



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

Abstract	vii
Contents	ix
1 Introduction	1
2 Motivation & Objectives	5
3 Preliminaries	7
3.1 General Notation and Definitions	7
3.2 Bitcoin	10
3.3 Privacy-enhancing Cryptocurrencies	10
3.4 Scriptless Scripts	15
3.5 Adaptor Signatures	15
4 Two Party Fixed Witness Adaptor Signatures	17
4.1 Definitions	17
List of Figures	21
List of Tables	23
List of Algorithms	25
Bibliography	27

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 [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].

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

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

¹https://harrypotter.fandom.com/wiki/Tongue-Tying_Curse

²<https://grin.mw/>

³<https://beam.mw/>

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 [FOS19].

CHAPTER 2

Motivation & Objectives

TODO

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 . \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 $g^a \cdot g^{g^b} = g^{a + b}$.

Definition 3.1 (Hard Relation[AEE⁺20]). 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. $\text{GenR}(1^n)$ 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

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

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 $(I, h(I))$.
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 (Setup, Sign, Verf) defined as follows:

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

Pedro: write the API of the algorithms in bullet points

$(sk, pk) \leftarrow \text{Setup}(1^n)$ 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 $\text{Sign}_m(sk)$ procedure to create a signature σ over the message m . Parties knowing pk can then test the validity of the signature by calling $\text{Verf}_m(pk, \sigma)$ 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 possession 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

Pedro: Except with negligible probability

- Correctness: For all messages m and valid keypairs (sk, pk) the following must hold $\text{Verf}_m(pk, \text{Sign}_m(sk)) = 1$
- Unforgeability: 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.

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

minor
we normally
omit H

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

- **Collision-Resistance (CR):** Collision-Resistance means that it is computationally infeasible to find two inputs I_1 and I_2 such that $h(I_1) := h(I_2)$ with $I_1 \neq I_2$.
- **Pre-image Resistance (Pre):** In a hash function h that fulfills Pre-image Resistance it is infeasible to recover the original input I from its hash output $h(I)$. If this security property is achieved, the hash function is said to be non-invertible.
- **2nd Pre-image Resistance (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 $h(I) = h(I')$.

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 [BBB⁺18]). A cryptographic Commitment is defined by a pair of functions (**Setup**(1^n), **Commit**(I, k)). **Setup** is the setup procedure, it takes as input a security parameter 1^n and outputs public parameters PP . Depending on PP we define an 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 fulfill 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 probability of finding a $I' \neq I$ and a k' with $\text{Commit}(I, k) = \text{Commit}(I', k')$ 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 $C := \text{Commit}(I, k)$, the probability of the adversary choosing the correct input $\{I, I'\}$ must not be higher than $\frac{1}{2} + \text{negl}(P)$.

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

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

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

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

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 Homomorphic 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 (Setup, Commit) are then instantiated as:

$$\text{Setup}(I^n) := g, h \xleftarrow{\$} \mathbb{G}$$

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

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 particularly the construction of the Atomic Swap protocol defined in ??.

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}, π) , where $\mathcal{C} := g^v \cdot h^k$ a Pedersen Commitment [Ped91] to the value v with blinding factor k . π is a range proof attesting to the fact that v is in a valid range in zero-knowledge.

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

Pedro: not sure whether the point below is required

- 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 $\mathcal{C}_{inp} := (\mathcal{C}_1, \dots, \mathcal{C}_n)$ input coins and $\mathcal{C}_{out} := (\mathcal{C}'_1, \dots, \mathcal{C}'_n)$ 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)

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:

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

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].

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

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:

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

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

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

Then we can build a single merged transaction:

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

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 $\mathcal{C}_{inp}^0 - \mathcal{C}_{out}^0 - h^{s_0} := \mathcal{E}_0$, $\{\pi_0\}$ contains valid range proofs for the outputs \mathcal{C}_{out}^0 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

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

must contain valid range proofs for the output coins and $\{\sigma_0\} \parallel \{\sigma_1\}$ 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 infeasible 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.

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: [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 then creates change coin outputs \mathcal{C}_{out}^A (could be multiple) of total value $v - p$ and then sends \mathcal{C}_{inp} , \mathcal{C}_{out}^A , a valid range proofs for \mathcal{C}_{out}^A , plus the opening $(-p, x)$ of $\sum \mathcal{C}_{out}^A - \sum \mathcal{C}_{inp}$ to Bob.
2. Bob creates himself additional output coins \mathcal{C}_{out}^B 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

Pedro: we need a way to express this clearer

$$tx := (0, \mathcal{C}_{inp}, \mathcal{C}_{out}^A \parallel \mathcal{C}_{out}^B, (\pi, \mathcal{E} := \sum \mathcal{C}_{out}^A \cdot \sum \mathcal{C}_{out}^B - \sum \mathcal{C}_{inp}, \sigma))$$

and publishes it to the network.

Figure 3.1 depicts the original transaction flow.

This protocol however turned out to be vulnerable. The receiver can spend the change coins \mathcal{C}_{out}^A 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.

Pedro: For security, privacy or both?

In detail this reverting transaction would look like:

$$tx_{rv} := (0, \mathcal{C}_{out}^A \parallel \mathcal{C}_{out}^B, \mathcal{C}_{inp}, (\pi_{rv}, \mathcal{E}_{rv}, \sigma_{rv}))$$

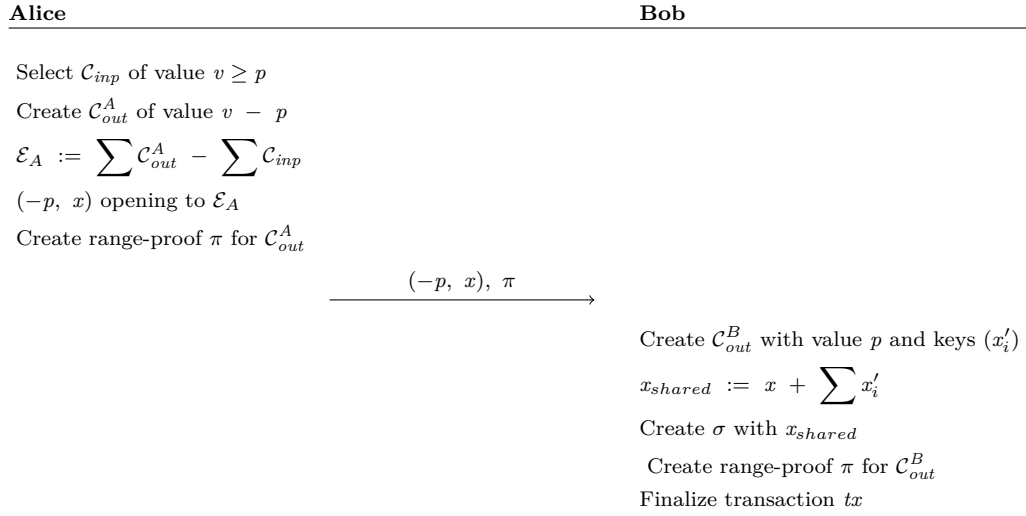


Figure 3.1: Original transaction building process

Pedro: Really nice that you created this protocol :)

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

$$\mathcal{E}_{rv} := \sum \mathcal{C}_{out}^A \parallel \mathcal{C}_{out}^B - \mathcal{C}_{inp}$$

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:

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

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 contain 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.

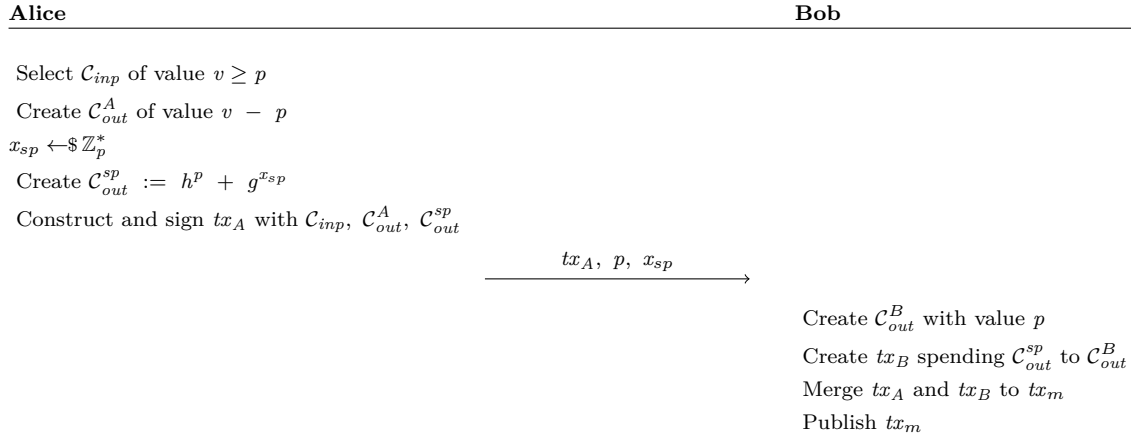


Figure 3.2: Salvaged tranction protocol by Fuchsbauer et al. [FOS19]

Pedro: put labels inside captions

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 chosen 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 therefore 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 3.2.

3.4 Scriptless Scripts

3.5 Adaptor Signatures

3.5.1 Schnorr Signature Construction

3.5.2 ECDSA Signature Construction

Two Party Fixed Witness Adaptor Signatures

In this chapter, we will define a variant of the adaptor signature scheme as explained 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.

This sentence is
hard to parse

Do you really prove security of 2-party signatures?

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 section ?? and finally prove its security in section ??.

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 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 $pk_{comp} := pk_A + pk_B$ without having to reveal their secret keys to each other. For this we add three procedures to our signature scheme:

$$\Phi_{MP} = (\Phi \parallel \text{setupPt}, \text{signPt}, \text{vrfPt}, \text{finSig})$$

- **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 [?]. 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 than 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 $pk_{comp} := pk_A \cdot pk_B$. Note that the procedure will also create a new composite nonce commitment $R := R_A \cdot R_B$ from the partial ones.

Here, you define a scheme which is independent on the realization later (i.e., Schnorr)

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 chosen at the beginning of the protocol. Again we extend our previously defined signature scheme with new functions:

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

- **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 chapter ??) 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 (similar to as defined in definition 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).
- **vrPtPreSig** 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 with 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 σ_A and Bob's partial pre signature σ_{apt}^B . 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 definition 4.2:

Definition 4.4 (Pre-signature correctness). Similar to how it is defined in [AEE⁺20] additionally to *correctness* we require our signature scheme to satisfy *pre-signature correctness*. This property is given when every presignature generated by **signPtPreSig** can be completed into a final (partial) signature for all pairs $(x, X) \in R$, 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) with 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:

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

Definition 4.5 (aEUF – CMA). Additionally to the regular definition of *existential unforgeability under chosen message attacks* as defined for example in [?] we additionally require that it is hard to produce a forged partial signature σ_{prt} if the adversary \mathcal{A} gets to know a valid pre signature σ_{prt_apt} w.r.t. some message m and a statement X .

For the definition of aEUF – CMA-security we define the experiment $\text{forgeAptSig}_{\mathcal{A}}$ for a *PPT* adversary \mathcal{A} with a fixed keypair $(sk_{\mathcal{A}}, pk_{\mathcal{A}})$ and an adapted signature scheme Φ_{Apt} as follows:

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

```

1:  $\mathbb{S} := \emptyset$ 
2:  $(k_{\mathcal{A}}, R_{\mathcal{A}}) \leftarrow \text{setupPt}(1^n)$ 
3:  $m \leftarrow \mathcal{A}^{\mathcal{O}_{s, sk, pk}(\cdot)}(pk_{\mathcal{A}}, R_{\mathcal{A}})$ 
4:  $\sigma_{prt\_apt} \leftarrow \text{signPtPreSig}_{sk, k, pk_{\mathcal{A}}, R_{\mathcal{A}}, x}(m)$ 
5:  $\sigma_{prt} \leftarrow \mathcal{A}^{\mathcal{O}_{s, sk, pk}(\cdot)}(\sigma_{prt\_apt})$ 
6:  $\sigma_{fin} \leftarrow \text{finAptPreSig}_{x, R, R_{\mathcal{A}}}(\sigma_{prt}, \sigma_{prt\_apt})$ 
7: return  $(m \notin \mathbb{S} \wedge \text{Verf}_{pk_{\mathcal{A}} \cdot pk}(m, \sigma_{fin}))$ 
    
```

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

```

1:  $\mathbb{S} := \mathbb{S} \cup m$ 
2:  $((k, R), (x, X)) \leftarrow \text{setupApt}(1^n)$ 
3: return  $\text{signPtPreSig}_{sk, k, pk_{\mathcal{A}}, R_{\mathcal{A}}, x}(m)$ 
    
```

The adapted signature scheme Φ_{Apt} is called aEUF – CMA-secure if

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

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 σ_{apt}^B and valid partial signature σ_A the following must hold:

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

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 were created maliciously.

Definition 4.7 (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} , partial presignature σ_{prt}^A and corresponding partial signature σ_B . To formalize this statement we describe an experiment $\mathbf{aExtrWit}_A$ for a *PPT* adversary \mathcal{A} with a fixed keypair (sk_A, pk_A) and the fixed keypair (sk_B, pk_B) of the second party.

$\mathbf{aExtrWit}_A(n)$

```

1:  $\mathbb{S} := \emptyset$ 
2:  $(k_A, R_A) \leftarrow \text{setupPt}(1^n)$ 
3:  $(m, X) \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot), \mathcal{O}_{s, pk_A, R_A}(\cdot)}(pk_A, R_A)$ 
4:  $\sigma_{prt}^A \leftarrow \text{signPtPreSig}_{sk_A, k_A, pk_B, R_B, X}(m)$ 
5:  $(\sigma_B) \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot), \mathcal{O}_{s, pk_A, R_A}(\cdot)}(pk_A, R_A, \sigma_{prt}^A)$ 
6:  $\sigma_{fin} \leftarrow \text{finAptPreSig}_{R_A, R_B, X}(\sigma_{prt}^A, \sigma_B)$ 
7:  $x' \leftarrow \text{extWit}(\sigma_{fin}, \sigma_B, \sigma_{prt}^A)$ 
8: return  $(m \notin \mathbb{S} \wedge (X, x') \notin R \wedge \text{Verf}_{pk_A \cdot pk_B}(m, \sigma_{fin}))$ 

```

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

$\mathcal{O}_{s, pk_A, R_A}(m)$

```

1:  $(k_B, R_B) \leftarrow \text{setupPt}(1^n)$ 
2: return  $R_B$ 

```

```

1:  $\mathbb{S} := \mathbb{S} \cup m$ 
2: return  $\text{signPt}_{sk_B, k_B, pk_A, R_A}(m)$ 

```

In order to satisfy witness extractability the following must hold:

$$\Pr[\mathbf{aExtrWit}_A(n) = 1] \leq \text{negl}(n)$$

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 is a publicly known message.

- **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.

Setup(1^n)	Sign $_{sk}(m)$	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 : $e := h(m R pk)$	3 : $e := h(m R pk)$
	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 first defined in [Sch89]

- **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.

The concrete implementation can be seen in figure ???. 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 definition 4.1 in figure ???. 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.

We further show in figure ??? 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 $pk_{comp} := pk_A \cdot pk_B$. 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

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

Correctness is proven by showing that for partial signatures σ_A and σ_B :

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

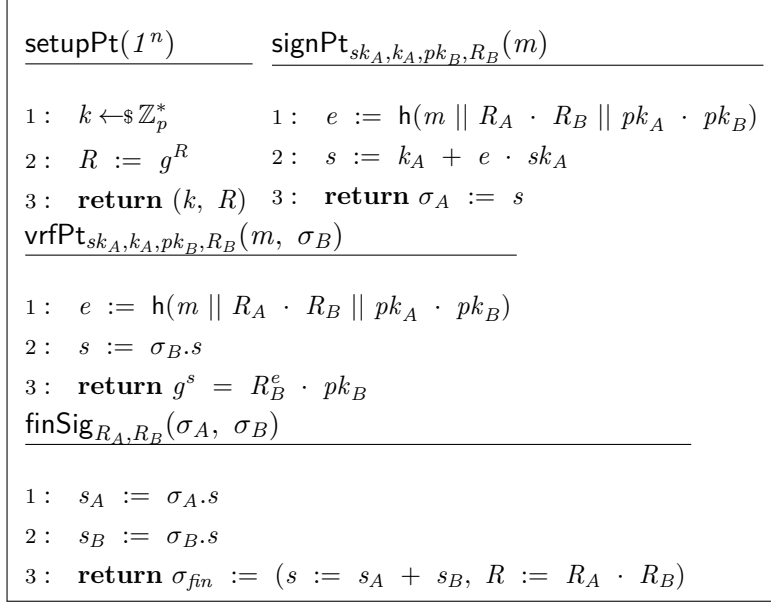


Figure 4.2: Two Party Schnorr Signature Scheme

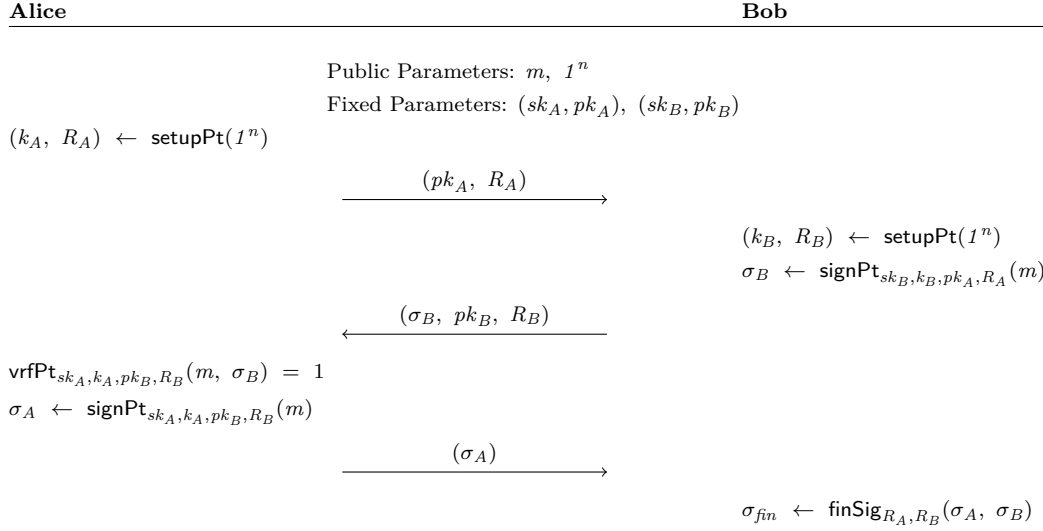


Figure 4.3: Two Party Schnorr Signature Scheme Interaction

<u>setupApt(1^n)</u>	<u>signPtPreSig$_{sk_A, k_A, pk_B, R_B, x}(m)$</u>
1: $(k, R) \leftarrow \text{setupPt}(1^n)$	1: $\sigma_B \leftarrow \text{signPt}_{sk_A, k_A, pk_B, R_B}(m)$
2: $x \leftarrow \mathbb{Z}_p^*$	2: $s := \sigma_B \cdot s + x$
3: $X := g^x$	3: return $\sigma_{apt}^A := (s, X)$
4: return $((k, R), (x, X))$	
<u>vrfPtPreSig$_{sk_A, k_A, pk_B, R_B, X}(m, \sigma_{apt}^B)$</u>	<u>finAptPreSig$_{R_A, R_B, x}(\sigma_A, \sigma_{apt}^B)$</u>
1: $e := h(m \parallel pk_A \cdot pk_B \parallel R_A \cdot R_B)$	1: $\sigma_B := \sigma_{apt}^B - x$
2: return $g^{\sigma_{apt}^B} = R_B^e \cdot pk_B \cdot g^x$	2: return $\text{finSig}_{R_A, R_B}(\sigma_A, \sigma_B)$
<u>extWit($\sigma_{fin}, \sigma_A, \sigma_{apt}^B$)</u>	
1: $\sigma_B := \sigma_{fin} - \sigma_A$	
2: $x := \sigma_{apt}^B - \sigma_B$	
3: return (x)	

Figure 4.4: Fixed Witness Adaptor 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:

$$g^s = R^e \cdot pk_{comp} \quad (4.1)$$

$$g^{s_A} \cdot g^{s_B} \quad (4.2)$$

$$g^{k_A + e \cdot sk_A} \cdot g^{k_B + e \cdot sk_B} \quad (4.3)$$

$$g^{k_A \cdot e} \cdot g^{sk_A} \cdot g^{k_B \cdot e} \cdot g^{sk_B} \quad (4.4)$$

$$R_A^e \cdot pk_A \cdot R_B^e \cdot pk_B \quad (4.5)$$

$$R^e \cdot pk_{comp} = R^e \cdot pk_{comp} \quad (4.6)$$

□

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

Again in figure ?? we show an interaction between Alice and Bob creating a signature σ_{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.

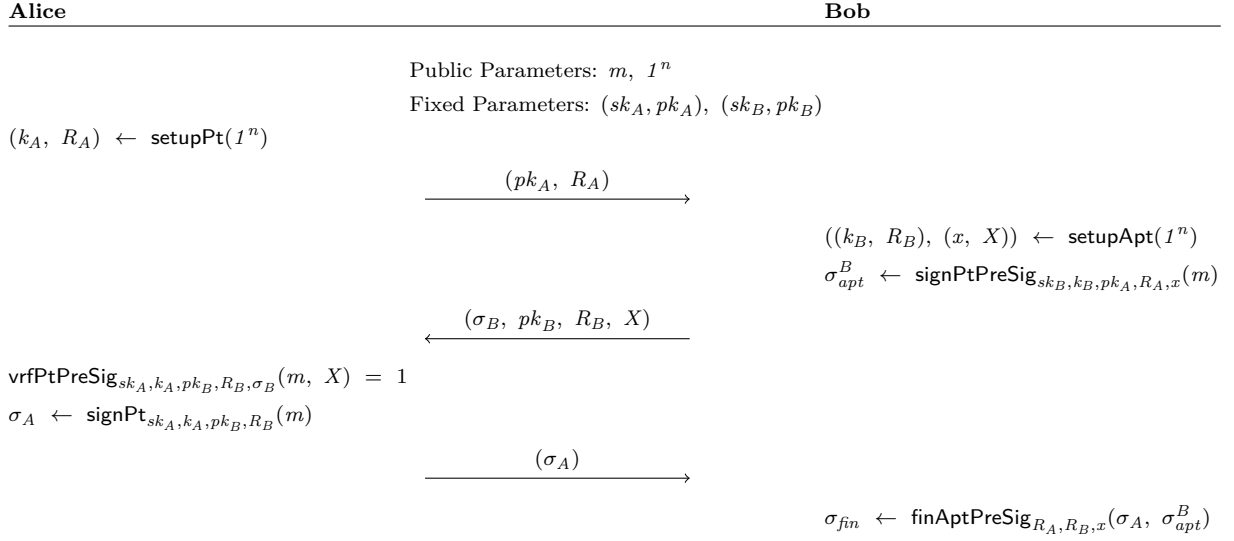


Figure 4.5: Fixed Witness Adaptor Schnorr Signature Interaction

4.3 Security

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

$$\text{Verf}_{pk_A \cdot pk_B}(m, \text{finAptPreSig}_{R_A, R_B, x}(\sigma_A, \sigma_{apt}^B)) = 1$$

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

$$g^s = R^e \cdot pk_{comp_apt} \quad (4.7)$$

$$g^{s_A + s_B - x} \quad (4.8)$$

$$g^{s_A} \cdot g^{s_B - x} \quad (4.9)$$

$$g^{k_A + e \cdot sk_A} \cdot g^{k_B + e \cdot sk_B + x - x} \quad (4.10)$$

$$g^{k_A} \cdot g^{e \cdot sk_A} \cdot g^{k_B} \cdot g^{e \cdot sk_B} \quad (4.11)$$

$$R_A^e \cdot pk_A \cdot R_B^e \cdot pk_B \quad (4.12)$$

$$R^e \cdot pk_{comp} = R^e \cdot pk_{comp} \quad (4.13)$$

□

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

$\text{Verf}_{pk_A \cdot pk_B}(m, \sigma_{fin}) = 1$ holds in our instantiation of the signature scheme in the correctness proof ???. It remains to prove that $\text{vrfPtPreSig}_{sk_A, k_A, pk_B, R_B, \sigma_B}(m, X) = 1$ and $(X, x' \in R)$.

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

$$\text{vrfPtPreSig}_{sk_A, k_A, pk_B, R_B, X}(m, \sigma_{apt}^B) = 1$$

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

$$g^{\sigma_{apt}^B} = R_B^e \cdot pk_B \cdot X \quad (4.14)$$

$$g^{\sigma_B + x} \quad (4.15)$$

$$g^{k_B + e \cdot sk_B + x} \quad (4.16)$$

$$g^{k_B \cdot e} \cdot g^{sk_B} + g^x \quad (4.17)$$

$$R_B^e \cdot pk_B \cdot X = R_B^e \cdot pk_B \cdot X \quad (4.18)$$

$$1 = 1 \quad (4.19)$$

We now continue to prove the last equation required:

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

To prove correctness we show that x is calculated correctly in extWit :

$$x := \sigma_{apt}^B - (\sigma_{fin} - \sigma_A) \quad (4.20)$$

$$\sigma_{apt}^B - ((s_A + s_B) - s_A) \quad (4.21)$$

$$s_B + x - (s_B) \quad (4.22)$$

$$x := x \quad (4.23)$$

$$(4.24)$$

□

TODO Proof for pre-signature adaptability, aEUF – CMA and witness extractability.

List of Figures

3.1	Original transaction building process	14
3.2	Salvaged transaction protocol by Fuchsbauer et al. [FOS19]	15

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.