $pk_{comp\_apt} := pk_A + pk_B + g^x$. We assume Alice built her partial signature using the regular genPtSig and Bob called the adapted variant getAptPtSig hiding the secret x in his partial signature.

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

```
setupApt($1^n$)
1 : $x \leftarrow_{\$} \mathbb{Z}_p^*$
2 : return $(x, g^x)$
```

```
(getAptPtSig($m,\ sk_A,\ k_A,\ pk_B,\ r_B,\ x$)
1 : $e\ :=\ \mathsf{h}(m \mid\mid r_A\ +\ r_B \mid\mid pk_A\ +\ pk_B)$
2 : $s\ :=\ k_A\ +\ e\ *\ (sk_A\ +\ x)$
3 : return $\sigma_{apt}^{A}\ :=\ (s)$
```

```
vrfAptSig($m,\ sk_A,\ k_A,\ pk_B,\ r_B,\ g^x,\ \sigma_{apt}^B$)
1 : $e\ :=\ \mathsf{h}(m \mid\mid pk_A\ +\ pk_B \mid\mid r_A\ +\ r_B)$
2 : return $\sigma_{apt}^B\ =\ pk_B\ +\ e\ *\ g^{r_B}\ +\ g^x$
```

```
extWit($\sigma_{fin},\ \sigma_A,\ \sigma_{apt}^B$)
1 : $\sigma_B\ :=\ \sigma_{fin}\ -\ \sigma_A$
2 : $x\ :=\ \sigma_{apt}^B\ -\ \sigma_B$
3 : return $(x)$
```

Figure 4.3: Fixed Witness Adaptor Schnorr Signature Scheme

verifier:

$$g^s\ =\ r\ +\ e\ *\ pk_{comp\_apt} \tag{4.7}$$

$$g^{s_A}\ +\ g^{s_B} \tag{4.8}$$

$$g^{k_A\ +\ e\ *\ sk_A}\ +\ g^{k_B\ +\ e\ *\ (sk_A\ +\ x)} \tag{4.9}$$

$$r_A\ +\ e\ *\ pk_A\ +\ r_B\ +\ e\ *\ (pk_A\ +\ g^x) \tag{4.10}$$

$$r_A\ +\ r_B\ +\ e\ *\ (g^x\ +\ pk_A\ +\ pk_B) \tag{4.11}$$

$$r\ +\ e\ *\ pk_{comp\_apt}\ =\ r\ +\ e\ *\ pk_{comp\_apt} \tag{4.12}$$

□

Definition 4.3 (Pre-signature correctness)**.** TODO [AEE+20]

Definition 4.4 (Pre-signature adaptability)**.** TODO [AEE+20]

Definition 4.5 (Witness extractability)**.** TODO [AEE+20]

Definition 4.6 (Secure Adaptor Signature Scheme)**.** TODO [AEE+20]

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

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.

[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.