UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

TNO innovation for life

Balancing privacy and accountability in digital payment methods using zk-SNARKs

Tariq Bontekoe

M.Sc. Thesis

October 2020

Supervisors:
prof. dr. M. J. Uetz (UT)
dr. M. H. Everts (TNO/UT)
dr. B. Manthey (UT)
dr. A. Peter (UT)

Department Applied Mathematics
Discrete Mathematics & Mathematical Programming
Department Computer Science
4TU Cyber Security
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente

Preface

This thesis concludes my seven years (and a month) as a student. During all these years I have certainly enjoyed myself and feel proud of everything I have done and achieved. Not only have I completed a bachelor's in Applied Mathematics, I have also spent a year as a board member of my study association W.S.G. Abacus, spent a lot of time as a student assistant, and have made friends for life.

I have really enjoyed creating this final project, in all its ups and downs, that concludes not only my master's in Applied Mathematics but also that in Computer Science. This work was carried out at TNO in Groningen in the department Cyber Security & Robustness. My time there has been amazing and the colleagues in the department have made that time even better. I am also happy to say that I will continue my time there soon.

There are quite some people I should thank for helping my realise this thesis. First of all, my main supervisor Maarten who helped me with his constructive feedback, knowledge of blockchains and presence at both TNO and my university. I would also like to thank my other supervisors, Bodo and Andreas. They both provided me with useful feedback from their respective backgrounds, which helped my to formulate this thesis as lies before you. My graduation committee is completed by Marc and Jasper, whom I thank for spending their valuable time on reading this quite lengthy document. Finally, many thanks go out to Femmy for encouraging and supporting me during my ups and downs.

I hope you enjoy reading my thesis.

Tariq Bontekoe
Leek, September 21, 2020

Summary

In the last few years the bank card has overtaken cash as the most used payment method. This increased use has also given our banks larger amounts of information on our payment behaviour, whereabouts, and financial situation. In a world where the value of information has increased this puts clients in a weak position with respect to their banks. Legislation also requires banks to use this information to ensure that their clients do not use bank accounts for malicious activities. At the same time many people start to value their privacy more and more, leading to a conflict of interest.

To solve this conflict we present a digital permissioned decentralised anonymous payment scheme. Our scheme provides anonimity for its users, whilst also allowing banks to adhere to current regulations without decreasing anonimity. zk-SNARKs form the basis for our anonimity and implement possibilities for banks to enforce certain regulations. Next to making payments from one user to another, our scheme also allows for banks to (dis)allow their clients access to the payment scheme. Next to his banks can impose a limit on the amount any client can spend anonymously in a certain amount of time. We also present the option for clients to apply a 'time-lock' to the output value of a transaction, making the output value of a transaction unspendable for a certain amount of time.

Finally, we introduce an additional group of actors in our anonymous payment scheme called judges. These judges have the ability to view encrypted transaction details of any transaction that does not adhere to the limits imposed by the scheme. The details can be viewed at any later point in time, as correctness of the values is guaranteed by the verifiable encryption scheme.

This thesis not only presents a construction of the payment scheme, but also provides proofs for correctness and security. Next to this, we discuss the performance of our proof of concept implementation.

Contents

List of acronyms

AML	anti-money laundering
CDD	customer due diligence
$\mathbf{CRH}$	collision-resistant hash function
$\mathbf{EUF\text{-}CMA}$	existential unforgeability against chosen message attack
$\mathbf{IK\text{-}CCA}$	key indistinguishability under chosen ciphertext attack
$\mathbf{IND\text{-}CCA}$	indistinguishability under chosen ciphertext attack
I2P	Invisible Internet Project
$\mathbf{KDF}$	key derivation function
KYC	know your customer
MAC	message authentication code
NIZK	non-interactive zero-knowledge proof
$\mathbf{NP}$	non-deterministic polynomial time
$\mathbf{PRF}$	pseudo random function
QAP	quadratic arithmetic program
RingCT	ring confidential transactions
R1CS	rank-1 constraint system
SAVER	SNARK-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization
SNARG	succinct non-interactive argument
SNARK	succinct non-interactive argument of knowledge
$\mathbf{SUF\text{-}CMA}$	strong existential unforgeability against chosen message attack
$\mathbf{SUF\text{-}1CMA}$	strong existential unforgeability against one-time chosen message attack
UTXO	unspent transaction output
ZKP	zero-knowledge proof
zk-SNARK	zero-knowledge succinct non-interactive argument of knowledge
zk-STARK	zero-knowledge succinct transparent argument of knowledge

Chapter 1

Introduction

While in the previous century most of our transactions were still made with cash, nowadays virtually no one carries cash around. Most people perform all their monetary transaction via their plastic bank card, mobile banking, or another form of digital transactions. Using these methods, the details of every single transaction are stored by a person's bank. This gives the bank great insight in a client's shopping behaviour, whereabouts, and financial situation. Even though this might be acceptable in some situations, it is not desirable as it gives banks a too strong position with respect to their clients. As a client, one might wonder if a bank actually requires all the information about every transaction. Moreover, it is generally not insightful to the client how his or her personal information is processed. Giving control of the data involved in monetary transaction back to the clients would clearly be beneficial to their privacy. A recent approach to return the power over financial transactions to the clients, without requiring them to pack their wallets with physical money, are so called cryptocurrencies.

At the end of 2008 a person, or group of persons, published under the name of Satoshi Nakamoto a paper called: “Bitcoin: A Peer-to-Peer Electronic Cash System” [1]. A couple of months later, in January 2009, the first operational Bitcoin software was launched. The genesis (initial) block got mined and the foundation for many new and different decentralised ledger technologies was laid.

In the years that followed, one after the other virtual coin was introduced, most of them having similar behaviour to Bitcoin. These coins however did not focus on user privacy. Their main goal was decentralisation of financial power. In most cases the privacy of users was virtually non-existent as every transaction is publicly available under a, generally traceable, pseudonym. Eventually, this lead to the introduction of so called privacy coins, i.e. cryptocurrencies that provide, up to some level, transactional privacy for its users. Unfortunately, these currencies are to some extent in conflict with current anti-money laundering (AML) regulations, which causes some governments to take actions against these privacy coins [2]–[5]. In this thesis a solution that solves the privacy problems with current (AML) regulations in mind is proposed.

1.1 Related work

Before deciding on a solution direction for the problem as described above, we take a brief look at existing solutions for this and similar problems. In Section 3 we discuss these solutions in more detail. There are two main types of solution directions to be considered: centralised and decentralised.

An early, well-known centralised solution for anonymous payments is **Digicash** [6]. This company was founded by David Chaum and provided untraceable e-cash throughout the early nineties. The company's solution was based on Chaum's article on “Blind Signatures for Untraceable Payments” [7]. Chaum's solution required a client and bank to create a new note of a fixed value together. The client then creates a random value x and asks the bank to sign this, without revealing x. The bank first deducts the fixed value from the client's account balance and then issues a signature σ for x. The client morphs this signature σ to an anonymous signature σ' that is still valid. The client can now spend the note anonymously by passing σ' to another client, called the receiver. Finally, the receiver can cash in this note by showing σ' to the bank. In this process the bank and receiver learn nothing about the identity of the sender, who remains anonymous.

A disadvantage of the above problem is that anyone can forge arbitrary notes after the secret signature key of the bank is compromised. **Sander and Ta-Shma** [8] propose a (centralised) solution to this problem. Instead of issuing certificates, the bank should maintain a public Merkle tree of commitments to notes. Every time a client requests a new note of fixed value, the client sends a commitment, to a secret x, to the bank. Upon receiving a new commitment, the bank subtracts the fixed value from the client's account and includes the commitment in the public Merkle tree. The client can now send the note to a receiver by sending x along with a proof, in zero-knowledge, that there is a commitment to x in the public Merkle tree. Finally, the receiver can cash in this note by showing x and the proof to the bank. In this process the bank and receiver learn nothing about the identity of the sender.

A big drawback of both methods described above, is that the notes are not divisible. In other words, the notes have fixed value and can not be split or combined into one new note. This is not only inconvenient, but also reveals the values spent and received. Possibly, enough information is leaked to reveal a client's identity to the bank or another client.

Another solution frequently mentioned when talking about privacy-friendly payments is **GNU Taler** [9], [10]. This is a decentralised payment method, with some central control, that does provide it's users with divisible 'notes', and is also auditable to some extent. GNU Taler also provides anonymity for the payer of a transaction through an extension of Chaum's online e-cash [7]. A big disadvantage is that anonymity of the payee is not present. This is mainly due to a design choice, where the payer is expected to be an individual who wants to remain anonymous. The payee is expected to be a merchant, who should be auditable and does not require anonymity. An important difference with the cryptocurrencies that we discuss below is that GNU Taler is always backed by an existing fiat currency, and is not a new currency.

A disadvantage of centralised solution is that all trust lies in one party, there is a single point of failure. Moreover, in the case of transactions between clients of different banks a central solution might not even be possible at all. To address these problems we will take a look at the above mentioned subcategory of cryptocurrencies called privacy coins. These coins are divisible, and often provide full anonymity, in contrast to GNU Taler. We will look into some successful techniques currently used in privacy coins. We distinguish two types of privacy coins, the ones based on obfuscation and others based on cryptography. This distinction is not always completely strict, i.e. combinations are possible, but it does give an insight into how privacy is provided.

In privacy coins of the obfuscation type the source of a transaction is obfuscated, i.e. the sender/receiver is hidden amongst a subset of the entire user group. This ought to make it more difficult for an observer to determine the sender or receiver of a transaction. Because the sender and/or receiver are only hidden amongst a subset of all the users there is still some structure present on the blockchain. Specifically, an observer can still construct a transaction graph of the senders and receivers of all transactions, with one small difference. The edges of this graphs now represent multiple possible sender-receiver combinations per transaction instead of one certain sender-receiver pair. Examples of obfuscation type cryptocurrencies are **Monero** [11], **Verge** [12], and **Grin** [13]. The techniques used in these currencies are amongst others: traceable ring signatures [14], Tor, Invisible Internet Project (I2P), (Pedersen) commitments and zero-knowledge (range) proofs.

The cryptocurrencies discussed above provide privacy by obfuscating parts of the transaction graph. It is however also possible to hide the transaction graph completely, i.e. to hide the sender/receiver of a transaction among all users of the currency. When the transaction graph is completely hidden, a transaction is no longer linkable to a (small) set of people of which one is the actual sender or receiver, since there is no link at all. In the unspent transaction output (UTXO) model, this implies that the anonymity set of a transaction input is simply every transaction in the entire history of the ledger. Transaction graph hiding privacy coins provide a stronger level of privacy and are thus more interesting for our solution. There are two well-known transaction graph hiding privacy coins, called **Zerocoin** [15] and **Zerocash** [16]. Zerocash can be seen as the more mature and more practical version of Zerocoin. Zerocash, or **Zcash** uses a cryptographic primitive known as a zk-SNARK, a type of zero-knowledge proof. ZCash does not only provide divisibility of the currency and sender and receiver anonymity, but also allows for direct payments, whilst hiding transaction amounts.

Since we want to provide the highest possible level of anonymity for users of our digital payment scheme, we will disregard the techniques used in the privacy coins of the obfuscation type. Next to this, the discussed centralised solutions also had a couple of disadvantages when compared to the decentralised solutions: single point of failure, required cooperation between different banks. The transaction graph hiding privacy coins on the other hand perfectly fit our goal of strong anonymity and decentralised payment and will thus be further considered by us. As Zerocash is an improved version of Zerocoin and, as far as the author knows, has no competitors with a similar level of anonymity, we deem the techniques used for the Zerocash protocol to be an interesting starting point for our solution.

1.2 Research goal and questions

In this thesis we discuss a new distributed ledger based protocol for decentralised anonymous transactions. This new protocol is the first step in the realisation of a digital decentralised anonymous payment system that adheres to existing regulations for digital transactions. It can also be implemented on top of the transaction channels that clients of a bank currently use. The scheme will be implemented on top of a permissioned blockchain, to allow for customer due diligence or know your customer (KYC) ‘at the gate’ and prevent misuse of the provided anonymity. On this permissioned blockchain a select set of actors, i.e. participating banks and other financial institutions, will have the role of administrator and gatekeeper. In order to maintain strong anonymity for users we use zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs).

The goal of this research is to make a first step in the development of a decentralised, yet permissioned digital payment method that keeps transaction details private and is compliant with AML regulations. In particular a solution to the following research questions in the setting as mentioned above will be presented.

1. *How can zero-knowledge proof systems help in realising privacy in digital transactions?*

2. *How can the amount of value spent, during a certain time frame, be limited without infringing on the provided privacy?*
3. *How can transaction details be enclosed in a transaction, such that they can only be viewed by one select group of actors at a later point in time?*
4. *How can transferred value be locked for a certain amount of time, before being transferred again?*

1.3 Our contribution

Because the Zerocash protocol already largely fits our problem setting, we choose to adapt and improve this protocol to suit our goals instead of devising an entirely new protocol. To address the missing functionality for our use case, this thesis introduces the following main contributions:

Firstly, we present an **account-based** and **permissioned** version of the decentralised anonymous payment scheme as defined in Zerocash [16]. This implies that any client that wants to perform a transaction using the payment scheme must be registered. For this registration the potential client must be approved by a so called administrator, e.g. the bank or any other financial institution. This administrator can for example perform an identity check and a KYC check. When the administrator is satisfied, the client can add his or her account to the blockchain and publish new transactions. Moreover, we also transform the UTXO-model based protocol of Zerocash to an account based model, or actually a combination of the UTXO-model and the account-based model. This allows for more possibilities regarding audibility, performing KYC, and limiting spending behaviour.

Secondly, we show how the **conversion** between existing fiat currency such as Euros or American Dollars and its anonymous counterpart can be achieved. We emphasise that this anonymous counterpart is *not* a new currency. It is simply a digital and anonymous equivalent with the same value as it's real-world counterpart. This conversion makes use of the fact that both banks and their clients are involved in this payment scheme, such that clients can convert some of their regular account balance to anonymous virtual notes with the same value.

Thirdly, we add **auditability** functionalities to the decentralised anonymous payment scheme, without giving in on the provided level of anonymity. Using the above mentioned account-based functionality we are able to limit the amount of value that any user can transfer anonymously in a certain fixed time frame. Next to this, we present a subsystem that clients can use to transfer value beyond this limit. This subsystem requires the client to include encrypted transaction details in such a transaction. The used encryption scheme is a verifiable encryption scheme in the sense that correctness of the ciphertext can be guaranteed without knowing the plaintext. This allows us to have a designated set of judges that can view the plaintext of such a transaction at any later point in time if this is required.

Lastly, we introduce anonymous **timelocks**. Anonymous in the sense that only the sender and the receiver of a transaction are aware of the timelock being in place. Since timelocks are an essential feature for making decentralised payment schemes more efficient – consider for example the Lightning Network [17] – we decided to also look into introducing this feature. The practical application of these timelocks is out of the scope of this research and will be left for possible future work.

Next to presenting the specifics of our newly designed protocol, we also validate our work by presenting a **proof of concept implementation** as well as a set of **security and completeness proofs**. The proof of concept implementation allows us to validate that the protocol works as intended and could be implemented and used in practice. Additionally, we use the implementation to measure the efficiency of our system, in particular the intensive computations caused by zk-SNARKs. The set of security and completeness proofs shows, more rigorously, that our payment scheme actually accomplishes secure and anonymous payments between registered clients of financial institutions.

1.4 Thesis structure

The rest of this thesis is structured as follows. Chapter 2 provides more background on the used notation and cryptographic techniques, including zk-SNARKs, verifiable encryption, and Merkle trees. In Chapter 3 we present a detailed overview of the briefly discussed existing solutions for anonymous transactions. A step-by-step sketch of our solution and the reasoning behind it is given in Chapter 4. Subsequently, we define our anonymous payment scheme in Chapter 5 and construct it in Chapter 6. A concrete implementation of the protocol, together with performance measures are discussed in Chapter 7. We conclude the thesis in Chapter 8, with a brief discussion and mention some suggestions for future work.

Chapter 2

Preliminaries

We begin this chapter with an explanation of our use of notation. Moreover, we provide a reference list of the relevant terms and variables. Subsequently, we provide an overview of existing (cryptographic) building blocks that form the basis of our protocol. We present a detailed description of the three most important building blocks: zk-SNARKs, verifiable encryption, and Merkle trees. Moreover, we briefly touch upon other, more common, cryptographic building blocks such as commitment and signature schemes.

2.1 Notation and terminology

In this section we discuss the notation that will be used throughout this thesis. We also provide a list of the used functions with in explanation 2.3, a glossary of relevant terminology 2.2, and an overview of the variables that are used in the protocol 2.1.

In the context of zk-SNARKs and verifiable encryption we will work over bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g$ with the following properties:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are all groups of prime order p;
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, also known as the pairing (function);
- g is the generator for $\mathbb{G}_1$, h is the generator for $\mathbb{G}_2$, and $e(g, h)$ is the generator for $\mathbb{G}_T$
- Computing group operations, evaluating the pairing function, deciding membership of groups, deciding equality of group elements, and sampling generators of groups can all be done efficiently.

We will use the multiplicative notation in all groups, i.e. we will write g^a and not $a \cdot G$, and $g^a \cdot g^b = g^{(a+b)}$.

Next to the above group notation, we also use some other specific notation in this thesis. The $\|$ operator will be used for concatenating two strings, i.e. $0\|1 = 01$. The exponent notation in strings will be used for repetition of elements, $0^5 = 00000$. For (uniform) random element selection in a group we use the $\in_R$ operator. In parsing vectors we will use the symbol $*$ to denote the remainder of the variables that is not unwrapped from the vector. For example, in the case that we only want to parse x from the vector $v = (x, y, z)$, we write parse v as $(x, *)$.

In the context of zk-SNARKs we will the use the vector x to denote the public inputs, the vector a contains all witnesses or auxiliary inputs. We will include this vectors in a statement R that is to be proved as follows: *Given x, the prover knows a, such that the following statement(s) hold: R.* We give a detailed description in Section 2.2 for those unfamiliar with zk-SNARKs.

λ	security parameter	pp	public parameters
sk	secret key	C	arithmetic circuit
pk	public key	vk	verification key
s	randomness	x	public inputs
a	auxiliary inputs	cm	commitment
π	zero-knowledge proof	m	message
σ	signature	tx	transaction
rt	Merkle root	v	value (monetary)
info	extra info	note	(anonymous) note
pos	position of leaf in Merkle tree	path	Merkle path
η	nullifier for a note	mem	memory (cell)
b	binary value/bit	μ	nullifier for a memory cell
data	field with encrypted secrets	cred	user credentials
k	hash of public signature key	κ	message authentication code (MAC)
t	(block) time	c	aggregated outgoing value

Table 2.1: List of variables with meaning.

To denote users in the protocol we will always user the letter u, and when talking about multiple users we will user u_A for the one (Alice) and u_B (Bob) for the other. The other protocol variables are listed in Table 2.1, we use a subscript to denote a sub-type of a variable, e.g. $\mathrm{path}_{\text{note}}$ and $\mathrm{path}_{\text{mem}}$ are both Merkle paths but in different trees, respectively the Note and Memory tree. A superscript on a variable is used to denote a different version of the same variable. For example we use $v_{\text{note}}^{\text{old}}$ to denote the note value of the old note and we use $v_{\text{note}}^{\text{new}}$ to denote the note value of the new note. If we desire to denote a variable in general, without a specific sub-type we use the asterisk $*$, i.e. rt_* denotes a Merkle tree root in any of the Merkle trees.

transaction	transfer of value from one entity to another
blockchain/ (distributed) ledger	data structure consisting of blocks that store all past transactions
user/client	person using the payment scheme, must be client of a bank
admin(istrator)	bank or financial institution (partly) controlling the blockchain
judge	actor allowed to view transaction details, possibly an actual judge
sender	payer, user that pays with the input of a the transaction
receiver	payee, user that receives the output of a transaction
fiat currency	regular currency such as Euros or Dollars, cash or digital
(anonymous) note	virtual representation of fiat currency in our payment scheme
conversion	trading fiat currency for an (anonymous) note or vice versa
account balance	aggregated value of fiat currency/notes in a client's bank account/on the blockchain
public parameters	set of system parameters available for all users/admins
(public-private) key pair	two linked keys of which one is publicly available and one must be kept secret
address key pair	key pair used to identify a target address and send transactions
encryption key pair	key pair used to encrypt/decrypt some transaction fields
signature key pair	key pair used to sign transactions/verify signatures
credentials	all key pairs of a user/admin, or (commitment to) the address key pair
memory (cell)	(commitment to) account balance and possible other account specific values
nullifier	unique value for each note to prevent double spending
plaintext	message that is to be encrypted/has been decrypted
ciphertext	encrypted plaintext
arithmetic circuit	encoding of the statements for a zk-SNARK proof
Merkle tree/ (binary) hash tree	data structure containing all notes on the blockchain
Merkle root	top-most node of the Merkle tree, public value

Table 2.2: Glossary of relevant terminology.

For functions we will use the superscript to denote a specific instantiation thereof. For example, $\mathrm{COMM}^{\text{note}}$ is the note commitment function, whereas $\mathrm{COMM}^{\text{mem}}$ represent the commitment function for a memory cell. Every now and then a function will also have a subscript, this subscript will be a variable that is either used as key or seed.

$\mathbf{COMM}$	commitment function	$\mathbf{PRF}$	pseudo-random function
$\mathbf{CRH}$	collision-resistant hash	$\mathbf{KDF}$	key-derivation function
$\mathbf{Prove}$	functions that creates a zk-SNARK proof	$\mathbf{Verify}$	function that verifies a signature or zk-SNARK proof
$\mathbf{KeyGen}$	key generation function for signature scheme	$\mathbf{Sign}$	generates signature for a message
$\mathbf{Setup}$	setup function, can be used for zk-SNARK, encryption, or signature scheme	$\mathbf{Enc}$	encryption function that transforms plaintexts into ciphertexts
$\mathbf{Dec}$	decryption function that transforms a ciphertext into the original plaintext		

Table 2.3: List of functions.

2.2 zk-SNARKs

In this section we give an overview of the workings of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) and dive deeper into one specific proof system that we deem most suitable for our scheme. A description of general zero-knowledge proofs is provided in Appendix A.2. In that Appendix we also provide the definitions that should hold for any type of zero-knowledge proof (ZKP): *completeness*, *soundness* and *zero-knowledgeness*. For the unfamiliar reader we strongly advise to read that before continuing with this section.

The notion of a zk-SNARK, or zero-knowledge SNARK, was first mentioned in 2011 by Bitansky et al. [18]. A succinct non-interactive argument of knowledge (SNARK), named after a creature in one of Lewis Carroll's novels, is a succinct non-interactive argument (SNARG) of knowledge and is a well-used and relatively efficient technique that can be used in proving knowledge. A SNARK is a *succinct* non-interactive argument attesting to the fact that there exists a witness that will evaluate a statement to true, and moreover the prover also knows this witness. It is similar to a non-interactive zero-knowledge proof (NIZK), in the sense that it requires only one extra condition:

- **Succinctness.** Let R be a polynomial time decidable binary relation. For any pair $(x, a) \in R$ we call y the instance and a the witness, where $y = (M, x, t)$, $|w| \leq t$, and M is Turing machine that accepts (x, a) after at most t steps. Now, for a proof π returned by the prover for any statement $(y, w) \in \mathcal{R}$, the proof size as well as the verification time is bounded by

  $$p(\lambda + |y|) = p(\lambda + |M| + |x| + \log t),$$

  where k is the security parameter and p a universal polynomial independent of $\mathcal{R}$.

This succinctness provides the users with upper bounds on the verification time and the memory needed to actually construct, relay, and store the proof. A disadvantage of most SNARKs is that in order to achieve succinctness, a rather large common reference strings is constructed up front, where all users need to trust that this string is indeed constructed correctly. The size of the common reference string is generally not a very big problem, although for larger statements this could end up being several hundreds of Megabytes in size. The fact that the string has to be generated by a trusted party, or group of trusted parties, might be more problematic to some users. In the case that no such trusted party can be found, the zero-knowledge succinct transparent argument of knowledge (zk-STARK) could be considered as an alternative. However, zk-STARKs are currently too computationally expensive, in the sense that the computational efforts required for the prover and verifier are just too large for real-world applications. We will therefore not consider these zk-STARKs and focus on zk-SNARKs, assuming that a trusted party or group can be found to perform the setup for the proof system.

Most SNARKs are constructed using the same set of steps, we will later on explain these steps here. In the first step, a boolean or arithmetic circuit is constructed that can be used to verify the statement. After that, the circuit is transformed into the rank-1 constraint system (R1CS) language consisting of three vectors. These vectors can then be encoded into polynomials, a so called quadratic arithmetic program (QAP). This QAP can be securely evaluated using blind evaluation of polynomials using a homomorphic hiding scheme. This homomorphically hidden encoding serves as the proof, and the evaluation of these polynomials at randomly selected points is used to empirically verify (with high probability) the correctness of the polynomials. Different SNARKs use different ways of achieving this, but most SNARKs take an approach that is mostly similar to the one sketched here, in order to achieve succinct non-interactive proofs. More details on arithmetic circuits and QAPs are given in Appendix A.3.

There are several ways to distinguish SNARKs, we choose to adopt the same distinction as is made in the ZKProof Community Reference document [19]. This

	Preprocessing	**Non-preprocessing**
Non-universal	QAP-based	*unknown (yet)*
Universal	vnTinyRAM or Bulletproofs (with explicit CRH)	Bulletproofs (with PRG-based CRH generation)
Universal and scalable	*impossible*	Recursive composition of SNARKs

Table 2.4: Distinction in NIZK's based on setup phase.

distinction, together with some examples, is shown in Table 2.4. The difference between the preprocessing and non-preprocessing ZKP's is determined by the runtime and output size of the setup algorithm, i.e. if both are at most polylogarithmic the ZKP is non-preprocessing, otherwise it is preprocessing. Furthermore, if a SNARK is non-universal the setup needs the constraint system as input. A universal SNARK only needs a size bound of the circuit as input, when this is also not necessary, i.e. setup is only dependant on the security parameter, we call a SNARK universal and scalable.

We choose to use a non-universal, preprocessing SNARK for two reasons. The first reason being that preprocessing SNARKs have lower proving and verification time than non-preprocessing SNARKs. Secondly, we do not require the more complex, and therefore also less efficient, universal SNARK because the statements that we will want to prove are known up front. The specific proving system that we will use is known as *Groth16*, named after the author and year of the original publication [20], since it is most suitable for our research. As far as the author of this thesis knows, it is more efficient, considering memory usage as well as proof generation and verification time, than any other non-universal preprocessing SNARK out there.

The *Groth16* proving system is a zk-SNARK with perfect completeness and perfect zero-knowledgeness. Next to this, it has statistical knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations. It is a pairing-based non-interactive proof system that is used to construct proofs for $\mathbb{F}$-arithmetic circuit satisfiability. The most relevant details of arithmetic circuits and their relation with quadratic arithmetic programs are given in Appendix A.3. For more information on this and other related topics we refer the reader to [21].

The proof system can be used to construct proofs over relations of the form

$$R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)),$$

where the bit length of p is equal to the security parameter λ. The arithmetic circuit, with base field $\mathbb{F} = \mathbb{Z}_p$, is encoded using a quadratic arithmetic program encoding of the relation R. The arithmetic program describes a satisfiability problem, with m

variables and n equations, over the field $\mathbb{Z}_p$ with ℓ input variables $a_i \in \mathbb{Z}_p(1 \leq i \leq \ell)$ and $m - \ell$ auxiliary variables $a_i \in \mathbb{Z}_p, (\ell + 1 \leq i \leq m)$, with $a_0 = 1$ as

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X),$$

for some degree $n - 2$ polynomial $h(X)$.

The variables $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ represent the bilinear groups that are used in the proof scheme. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, e is a bilinear map that takes one element of $\mathbb{G}_1$ and one of $\mathbb{G}_2$ to the target group $\mathbb{G}_T$. Moreover g is the generator of $\mathbb{G}_1$, h of $\mathbb{G}_2$, and $e(g, h)$ of $\mathbb{G}_T$. Finally, we require the generic group operations, which also includes e, to be efficiently computable.

Groth16 is compromised of 3 functions, a *setup* function that pre-computes the common reference string that is generated once and is used in every proof and verify session for the same circuit. The second function is the *prove* function, with as input the public inputs, auxiliary variables, and common reference string. After receiving the inputs, *prove* outputs three group elements (in $\mathbb{G}_1$ and $\mathbb{G}_2$) that together form the proof. The third and final function is the *verify* function that, on input the three proof elements, computes two elements in $\mathbb{G}_T$ using the pairing function e and checks equality of both elements. For an exact definition of these functions and a security proof of the system we refer the reader to the original publication [20].

2.3 Verifiable encryption and SAVER

To ensure auditability of conspicuous transactions we will be using a verifiable encryption scheme known as SNARK-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization (SAVER). As far as the authors are aware, SAVER is the first SNARK-friendly[1] encryption scheme out there. Next to providing us with verifiable encryption, the scheme has more features. SAVER has verifiable decryption, rerandomisation, and is additively homomorphic. We only need the verifiable encryption and decryption features, so we will focus on that. A verifiable encryption scheme is a scheme in which one can prove certain properties of a message m, when only given the encryption c of m. We can use this in our use case, be encrypting transaction details such as address keys whilst simultaneously using these address keys in our zk-SNARK proof. The verifiable encryption of SAVER allows us to proof valid decryption of a ciphertext without revealing the decryption key. This is especially useful in our setting, since the decryption key is also used on other messages that must remain secret and must thus not be leaked.

[1]Efficient to use in a SNARK setting.

```
$\mathrm{Setup}(1^\lambda, C)$:
    $CRS \leftarrow \mathrm{Setup}^{\mathsf{zkp}}(1^\lambda, C) \cup \{G^{-\gamma}\}$
    return $CRS$

$\mathrm{KeyGen}(CRS)$:
    $\{s_i\}_{i=1}^n, \{z_i\}_{i=1}^n, \{t_i\}_{i=0}^n, \rho \leftarrow_R \mathbb{Z}_p^*$
    $PK \leftarrow \left(g^\delta, \{g^{\delta s_i}\}_{i=1}^n, \{g_i^{t_i}\}_{i=1}^n, \{h^{t_i}\}_{i=0}^n, g^{\delta t_0} \prod_{j=1}^n g^{\delta t_j s_j}, g^{-\gamma(1+\sum_{j=1}^n s_j)}\right)$
    $SK \leftarrow \rho$
    $VK \leftarrow (h^\rho, \{h^{s_i z_i}\}_{i=1}^n, \{h^{\rho z_i}\}_{i=1}^n)$
    return $SK, PK, VK$

$\mathrm{Enc}(CRS, PK, \{m_i\}_{i=1}^n, \{\phi_i\}_{i=n+1}^\ell, a)$:
    Parse $PK \rightarrow (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
    $r \leftarrow_R \mathbb{Z}_p^*$
    $\mathrm{ct} \leftarrow \left(X_0^r, X_1^r g_1^{m_1}, \ldots, X_n^r g_n^{m_n}, P_1^r \prod_{j=1}^n Y_j^{m_j}\right)$
    $\left(g^A, h^B, g^C\right) \leftarrow \mathrm{Prove}^{\mathsf{zkp}}(CRS, \{m_i\}_i \cup \{\phi_i\}_i, a)$
    $\pi \leftarrow \left(g^A, h^B, g^C P_2^r\right)$
    return $\pi, \mathrm{ct}$

$\mathrm{VerifyEnc}(CRS, PK, \pi, ct, \{\phi_i\}_{i=n+1}^\ell)$:
    Parse $PK \rightarrow (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
    Parse $\pi \rightarrow \left(g^A, h^B, g^C\right)$ and $\mathrm{ct} \rightarrow (c_0, \ldots, c_n, \psi)$
    assert $\prod_{j=0}^n e(c_j, Z_j) = e(\psi, h)$
    assert $e(g^A, h^B) = e(g^\alpha, h^\beta) \cdot e(\prod_{i=0}^n c_i \cdot \prod_{i=n+1}^\ell g_i^{\phi_i}, h^\gamma) \cdot e(g^C, h^\delta)$

$\mathrm{Dec}(CRS, VK, SK, ct)$:
    Parse $VK \rightarrow (V_0, \{V_i\}_{i=1}^n, \{W_i\}_{i=1}^n)$, $SK \rightarrow \rho$ and $\mathrm{ct} \rightarrow (c_0, \ldots, c_n, \psi)$
    for $i \leftarrow 1$ to $n$ do
        $e(g_i, W_i)^{m_i} \leftarrow \frac{e(c_i, W_i)}{e(c_0, V_i)^\rho}$
        Brute force compute $m_i \leftarrow \mathrm{dlog}(e(g_i, W_i)^{m_i})$
    end
    $\nu \leftarrow c_0^\rho$
    return $(m_1, \ldots, m_n, \nu)$

$\mathrm{VerifyDec}(CRS, VK, \{m_i\}_{i=1}^n, \nu, ct)$:
    Parse $VK$ as $(V_0, \{V_i\}_{i=1}^n, \{W_i\}_{i=1}^n)$ and $\mathrm{ct}$ as $(c_0, \ldots, c_n, \psi)$
    assert $e(\nu, h) = e(c_0, V_0)$
    for $i \leftarrow 1$ to $n$ do
        assert $e(g_i, W_i)^{m_i} = \frac{e(c_i, W_i)}{e(\nu, V_i)}$
    end
```

Algorithm 1: SAVER construction (relevant parts only).

SAVER builds directly upon the *Groth16*, and related[2], proving systems. Recall that a proving systems has two types of inputs: public inputs $\phi = (\phi_1, \dots, \phi_l)$ and secret auxiliary inputs a (witnesses). These public inputs ϕ combined with a zk-SNARK proof $\pi = (g^A, h^B, g^C)$ allow a verifier to ascertain that the prover indeed has knowledge of these secret inputs a. The verifier achieves this by checking the verification equation, with $g_i = g^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}}$:

$$e(g^A, h^B) \overset{?}{=} e(g^\alpha, h^\beta) \cdot e(\prod_{i=0}^{l} g_i^{\phi_i}, h^\gamma) \cdot e(g^C, h^\delta).$$

SAVER exploits the fact that some values ϕ_i might be considered a plaintext message and that $g_i^{\phi_i}$ is very similar to ElGamal encryption. The verifiable encryption scheme SAVER also uses two algorithms from *Groth16* as a subroutine, namely $\mathrm{Setup}^{\mathsf{zkp}}$ and $\mathrm{Prove}^{\mathsf{zkp}}$. These algorithms are used for key generation and message encryption.

Concretely, SAVER splits a plaintext message M into n k-bit blocks $\{m_i\}_i$ as $M = (m_1 \| \dots \| m_n)$, with k chosen properly. k should be a non-negative number that is chosen in such a way that it is still feasible[3] to compute the discrete log $m_i = \mathrm{dlog}(g^{m_i})$ by brute forcing all options for all k bit blocks. Note that choosing k too small will result in a large amount of a message blocks, and thus a larger public and verification key, more encryption time, and a possibly larger prove time. Algorithm 1 depicts the construction of the relevant parts (for this research) of SAVER.

2.4 Cryptographic building blocks

In this section we present the relevant concepts and definitions for the cryptographic building blocks that comprise a significant part our final protocol. Specifically, we discuss commitment schemes, collision-resistant hash functions, signature schemes, secure pseudo-random functions, and encryption schemes.

Commitment schemes. A commitment scheme or commitment function $C(\cdot, \cdot)$ is a function that takes as input a certain plaintext x and some randomness r. The function returns a commitment value c as $C(x, r)$. A commitment scheme is called secure when it is (computationally) binding and (computationally) hiding. The informal definitions for binding and hiding commitment schemes are given below, for more details on commitment schemes and the formal definitions we refer the reader to Appendix A.4 and B.

[2]See the original publication [22] for more information on this and other details regarding SAVER.
[3]It should take a practical amount of time, depending on how fast the decryption is required to be.

Binding. Given the commitment function $C(\cdot, \cdot)$ it should be hard to find two openings to the same commitment value. Specifically, it should be hard to find two pairs (x, r) and (x', r') with $x \neq x'$ such that $C(x, r) = C(x', r')$.

Hiding. Given a commitment value $c = C(x, r)$ it should be hard to determine the value x used as input for $C(\cdot, \cdot)$. Specifically, when an adversary selects two values x_0, x_1 and a challenger computes $c = C(x_i, r)$ for some random value r and $i \in \{0, 1\}$, it should be hard for the adversary to determine the value of i with a probability higher than $\frac{1}{2}$.

Collision-resistant hash functions. A hash function is a function $H(\cdot)$ that takes inputs x of arbitrary bit lengths and produces fixed-length bit strings as output $y := H(x)$. In this research we consider collision-resistant hash functions (CRHs). For sake of disambiguation we note that collision-resistant hash functions are not necessarily cryptographic hash functions as collision-resistance does not imply preimage-resistance. On the other hand, collision-resistance does imply second-preimage-resistance. In this research we only rely on collision-resistance, though we present all above mentioned (informal) definitions for completeness.

Preimage-resistance. Given a hash value y it should be hard to find an input, or preimage, x such that $y = H(x)$.

Second-preimage-resistance. Given an input, or preimage, x it should be hard to find another input x' such that $H(x) = H(x')$.

Collision-resistance. It should be hard to find two different messages m, m', with $m \neq, m'$, such that $H(x) = H(x')$.

Signature schemes. A digital signature scheme is a form of asymmetric cryptography. It allows the sender of message to guarantee three properties on the sent message: message integrity, message origin, and non-repudiation. Message integrity ensures that a message has not been altered by anyone other than the sender, message origin ensures that the message has been sent by the owner of public key. Non-repudiation ensures that the owner of the public key cannot claim that the message was sent by someone else. There exist several security notions for digital signature schemes. In this thesis we will only consider the security notion called strong existential unforgeability against chosen message attack ($\mathrm{SUF\text{-}CMA}$), which is strong variant of the more prominent existential unforgeability against chosen message attack ($\mathrm{EUF\text{-}CMA}$). The informal definition of $\mathrm{SUF\text{-}CMA}$ is as given below, for a more detailed definition we refer the reader to Appendix B.

$\mathit{SUF\text{-}CMA}$. Given only the public signature keys and a list of message-signature pairs (m, σ) signed under the according secret signature key it should be hard to

construct a new different message-signature pair (m, σ). Specifically, an adversary gets a public signature key pk belonging to a secret signature key sk. The adversary is allowed to ask for signatures σ on arbitrary messages m under the secret signature key sk. If the signature scheme is $\mathrm{SUF\text{-}CMA}$ secure it should be hard to construct a pair (m', σ') that can be verified using the public signature key pk and for which $(m', \sigma') \neq (m, \sigma)$ for any previously constructed pair (m, σ).

In our protocol we consider a slight variation on $\mathrm{SUF\text{-}CMA}$ called strong existential unforgeability against one-time chosen message attack ($\mathrm{SUF\text{-}1CMA}$). As the name already hints, the only difference with $\mathrm{SUF\text{-}CMA}$ is that the adversary is only allowed to ask for the signature on *one* arbitrary message instead of multiple.

Pseudo-random functions. A family of functions $\{F_k\}_K$ can be a so called pseudo random function (PRF) family if certain requirements are satisfied. We assume that all functions $F_k(\cdot)$ in the family have the same domain and codomain, and that k is chosen from the key space K. The input x to a PRF is often called a seed. Such a family of functions is considered to be a secure pseudo-random function family if it satisfies the definition below, for a more detailed definition we refer the reader to Appendix B.

Secure PRF family. Given access to an oracle that computes either the values of $y = F_k(x)$ for some secret value k or returns completely random values on input x, it should be hard for the adversary to determine whether the oracle computes $F_k(x)$ or returns a completely random value. It should be noted that the oracle always returns the same value y for the same input x, to achieve this it internally stores all previously computed input-output pairs (x, y).

Encryption schemes. An encryption scheme is generally comprised of two algorithms: an encryption function and a decryption function. The encryption function takes a regular message, called plaintext, as input and outputs a seemingly random text called the ciphertext. A ciphertext that is generated by a secure encryption scheme reveals no information about the original plaintext. The decryption algorithm is used to transform the ciphertext back into the plaintext. Both methods require (secret) values, called keys, to encrypt and decrypt messages.

There exist two types of encryption: symmetric key encryption and asymmetric or public key encryption. In symmetric key encryption the decryption key is equal to the encryption key and must be kept secret at all times. In asymmetric encryption the encryption key and decryption key are different. We call the the encryption key the public key and the decryption key the secret key, referring to their (non)-availability to other users. In an asymmetric encryption scheme, the sender of a message uses the public key of the receiver to encrypt the message. The receiver of the message is the only person that knows the secret key belonging to the public key. Therefore, the receiver is the only person who can decrypt the ciphertext to the original message. There exist several security notions for encryption scheme, we will consider two: key indistinguishability under chosen ciphertext attack ($\mathrm{IK\text{-}CCA}$) and indistinguishability under chosen ciphertext attack ($\mathrm{IND\text{-}CCA}$) [23]. In our protocol we will be using asymmetric encryption schemes and we will therefore also give the security definitions in that setting.[4] Below, we give the (informal) variant of these definitions. Fr a more detailed definition we refer the reader to Appendix B.

$IK\text{-}CCA$ Given two public keys of the same encryption scheme, and the ability to request decryptions of arbitrary messages under either key, it should be difficult to distinguish under which of the two public-keys the encryption of a self-chosen message is encrypted. Specifically, an adversary is given two public keys belonging to the same encryption scheme. The adversary then sends one message of choice to a challenger. The challenger returns the encryption c of this message under one of the two public keys and challenges the adversary to say under which key the ciphertext c is encrypted. Before making this guess the adversary is allowed to request the decryption of an arbitrary number of arbitrary ciphertexts (not equal to c) under either key.[5] The encryption scheme is called secure if it is hard for the challenger to make the correct guess with a probability more than $\frac{1}{2}$.

$IND\text{-}CCA$ Given the public key of an encryption scheme, and the ability to request decryptions of arbitrary messages, it should be difficult to distinguish the encryptions of two self-chosen messages. Specifically, an adversary is given the public key of an encryption scheme. The adversary then sends two messages of choice to a challenger. The challenger returns the encryption c of one of these messages and challenges the adversary to say which of the two messages is encrypted in c. Before making this guess the adversary is allowed to request the decryption of an arbitrary number of arbitrary ciphertexts (not equal to c).[5] The encryption scheme is called secure if it is hard for the challenger to make the correct guess with a probability more than $\frac{1}{2}$.

[4]Extension to the symmetric case of $\mathrm{IND\text{-}CCA}$ is rather straightforward. Logically, there exists no symmetric variant of $\mathrm{IK\text{-}CCA}$.

[5]The adversary is also allowed to encrypt arbitrary messages under either key. We do not define this explicitly, since the adversary knows the public key(s) and can thus easily encrypt messages by itself.

2.5 Merkle trees

A Merkle tree, or hash tree, is a data structure capable of storing large sets of data that allows for efficient membership proofs. A Merkle tree consists of two types of nodes: internal nodes and leafs. Each leaf represents one data record, whereas the internal cells are used to be able to proof membership of a data record efficiently. We will only consider binary Merkle trees here, i.e. trees where each non-leaf node has exactly two children. These binary trees allow for membership proofs that scale logarithmically, in both size and time, in the number of leafs of the tree.

A leaf of a Merkle tree is either a hash of a piece of data, or just the piece of data if each and every cell has the same suitable length. Each internal node is the hash of the concatenation of both its children. A Merkle tree has one internal node without a parent, this node is called the root of the Merkle tree or Merkle root. This root is used to denote the current state of the Merkle tree, moreover this root plays a key role in a membership proof.

An example of a small Merkle tree is given in Figure 2.1. This figure also depicts and describes an example of a membership proof for a certain leaf in a Merkle tree. Figure 2.1 proofs that the data `data3` in `LEAF 3` is contained in the Merkle tree with Merkle root `ROOT`. This can be done by showing that there are other nodes, the so called path nodes, that together with `data3` compute the value of the Merkle root `RT`.

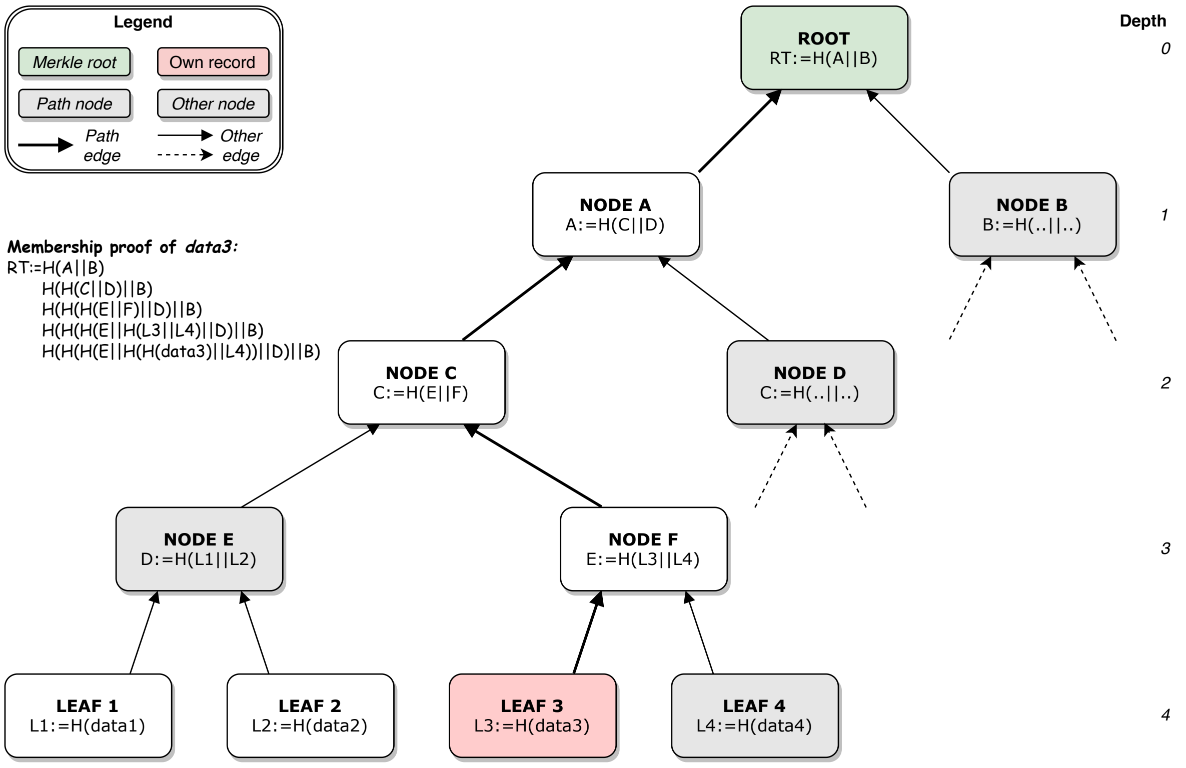


Figure 2.1: Example of a Merkle tree of depth 4.

In general, a membership proof consists of three parts, the Merkle root, a path of internal nodes, and the leaf/piece of data over which membership is proved. The path of internal nodes contains exactly $d-1$ internal nodes, with d the depth of the Merkle tree, and one other leaf. To be precise, this other leaf is the sibling of the leaf over which membership is proved, the first internal node is the sibling of the parent of both leaves, the second internal node is the sibling of the parent of this parent, and so on, all the way till the Merkle root is reached. The membership proof checks that taking the hash recursively over all these nodes gives the value of the Merkle root.

Using Merkle trees in a blockchain setting requires us to store the tree efficiently. It would be rather inconvenient if each block on the chain contains the full contents of the Merkle tree at that point. Fortunately, the Merkle root includes the entire state of the Merkle tree at that point in time, and due to the collision-resistance of the used internal hash function it is infeasible to find a tree with different contents and the same root. Therefore, in blockchain applications it is sufficient to store only the Merkle root on the blockchain and keep other information about the tree 'off-chain'.

Chapter 3

Related work

In this section we discuss the techniques used in several prominent solutions for anonymous payments, in more detail. We start with some older centralised anonymous e-cash solutions, including Chaum's Digicash [6]. We then discuss a decentralised payment scheme called GNU Taler [10] that is based on the principles of Digicash. What follows is a discussion of two types of privacy coins, obfuscation based and cryptography based cryptocurrencies. The first category considers among others Monero [11], Verge [12], and Grin [13]. Finally, the second category focuses on the basis for our protocol: Zerocash [16] and its predecessor Zerocoin [15].

3.1 Centralised anonymous e-cash

The first prominent solution for anonymous digital payments, also known as e-cash, was Chaum's Digicash. Digicash is based on the scheme presented in Chaum's work on blind signatures for untraceable payments from 1983 [7]. Chaum's system identifies three parties in a transaction: the bank, the payer, and the payee. The system relies on a blind signature system that allows to bank to sign certain values without learning them. This is best explained by a envelope lined with carbon paper. When the payer want the bank to sign a new note, the payer puts this note in the carbon paper lined envelope, closes it and sends it to the bank. On receiving this closed envelope, the bank signs it and returns the signed envelope to the payer. Because the envelope was lined with carbon paper the signature also left a carbon copy on the note inside. The payer has now received a signed note without the bank learning what exactly was signed.

The payer can subsequently give this note to the payee, who can easily verify that the note was indeed signed by the bank and thus that it is valid. The payee can later cash this note at the bank and receive the appropriate value in fiat currency on his or her bank account. In this entire transaction, neither the bank nor the payee learn anything about the identity of the payer, since the signed note is unlinkable to the signed envelope. The digital version of this system works similarly, with the addition that each new note should have a unique number written on it. When the bank receives this note, the number on it is put on a long list of all spent notes. This method ensures that double spending of a note is not possible, even though digital duplications are a lot easier than physical ones.

A big advantage of this system is that it is rather simple to implement and the required computations can be done very efficiently. Disadvantages are that the notes in the system are of fixed value, which is not just inconvenient but also leaks the amount payed in a transaction. Moreover, comprising the secret keys for the blind signature system allows one to forge an arbitrary amount of notes. This single point of failure is rather undesirable. Moreover, a centralised digital system might also inspire less trust regarding the anonymity of it's users and could even be prone to side-channel attacks.

A centralised anonymous payment scheme that does not suffer from the drawback of the single point of failure caused by the secret keys of a blind signature system is suggested by Sander and Ta-Shma in 1999 [8]. Their system actually does not rely on the bank to keep a secret at all. Instead the system relies on the ability of the bank to reliably maintain a public database. This system again identifies three parties in a transaction: the bank, the payer, and the payee.

The payer start a transaction by withdrawing money from her bank account. Since this system, like the previous one, only considers fixed value anonymous e-cash, she withdraws the amount of one 'note'. The payer also chooses two random secret values x and r and sends a (hiding and binding) commitment $\mathrm{cm} = C(x, r)$ to the bank. In return for withdrawing a 'note' from the payer's account, the bank adds cm to the public list of coins. This public list of coins is encoded as a Merkle tree to significantly increase the performance. The payer can spend the newly minted note cm anonymously by using zero-knowledge proofs. To be precise, the payer needs to construct a zero-knowledge proof π that she knows values x and r such that $C(x, r)$ is in the public Merkle tree. The payee of a transaction can use this proof π to cash the received note. The payee need only show the proof π to the bank and in return receive the value of a 'note' in his or her bank account. To prevent double-spending each note is linked to a serial number. However discussing the details of this is outside the scope of this literature review.

This system is still rather simple to implement, and given the current progress in the efficiency of zero-knowledge proofs can be made rather efficient. An additional advantage of this system is that it does not rely on a single secret held by the bank to prevent forgery. However, the other drawbacks as mentioned for Digicash still hold and seem to be characteristic for centralised solutions. An additional disadvantage of centralised systems with multiple banks is that these banks have to cooperate strongly. There are several hurdles that might make this cooperation infeasible, such as regulations for financial institutions or unwillingness to cooperate. Therefore, we decide to disregard centralised solutions for now and take a look at (more) decentralised solutions.

3.2 Decentralised anonymous payment schemes

Probably the most well-known type of decentralised anonymous payment schemes are privacy coins, cryptocurrencies with a strong focus on user anonymity. However, there are also other options out there. In this section we will discuss one of these options called GNU Taler [9], [10], which is best portrayed as a decentralised payment scheme with some central control. GNU Taler is especially interesting because of its focus on both anonymity and auditability. GNU Taler identifies five types of actors: exchanges, customers, merchants, banks, and auditors. Exchanges could also be banks, but might also be independent third parties. On a very high level GNU Taler constructions work as follows. (1) The customer withdraws some coins from an exchange; (2) The customer spends some coins at a merchant; (3) A merchant deposits the coins at an exchange; (4) The auditor monitors the behaviour of an exchange to verify that the exchange operates honestly. The bank is requested to provide intermediate wire transfers for the customer and the exchange. The customer and exchange can both use their own, possibly different, bank.

For a customer to be able to pay a merchant, a customer must hold coins at an exchange that is trusted by the merchant. The exchange is used to hold funds (fiat currency) of its customers in escrow and to transfer these funds to a merchant when digital coins are deposited. The system assumes that there exists an anonymous, bi-directional communication channel, such as Tor, between a customer and the merchant, and between either party and the exchange. The anonymity of the system is thus not present in withdrawing or depositing coins. The anonymity of the system is on the payment level, the deposited coins cannot be linked to the withdrawn coins, hence the identity of the customer can not be linked to both transfers. On the other hand, the merchant's identity will always be learned in GNU Taler, since that is necessary to send funds to the merchant's bank account. In other words, GNU Taler only provides customer/payer/sender anonymity and does not provide merchant/payee/receiver anonymity.

This last fact is also the biggest drawback of GNU Taler in our use case, we want full anonymity for both payer and payee. Moreover, the exchanges also form another party that learn information about its clients, which to us seems undesirable.

The functionality of keeping ‘digital coins’ or ‘notes’ in escrow that is provided by the exchanges could also be placed directly with the customers and merchants. They could even re-spend the received notes in future transactions. Removing the exchanges from the equation does unfortunately also remove the auditability part of GNU Taler. We thus need to find a way to incorporate auditability or other financial measures in our system without relying on intermediaries, i.e. in a fully decentralised setting. Privacy coins could be a good resource for finding techniques that allow us achieve this in a fully decentralised setting.

3.3 Obfuscation based privacy coins

In the context of cryptocurrencies, two main types of privacy coins can be identified: obfuscation based and cryptography[1] based. This identification is quite fluent, some privacy coins might be clearly obfuscation based while others are a bit of both. In this section we will focus on the more prominent type: obfuscation based. To be precise, we will discuss several types of privacy coins that together use a wide range of techniques and give one overall conclusion on this type of currencies.

The first currency we discuss is Monero [11]. The privacy of Monero is based on a cryptographic technique known as the traceable ring signature [14]. Together with one-time stealth addresses unlinkability of transactions is provided. The ring signature is used to hide the sender of a transaction, the stealth address hides the receiver. On a high level, this works as follows. The sender wants to transfer a certain amount of value to the receiver and hence constructs a transaction containing the details of this transaction. This transaction is to be posted on the public blockchain, and should thus be signed in order to ensure integrity. To sign the transaction, the sender uses a ring signature. Where a normal signature would sign the transaction using the sender’s public key, the ring signature sings it on behalf of a group of different spend keys. One key in the signing group, or ring, has to be the sender’s public key. Effectively, the sender of a transaction is hidden amongst all the other (randomly selected) key owners. To hide the receiver’s incoming public key we use the stealth address. Normally, every transaction to the same user is sent to the same public address key. However, in Monero the sender of a transaction is forced to create a unique, per transaction, stealth address based on the receiver’s public address key. This stealth address is unlinkable to the actual public address key by anyone other than the receiver, thus providing anonymity for the receiver combined with the convenience of one incoming public key. Currently, Monero uses an updated version

[1]This does not imply that obfuscation based privacy coins do not use cryptography. However, obfuscation based currencies do not use cryptography to hide the full transaction graph.

of their original ring signature scheme, called ring confidential transactions (RingCT): a combination of ring signatures and confidential transactions. Next to hiding the receiver's address, RingCT also hides the transferred value. Hiding the transferred value makes side-channel attacks based on the transferred values impossible.

Monero is one of many cryptocurrencies that are based on the CryptoNote [24] technology and only one of the few that still exists. The history of many CryptoNote based currencies also reveals one of the biggest issues of unregulated cryptocurrencies and why many ceased to exist. Bytecoin, the predecessor of Monero and one of the first CryptoNote based currencies, has a very ambiguous history. After the popularity of Bytecoin greatly increased in a short time, it turned out that the creators of Bytecoin did not have the best intentions, and the value of Bytecoin dropped significantly in a short time span, causing many users to lose money [25], [26]. This again shows, that even though the ideas for some cryptocurrencies are very good, some form of regulation is desired in this domain.

Verge [12] is the second obfuscation based currency we discuss. Verge offers three types of transactions, in increasing order of anonymity those are: simple, stealth, and anon. In this discussion we focus on the last one, since it provides the highest level of anonymity. The anon transaction of Verge are in many ways similar to those in Monero. Verge also uses RingCT to preserve sender and value anonymity, and uses dual-key stealth addresses to provide privacy to the sender. There is one slight difference, Verge also heavily relies on Tor [27] for its anonymity. Tor is used to hide the IP-addresses of users interacting with the blockchain.

Of the three currencies we discuss here, Grin [13] is probably the most cryptography based. Grin uses a protocol called Mimblewimble [28]. Mimblewimble does not rely on stealth addresses or something similar to provide privacy, actually Mimblewimble does not really require addresses. Mimblewimble uses confidential transactions (second part of RingCT) to hide the transaction amounts that go into and out of a transaction. To be more specific, all inputs and outputs are combined into one Pedersen commitment, i.e. they are bulked together. To hide all the values this transaction also requires a secret blinding factor. Knowing this blinding factor allows one to spend the outputs of a transaction. Grin uses this blinding factor instead of using public address keys. The sender and receiver are the only ones involved in a transaction and are therefore also the only ones that know the blinding factor. Thus the receiver can simply spend the received funds, because he or she knows the blinding factor. Because of the lack of addresses the transactions in this system are completely anonymous.

Beam [29] is another currency based on the Mimblewimble protocol. In many ways it is similar to Grin, apart from the business model. In contrast to most privacy coins that are community-backed currencies, Beam is venture-backed. Because of this, Beam receives expert advise and investments from (financial) enterprises and also has more focus on regulation and auditability. This business direction is actually more interesting from the viewpoint of our use case, and shows that obfuscation based currencies can also include auditability features in their design.

The three above cryptocurrencies rely on different techniques to provide privacy and do so rather well. A disadvantage of the first two types of transactions is that the anonymity set that a sender belongs to is rather small. It would be better if the sender was hidden in a much larger set, possibly the entire user base. Grin on the other hand provides a larger anonymity set, however the used cryptographic techniques would make it hard to incorporate in a permissioned and auditable setting. To conclude, the techniques used in obfuscation based protocols are promising, but the delivered anonymity could be improved upon. The more cryptographic solution seem more promising on this terrain and it seems like the right direction to move forward to, provided that it can still fit our use case of permissions and auditability.

3.4 Zerocoin and Zerocash

The most well known examples of cryptography based privacy coins are Zerocoin and Zerocash. Both decentralised payment scheme use zero-knowledge proofs to hide the transaction graph completely. In other words, the anonymity set of the sender and receiver of a transaction is literally the entire user base of the currency. We will first discuss Zerocoin [15] and then it's 'successor' Zerocash [16].

In contrast to the earlier discussed anonymous transaction schemes, Zerocoin does not rely on signatures or a central authority to ensure transaction integrity. Zerocoin uses zero-knowledge proofs to proof that a 'coin' or 'note' is part of the public valid coin list and has not been spent before. Similarly, to the work of Sander and Ta-Shma [8], Zerocoin stores this valid coin list as a Merkle tree to improve computation times. On a high level, the protocol works as follows.

The Zerocoin protocol was originally implemented in the cryptocurrency known as Zcoin [30]. After this currency was used for a while a group of security researcher found a security fault in the original protocol, even though the protocol was proven to be secure and had received extensive peer review [31]. One might expect this to be the end of Zerocoin, however the community and core team decided to fix the problems by implementing a new protocol, something that is not seen too often in cryptocurrencies. The devised their own protocol called sigma to revive Zcoin as a secure and anonymous digital payment scheme. Later on, Zcoin replaced this protocol with the Lelantus protocol. The advantage of the new Zcoin is that a trusted setup is no longer required and that the efficiency with respect to the original publication is greatly increased.

The ideas of the original Zerocoin protocol also lead to the publication of the Zerocash protocol. A transaction in Zerocash has two actors: the sender and the receiver. Moreover, Zerocash requires a backing currency, such as Euros or bitcoin. There are two types of transactions in Zerocoin: Mint transactions and Spend transactions. A mint transaction is used by the sender to swap the backing currency to an anonymous coin. A coin consists of a commitment cm to a unique secret serial number sn (and randomness r). This mint transaction is announced publicly on the blockchain and includes the commitment and the withdrawal of backing currency. This commitment cm is then included into the blockchain.

To spend the newly minted coin, the sender creates a spend transaction. This spend transaction contains a zero-knowledge proof π that proofs that the sender knows a number sn and randomness r that together form a commitment that belongs to the Merkle tree. The sender than publishes a transaction containing this proof π, serial number sn to prevent double spending, and the public address key of the receiver. Because π is zero-knowledge this spend transaction cannot be linked to the mint transaction. After verification of this transaction the receiving address get the same amount of value in the backing currency deposited in his or her account. The commitments that Zerocoin uses are Pedersen commitments and the zero-knowledge proofs are double-discrete logarithm proofs. This results in rather large proof sizes of over 45 kB and a verification time of over 450 ms. These values are not very practical in blockchain settings, since every transaction needs to be verified and is also stored for ever. On the other hand, the proof creation time of less than a second is very suitable for use in real-life.

Zerocash and its current implementation Zcash [32] improve Zerocoin on several points. Firstly, zk-SNARKs are used to generate the zero-knowledge proofs. This leads to big decrease in proof size to approximately 300 bytes. The verification time is also decreased to just a couple of milliseconds. Unfortunately, the time time to create a proof has increased to around a minute in the first version of Zerocash. Fortunately, recent improvements in the Zcash implementation have brought the proof time back to well under 10 seconds. These practical performance measures make it very suitable for use in practice.

Next to a performance increase, Zerocash also significantly increases on convenience and anonimity. Anonymous notes or coins need not any longer be directly interchanged with the backing currency as the output notes of a transaction can now also be used as inputs in a new transaction. Moreover, the notes have been made fungible, i.e. the sender of a transaction can get change back and thus pay arbitrary amounts to another party. This also greatly increases anonimity, as the transaction amounts are not visible any longer. A final addition to the anonimity of users is the fact that receiver addresses of spend transactions are no longer visible any longer.

Addresses are only revealed upon conversion of the backing currency to anonymous coins or vice versa. For a detailed explanation of the Zerocash protocol we refer the reader to Section 4.3. To emphasise the popularity of the current *Sapling* version of the Zerocash protocol, we note that other currencies such as PIVX [33] and Tezos [34] have also implemented this protocol.

We conclude this section with the observation that the cryptography based privacy coins seem to be very suitable for our use case as they provide a significantly higher level of privacy than the other options. This especially holds for Zerocash and the revived version of Zcoin, since both are very practical regarding computation times and transaction sizes. We favor, Zerocash over Zcoin as the used primitives in Zcoin do currently not allow for the extensive statements that we most likely require to make our scheme anonymous ánd auditable. The zk-SNARKS used by Zerocash do provide us with the flexibility of proving nearly any thinkable statement These zk-SNARKs can also be used to provide auditability functionality and realise a permissioned blockchain, without diminishing the privacy of the user.

Chapter 4

Solution sketch

In the previous chapters we discuss existing approaches from literature and describe useful building blocks and definitions. We also set out a direction for our solution approach. In the first section of this chapter we refine our direction by defining a set of requirements that our auditable decentralised anonymous payment scheme should adhere to. In the subsequent sections of this chapter we give a step-by-step sketch of our final protocol. These steps are presented as incremental versions to give a complete insight in the design process and the made choices. A general overview of our payment scheme is given in Section 4.2.

In Section 4.3 we present the basis of our protocol as derived from the original Zerocash publication [16]. We then move towards an account-based model in Section 4.4 and include access control in Section 4.5. Conversion of anonymous notes to and from fiat currency is discussed in Section 4.6. A transaction limit is introduced in Section 4.7 and is extended with verifiable encryption for auditability in Section 4.8. Our final section 4.9 explains the incorporation of timelocks into the protocol.

4.1 Requirements

Below, we present a set of requirements that the solution to our problem should adhere to. In total we define a list of nine requirements with a short (informal) description, that together represent the goal and research questions as mentioned in Section 1.2. Formal definitions can be found in Appendix C. Since our solution will be an adaptation of the Zerocash protocol, we use the requirements of that article as a basis for our nine requirements. In line with the definition of the decentralised payment scheme from Zerocash [16] we define the following four basis requirements.

Completeness. A digital payment scheme is complete when any received value that has not yet been spent can actually be spent or stored in the receiver's account balance. Moreover, it requires that any value from the account balance that was not yet spent can be used, i.e. transferred to another user.

Ledger indistinguishability. This requirement focuses on the fact that no new information should be leaked to an adversary from everything that is published on the distributed ledger. The term ‘no new information’ refers to the fact that no other information should be leaked from the publicly available values than is required for the payment scheme to work.

Transaction non-malleability. Transaction non-malleability requires valid transactions to be constructed in such a way that no adversary can adapt a transaction such that it is either no longer valid, or performs a different transfer than the original sender intended. To be a bit more precise, no adversary should be able to alter any of the data included in a valid transaction.

Balance. A digital payment system is said to be in balance, if no user can own or spend[1] more value than what he or she received or converted from fiat currency into the system.

In addition to the requirements from Zerocash we also add five new requirements to include our own goals and research questions.

Access control. In a digital transaction scheme with access control, the administrators of the system should be able to decide who does and who does not get access to the transaction system. In this case, access describes the possibility to perform transactions in the system and write something on the distributed ledger or blockchain. Moreover, it should also be possible to revoke access of certain users in case this is needed. Finally, the access control also influences the way of reaching consensus on the current state of the distributed ledger. Namely, the administrators should have full control over this state, the regular users may use the blockchain, and validate the correctness thereof (possibly as means to check the administrators) but does not decide on whether or not a transaction gets added to the ledger.

Conversion to/from fiat currency. This requirement describes the linkage between the transfer of value in the digital transaction scheme and transfer of value using fiat currency. In order to link these two types of value, a conversion between both should be possible in the payment scheme. This conversion might be an actual conversion of one into the other, alternatively the scheme might also contain certain notes that represent fiat currency without an actual conversion taking place.[2]

Spend limit. The spend limit requires that the payment system provides a possibility to limit the amount of value any user can spend in a certain time frame. To make this more precise, it means that the the sum of the value of all outgoing transactions in

[1]Conversion to fiat currency is also considered to be a form of spending.

[2]We will adopt the first option.

time frame T, for any user u, may not be higher then a certain predetermined limit L.

Auditability. This requirement is defined in coherence with the *spend limit*. In our use case, we consider auditability to be only required for transactions that surpass the spend limit. These transactions should include the important transaction details in a special field: transferred value, sender address, receiver address. This field should be encrypted as to not leak information to other users. Decryption of the field should only be possible by a select group of actors referred to as judges.

Timelocks. It should be possible for a sender to define a transaction with a timelock. This timelock should make sure that the receiver can only spend the output value of the transaction after the set amount of time has passed. In order to prevent side-channel attacks, it should not be visible to actors other than the sender and receiver that a timelock is in place.

4.2 Overview

Before we dive into the workings of our new protocol, we sketch a general overview of our payment scheme. Our payment scheme consists of two type of actors: users and administrators. There is one central data structure, the blockchain or distributed ledger.

Banks, regulators, and other (financial) institutions together form the *administrator*s in our scheme. The clients of participating banks are all potential *user*s of the system. To become an actual user of the system, any client needs to request an account from their bank. In response, the bank performs KYC on any of their clients requesting permission to use the payment scheme. When the KYC is completed satisfactory, a new client gets an ‘account’ in the payment scheme.

With an account on the blockchain, any user can create anonymous transactions that transfer money from their account to another user’s account. When a user creates a new transaction he needs to add it to the blockchain. The blockchain stores all valid transactions that were performed using the system. The validity of a transactions is checked by the group of administrators. When they ascertain that a transaction is valid it gets included on the blockchain, otherwise the transaction is discarded.

At the same time, all users are checking the new transactions on the blockchain to find any value transferred into their account. When such a transaction is found, the receiver stores the details. The received details can later be used by the receiver to create a new transactions, where he or she is the sender. For future reference, we have schematically depicted this overview in Figure 4.1.

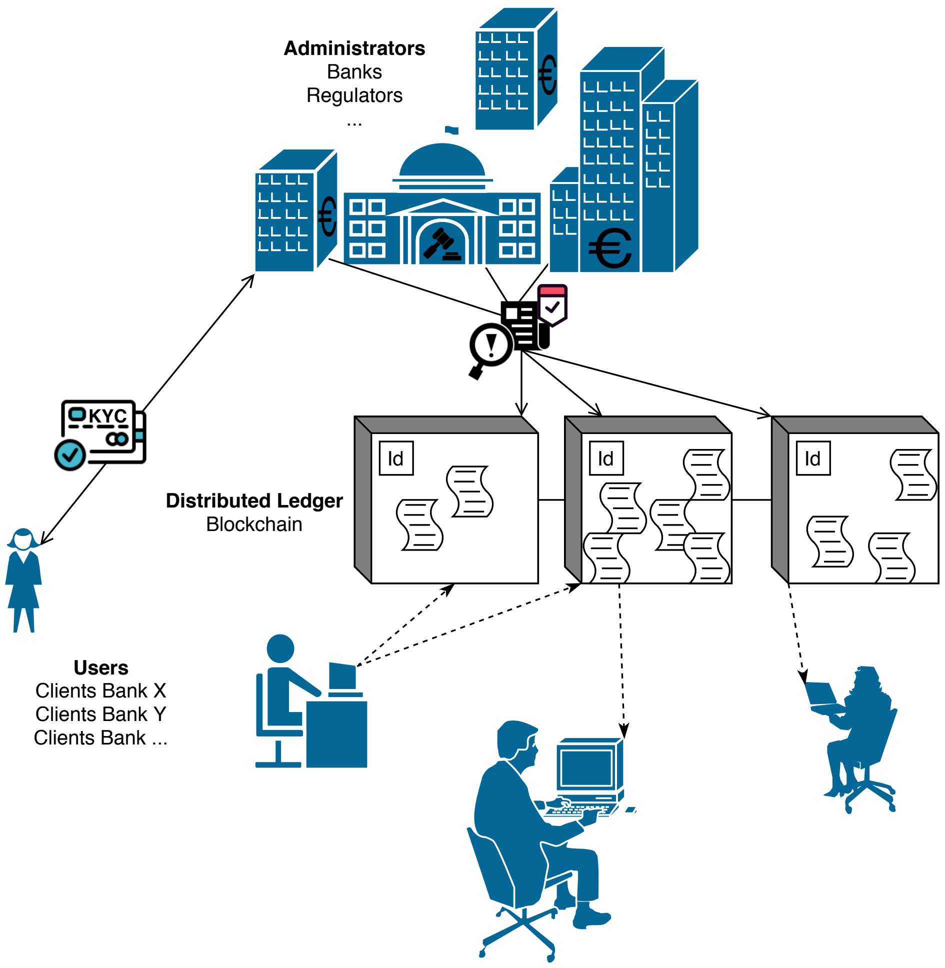


Figure 4.1: *Overview* A new client (left-most) requests access to the system. One of the administrators grants this access, only after having performed KYC. The other users (clients of banks participating in the system) send and receive the notes using several transactions. Their transactions are included on the blockchain, amongst the transactions of all other users. All transactions are checked for their validity by the group of administrators. These administrators can be banks, finanicial regulators or other relevant institutions.

4.3 Zerocash basis

In this section we sketch the parts of the Zerocash protocol that we will use as a basis for our further additions and improvements. Below, we present four incremental versions that step-by-step incorporate the four basis requirements as mentioned in Section 4.1. These requirements will also be taking into account in the versions that will be presented in later Sections. The versions that we present below are a slightly adapted and somewhat trimmed version of the original Zerocash protocol [16] and its implementation in ZCash [32]. The general concept remains the same. In Section 4.4 this adaptation is combined with some ideas in a publication from Dusk [35] about privacy in account-based cryptocurrencies, to arrive at a new basis protocol. We introduce the elements that are used in this basis protocol step-by-step, until we arrive at the complete version. For most of these steps or versions, a schematic depiction of the most relevant new parts and integration thereof is given. We do assume familiarity with the ideas behind distributed ledger technologies and cryptocurrencies, as briefly described in Appendix A.1.

Differences with Zerocash. The four versions that we present below are similar to the Zerocash protocol [16], although we incorporated a fix against the Faerie Gold Attack [32] and made some other minor alterations to make the protocol work with one input note instead of two. The fifth version, in Section 4.4, implements the (basic) idea we took from the Dusk publication on anonymity in account-based cryptocurrencies [35]. We adopt this idea, make it concrete, work out the details and combine it with the ideas from Zerocash to obtain a 'hybrid' payment scheme that is based both in the UTXO model as well as in the account-based model. The versions in subsequent sections are not based on any existing work that the author knows of. These new ideas, among others, add access control to the payment scheme and alter the design in such a way that the scheme can be used as an additional layer on top of existing (digital) payment schemes for fiat currencies.

Version 1: Variable-value notes with owner anonymity. We start with a highly simplified version of the protocol, where a user can only own a virtual *note* of a variable denomination, say €x for some $(x * 10^2) \in \mathbb{N}$, and spend it without revealing the users identity. To achieve this, we use commitment schemes and zk-SNARKs. A commitment scheme Comm takes as input a message m and some randomness r and outputs $\mathrm{cm} := \mathrm{Comm}_r(m)$ which reveals nothing about the input m and for which it is also hard to construct the same output cm for a different input message m'. zk-SNARKs will be used to proof knowledge of witnesses to certain non-deterministic polynomial time (NP) statements.

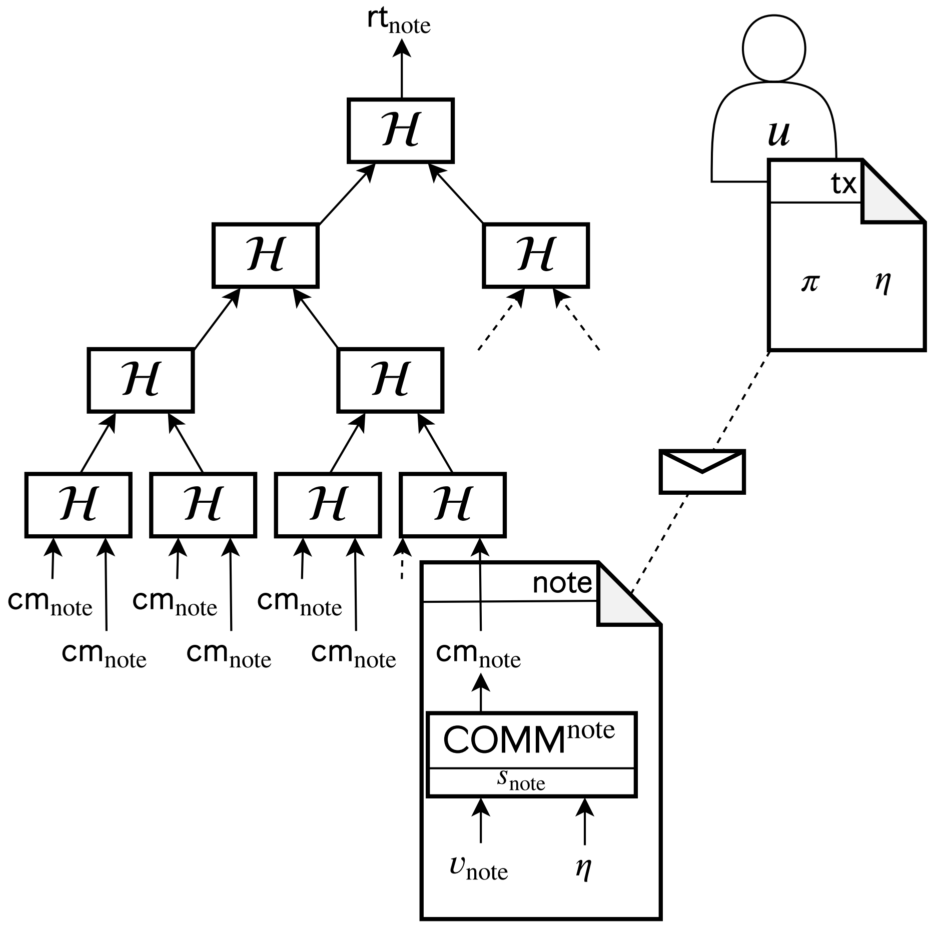

$\mathrm{rt}_{\mathrm{note}}$
$\mathcal{H}$
u
tx
π η
$\mathrm{cm}_{\mathrm{note}}$
note
$\mathrm{COMM}^{\mathrm{note}}$
s_{note}
v_{note} η

Figure 4.2: *Version1* User u publishes a transaction tx containing a proof π and nullifier η. These values are linked to the note note that is included via $\mathrm{cm}_{\text{note}}$ in the Note Merkle tree with root $\mathrm{rt}_{\text{note}}$. The value $\mathrm{cm}_{\text{note}}$ is a commitment to both the value of the note v_{note} and the unique nullifier η.

A user u can create a new note note of value v_{note} with a unique nullifier η as follows: he or she samples some randomness s_{note} and then calculates the note commitment $\mathrm{cm}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s_{\text{note}}}(v_{\text{note}} \| \eta)$. The nullifier η functions as a serial number for each note that prevents double spending, i.e. once published, the note is spent and cannot be spent again. It is important that η is a unique value for each note, since only one note with nullifier η can be spent. The user then needs to get the note added to the set of all the notes that circulate in the system. He or she can do this by proving to one of the admins that the corresponding amount of fiat currency v_{note} was paid to a backing pool of fiat currency. How this works exactly will be discussed later, but for now we will assume that this is possible. The admin thus accepts the commitment and adds it to the current set of notes in circulation in the system, by adding the commitment as a leaf to a Merkle tree, that we will call the Note Merkle tree. This Note Merkle tree is included on the blockchain by means of its Merkle root

$\mathrm{rt}_{\text{note}}$. This Merkle root gets updated every time when a new leaf is added to the tree. Figure 4.2 schematically depicts this construction.

To recapitulate, the user u now has constructed a valid note $\mathrm{note} = (s_{\text{note}}, v_{\text{note}}, \eta, \mathrm{cm}_{\text{note}})$ with denomination v_{note}, to which the commitment $\mathrm{cm}_{\text{note}}$ is acknowledged by inclusion in the Note Merkle Tree. u can spend this note note by, anonymously[3], publishing a transaction tx on the blockchain. This tx should contain the unique *nullifier* η to the note note and a zk-SNARK proof π that proves the following statement: *"I know s_{note}, v_{note} and cm_{note} such that $cm_{note} = Comm^{note}_{s_{note}}(v_{note}\|\eta)$ and cm_{note} is included in the Note Merkle Tree."* The transaction tx is now valid if and only if both the proof π can be verified, and the nullifier η has not been published in an earlier transaction. With tx verified, the publisher of this transaction can now again get the corresponding amount of fiat currency v_{note} from the backing pool of money. Again, how this works in practice will be discussed in later steps.

Version 2: Transfer notes to other users anonymously. Up to now, we have only discussed how to create a new anonymous note and how to spend this same note. However, we have not yet seen how one can transfer a note anonymously to another user. Suppose user u_a wants to transfer its note $\mathrm{note}^{\mathrm{old}}$ with value v_{note} to user u_b. u_a could send all the details of $\mathrm{note}^{\mathrm{old}}$ to u_b, however this would give rise to some problems. Firstly, u_a knows all the values that constructed the commitment, so there is nothing to stop u_a from still spending this $\mathrm{note}^{\mathrm{old}}$, before u_b can do so. Secondly, if u_b were to be able to spend $\mathrm{note}^{\mathrm{old}}$, he or she would have to publish the corresponding nullifier η. However, this would break u_b's anonymity, since u_a also knows this nullifier value and can thus easily link the transaction to this same $\mathrm{note}^{\mathrm{old}}$. To solve this problem, we introduce some additional elements to the protocol.

Firstly, each user needs to generate an address key pair, consisting of a public address key $\mathrm{pk}_{\text{addr}}$ and a secret address key $\mathrm{sk}_{\text{addr}}$. The public address key can be used as the target, or recipient, of a transfer. The accompanying secret address key can among others be used to prove ownership of a received note. $\mathrm{sk}_{\text{addr}}$ should be generated randomly by the user u, who can subsequently construct the public address key as $\mathrm{pk}_{\text{addr}} := \mathrm{PRF}^{\text{addr}}_{\mathrm{sk}_{\text{addr}}}(0)$, where $\mathrm{PRF}^{\text{addr}}_{x}$ is an appropriate PRF with seed x.

Secondly, we need to update the construction of a note. Given a user u with address key pair $(\mathrm{sk}_{\text{addr}}, \mathrm{pk}_{\text{addr}})$. u can now construct a note note with value v_{note} that looks slightly different from the one in the previous version. u randomly samples some s_{note} and calculates the note commitment $\mathrm{cm}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s_{\text{note}}}(\mathrm{pk}_{\text{addr}}\|v_{\text{note}})$, which uniquely defines the note $\mathrm{note} := (s_{\text{note}}, v_{\text{note}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}_{\text{note}})$. We include the public address key of u in the commitment to ensure that the note belongs to u. The

[3]We do not discuss this in the thesis, but using Tor [27] would be a way to achieve this.

user can proof ownership of the note by proving knowledge of the secret address key $\mathrm{sk}_{\text{addr}}$ that was used to construct the public address key $\mathrm{pk}_{\text{addr}}$. Just as in the previous version of the protocol, the newly created note can only be used if it is included in the Note Merkle Tree, which depends on whether the corresponding amount of fiat currency has been added to the backing pool.

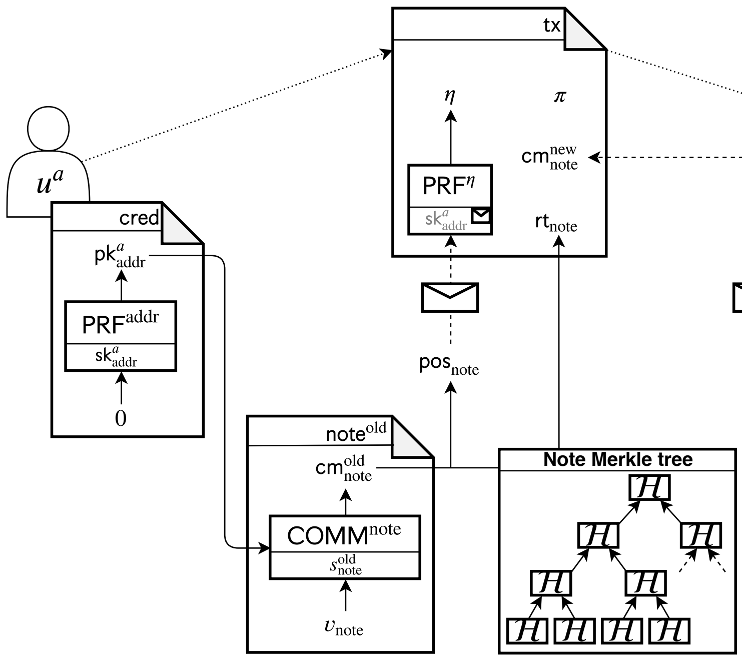

tx
η
π
$\mathrm{cm}^{\text{new}}_{\text{note}}$
PRF^{η}
$\mathrm{sk}^{a}_{\text{addr}}$
$\mathrm{rt}_{\text{note}}$
u^a
u^b
cred
$\mathrm{pk}^{a}_{\text{addr}}$
$\mathrm{PRF}^{\text{addr}}$
$\mathrm{sk}^{a}_{\text{addr}}$
0
$\mathrm{pos}_{\text{note}}$
cred
$\mathrm{pk}^{b}_{\text{addr}}$
$\mathrm{PRF}^{\text{addr}}$
$\mathrm{sk}^{b}_{\text{addr}}$
0
$\mathrm{note}^{\text{old}}$
$\mathrm{cm}^{\text{old}}_{\text{note}}$
$\mathrm{COMM}^{\text{note}}$
$s^{\text{old}}_{\text{note}}$
v_{note}
Note Merkle tree
$\mathcal{H}$
$\mathrm{note}^{\text{new}}$
$\mathrm{COMM}^{\text{note}}$
$s^{\text{new}}_{\text{note}}$
v_{note}

Figure 4.3: *Version 2* User u_A transfers note $\mathrm{note}^{\text{old}}$ to user u_B and thereby effectively replaces the u_A-owned note $\mathrm{note}^{\text{old}}$ by the u_B-owned note $\mathrm{note}^{\text{new}}$. The transaction tx includes a proof of validity π and a nullifier η. π ensures validity of the new note commitment $\mathrm{cm}^{\text{new}}_{\text{note}}$, presence of $\mathrm{cm}^{\text{note}}_{\text{old}}$ in the Note Merkle tree with root $\mathrm{rt}_{\text{note}}$, and validity of η. η is derived from the unique position of the note in the Merkle tree $\mathrm{pos}_{\text{note}}$ and the sender's secret address key $\mathrm{sk}_{\text{addr}}$. Note commitments $\mathrm{cm}_{\text{note}}$ are now commitments to the note value v_{note} and the public address key $\mathrm{pk}_{\text{addr}}$ of the note owner.

Lastly, the nullifier η will also have to be constructed in a different way. To do this, we introduce another pseudorandom function with seed x: PRF^{η}_{x}. This PRF is required to be collision-resistant to ensure uniqueness of the value of η for different notes. But how do went prevent double spending with this function, and construct a unique nullifier to a spent note? The nullifier η can be determined uniquely for any note by defining it as $\eta := \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{note}})$, where $\mathrm{pos}_{\text{note}}$ is the position of the note in the Note Merkle tree.

Having defined all required building blocks, we can now go about transferring an existing note $\mathrm{note}^{\text{old}} := (s^{\text{old}}_{\text{note}}, v_{\text{note}}, \mathrm{pk}^{a}_{\text{addr}}, \mathrm{cm}^{\text{old}}_{\text{note}})$ from user u_a to another user u_b, with respective key pairs $(\mathrm{sk}^a_{\text{addr}}, \mathrm{pk}^a_{\text{addr}})$ and $(\mathrm{sk}^b_{\text{addr}}, \mathrm{pk}^b_{\text{addr}})$. To create a correct transaction, u_a firstly determines the position $\mathrm{pos}_{\text{note}}$ of $\mathrm{cm}^{\text{old}}_{\text{note}}$ in the Note Merkle Tree. Then, u_a calculates the nullifier to this note as $\eta := \mathrm{PRF}^{\eta}_{\mathrm{sk}^a_{\text{addr}}}(\mathrm{pos}_{\text{note}})$. Subsequently, u_a computes a commitment to the new note with receiver u_b as $\mathrm{cm}^{\text{new}}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s^{\text{new}}_{\text{note}}}(\mathrm{pk}^b_{\text{addr}} \| v_{\text{note}})$, where $s^{\text{new}}_{\text{note}}$ is a randomly sampled value. Lastly, u_a constructs a zk-SNARK proof π for the following statement:

> *"I know s^{old}_{note}, v_{note}, pk^a_{addr}, sk^a_{addr}, cm^{old}_{note}, pos_{note}, pk^b_{addr}, s^{new}_{note} such that the following holds:*

- *The public address key of the sender matches the sender's secret address key: $pk_{addr} = PRF^{addr}_{sk_{addr}}(0)$;*
- *Both commitments are constructed correctly: $cm^{old}_{note} = Comm_{s^{old}_{note}}(pk^a_{addr} \| v_{note})$ and $cm^{new}_{note} = Comm^{note}_{s^{new}_{note}}(pk^b_{addr} \| v_{note})$;*
- *The commitment of the old note cm^{old}_{note} appears at position pos_{note} in the Note Merkle tree;*
- *The nullifier to the old note is constructed correctly: $\eta = PRF^{\eta}_{sk_{addr}}(pos_{note})$."*

Using the above steps, value is transferred from one user to another by creating a new note with the same value. The nullifier to the old note should also be published to ensure that the old note becomes unusable. When u_a has performed all calculations, he or she publishes the transaction $\mathrm{tx} := (\mathrm{cm}^{\text{new}}_{\text{note}}, \eta, \pi)$ to the blockchain. The admin can validate the proof of the transaction and if valid, accept the transaction and add $\mathrm{cm}^{\text{new}}_{\text{note}}$ to the Note Merkle Tree, which finalises the transfer of value from u_a to u_b.

In this construction u_a cannot use the newly created note, since u_a does not know the secret address key of u_b, which is required to construct the nullifier and zero-knowledge proof. Moreover, u_a cannot learn when u_b spends the newly created note, since the nullifier of this new note also depends on the secret address key of u_b. Other users that witness tx know even less about the transaction details, hence they cannot even infer who the sender and receiver are. Namely, the published commitment, nullifier and proof reveal nothing about the address keys or the input note. So we have achieved anonymous transfer of value with the steps as described here and depicted in Figure 4.3.

Version 3: Transferring note secrets. There is still an issue with the previous version of the protocol, namely the receiver of the output note u_b should be able to spend the received note. To do this, u_b can use the same protocol as described above. However, u_b would need some of the secret values that u_a generated to be able to use the new note commitment. Therefore, we need to enable u_a with a way of attaching an encrypted version of these secrets to the transaction, such that they can only be read by u_b, and u_b's anonymity is still maintained.

To enable this additional feature in the protocol, each user should construct an asymmetric encryption key pair $(\mathrm{sk}_{\text{enc}}, \mathrm{pk}_{\text{enc}})$ for a key-private encryption scheme $(\mathrm{IK\text{-}CCA})$[4] to accompany their address key pair. These keys can be derived from the secret address key using a key derivation function (KDF). Similarly to the address key pair, the public encryption key pk_{enc} should be publicly available, and the secret encryption key sk_{enc} not.

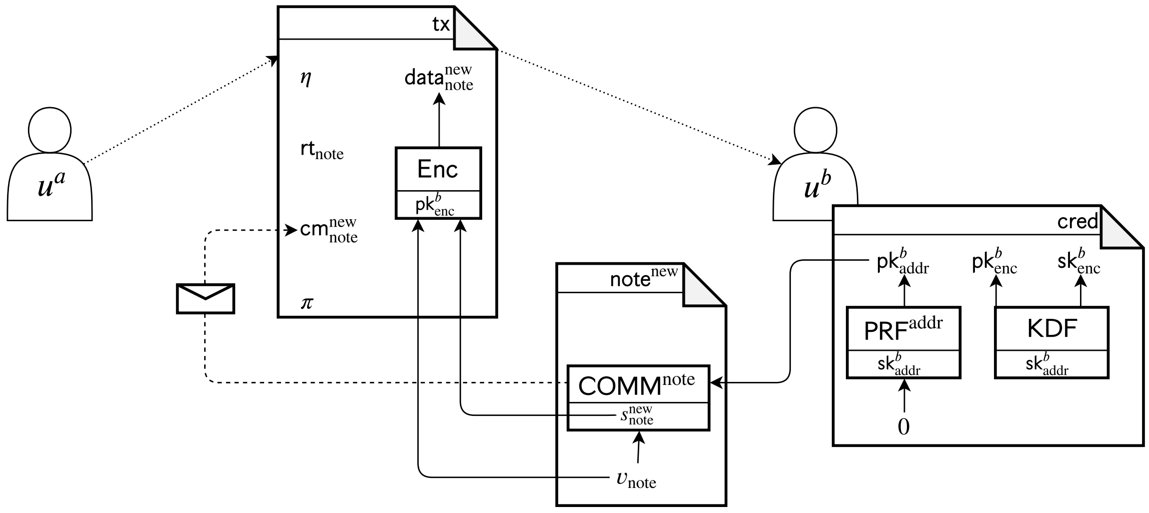

tx
η
$\text{data}^{\text{new}}_{\text{note}}$
rt_{note}
Enc
pk^b_{enc}
$\text{cm}^{\text{new}}_{\text{note}}$
π
u^a
u^b
cred
$\text{pk}^b_{\text{addr}}$
pk^b_{enc}
sk^b_{enc}
PRF^{addr}
$\text{sk}^b_{\text{addr}}$
KDF
$\text{sk}^b_{\text{addr}}$
0
note^{new}
$\text{COMM}^{\text{note}}$
$s^{\text{new}}_{\text{note}}$
v_{note}

Figure 4.4: *Version 3* A transaction from u_A to u_B that creates the ouput note $\mathrm{note}^{\text{new}}$ now also contains a field $\mathrm{data}^{\text{new}}_{\text{note}}$. The field $\mathrm{data}^{\text{new}}_{\text{note}}$ is an encryption of the note value $v^{\text{new}}_{\text{new}}$ and commitment randomness $s^{\text{new}}_{\text{note}}$ under the public encryption key of u_B: $\mathrm{pk}^b_{\text{enc}}$. The asymmetric encryption key pair $(\mathrm{pk}_{\text{enc}}, \mathrm{sk}_{\text{enc}})$ is derived from $\mathrm{sk}_{\text{addr}}$ using a KDF.

As an addition to the steps that u_a takes in the previous version of the protocol, u_a should also compute the ciphertext $\mathrm{data}^{\text{new}}_{\text{note}} := \mathrm{Enc}_{\mathrm{pk}^b_{\text{enc}}}(s^{\text{new}}_{\text{note}} \| v_{\text{note}})$, where $\mathrm{Enc}_x(m)$ is the function that encrypts the message m under the public key x. u_a adds this ciphertext to the transaction tx, as shown in Figure 4.4. u_b can now find the transaction that was sent to him or her by trying to decrypt the field $\mathrm{data}^{\text{new}}_{\text{note}}$ of every transaction that is published on the blockchain. If the decryption is successful, u_b was indeed the receiver of the transaction and also knows the required secrets to be able to spend the newly received note.

This addition of a ciphertext to the publicly visible transaction tx does not infringe on the provided anonymity, since the ciphertext reveals nothing about the used public encryption key, nor does it expose the contents of the original plaintext message.

[4]See Section 2.4 for a definition.

Version 4: Transaction integrity. The above version of the protocol works as expected when everyone behaves honestly and does not try to alter the transaction tx that is published by u_a. However, we cannot be certain of all the participants behaving honestly, therefore we need a way to ensure transaction integrity.

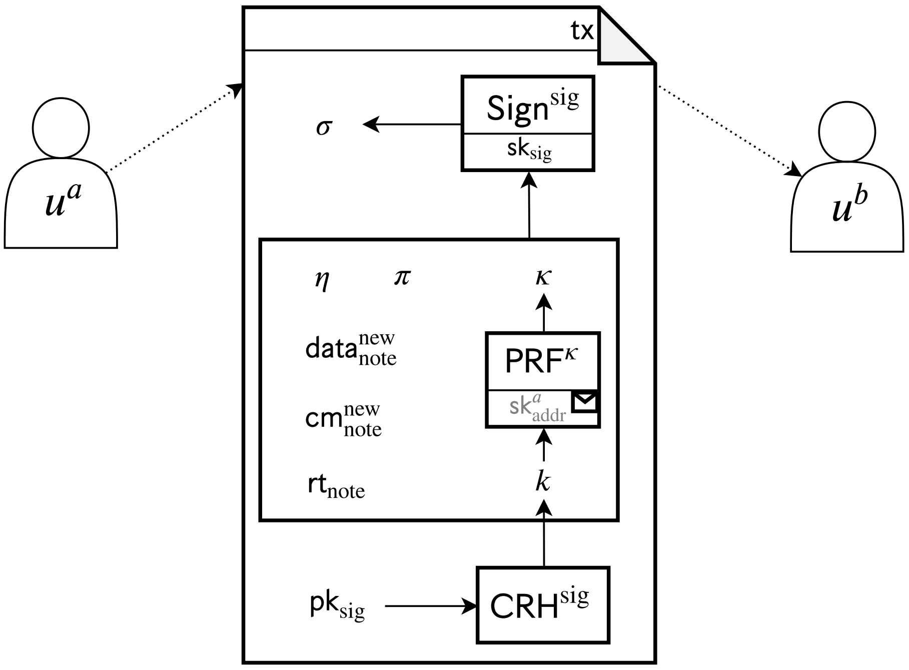

tx
u^a
u^b
σ
$\mathsf{Sign}^{\mathrm{sig}}$
$\mathsf{sk}_{\mathrm{sig}}$
η
π
κ
$\mathsf{data}^{\mathrm{new}}_{\mathrm{note}}$
PRF^{κ}
$\mathsf{sk}^{a}_{\mathrm{addr}}$
$\mathsf{cm}^{\mathrm{new}}_{\mathrm{note}}$
$\mathsf{rt}_{\mathrm{note}}$
k
$\mathsf{pk}_{\mathrm{sig}}$
$\mathsf{CRH}^{\mathrm{sig}}$

Figure 4.5: *Version 4* A transaction from u_A is now signed the secret key $\mathrm{sk}_{\mathsf{sig}}$ of a randomly sampled signature key pair. This signature σ is also included in the transaction tx. The related public key $\mathrm{pk}_{\mathsf{sig}}$ is used to tie the key pair to this message. From $\mathrm{pk}_{\mathsf{sig}}$ we generate a MAC κ through the intermediate value k. All three values are also included in the transaction tx and the computation of κ from k and $\mathrm{sk}_{\mathsf{addr}}$ is verified in the proof π.

To achieve integrity, another cryptographic primitive is needed: strong existential unforgeability against one-time chosen message attack ($\mathrm{SUF\text{-}1CMA}$) scheme. During the construction of a transaction, user u_a randomly samples an appropriate asymmetric signature key pair $(\mathrm{pk}_{\mathsf{sig}}, \mathrm{sk}_{\mathsf{sig}})$ for this scheme and computes $k := \mathrm{CRH}(\mathrm{pk}_{\mathsf{sig}})$ and $\kappa := \mathrm{PRF}^{\kappa}_{\mathrm{sk}^a_{\mathsf{addr}}}(k)$. The intermediate value k is used to ensure that the input to the PRF has the right size and format. The value of κ functions as a message authentication code (MAC). κ is needed to tie the signature key pair to this particular message, since only the sender knows its own secret address key and could have constructed κ correctly. To show that κ has indeed been constructed correctly, this computation should also be included in the zk-SNARK proof π as given in the previous version.

Once u_a has performed all the above steps, see also Figure 4.5, he or she can sign the public parts of the transaction $\mathrm{tx} := (\mathrm{cm}^{\text{new}}_{\text{note}}, \eta, \pi, \mathrm{data}^{\text{new}}_{\text{note}}, \kappa, k)$ using the secret signature key sk_{sig}, thus obtaining signature σ. Finally, u_a publishes the public signature key pk_{sig}, the public transaction values tx and its signature σ.

4.4 Towards an account-based model

In this section we implement the ideas sketched in [35] to move our protocol towards the account-based model. This change allows us to store an account balance for every user on the blockchain. Next to this, it plays an essential role in access control and defining a spend limit later on. What we present here is our take on the ideas as were presented in [35] mixed with the last version of the previous section. Below we present one version that incorporates our take on this at once. Many of the used techniques and data structures should feel familiar to previous versions.

Version 5: Account balance and change. The protocol as described in the previous section seems to provide a secure way of transferring value anonymously from one user to another. However, the attentive reader might have noticed that the transfer of the value in a note does not allow for altering this value, i.e. there is no change. Moreover, in the case of large amounts of transfer one has to send a lot of transactions from one user to another, which is neither convenient nor does it provide a high level of anonymity, i.e. transactions that take place at nearly the same time are probably part of one batch.

In order to solve these problems, we introduce another data structure: private memory cells for the storage of an account balance. These memory cells can be stored in another Merkle tree, which will be called the Memory Merkle tree. Analogously to the Note Merkle tree it will consist of commitments to memory cells, which contain the user's public address key and current account balance v_{mem}: $\mathrm{cm}_{\text{mem}} := \mathrm{Comm}^{\text{mem}}_{s_{\text{mem}}}(\mathrm{pk}_{\text{addr}} \| v_{\text{mem}})$, where s_{mem} is a randomly sampled commitment trapdoor.

Every time a user makes a transaction, he or she should also update his or her own memory cell in order to process the change in account balance. This change in account balance can be positive, negative, or zero, and is used to balance the difference between the value of the input note and that of the output note. To prevent linkability of transactions, these updates consist of publishing a new commitment to the updated account balance and same public address key and publishing a nullifier to the old memory cell such that it becomes invalidated, i.e. it cannot be used anymore. We will refer to a memory cell with mem and to the memory nullifier with μ.

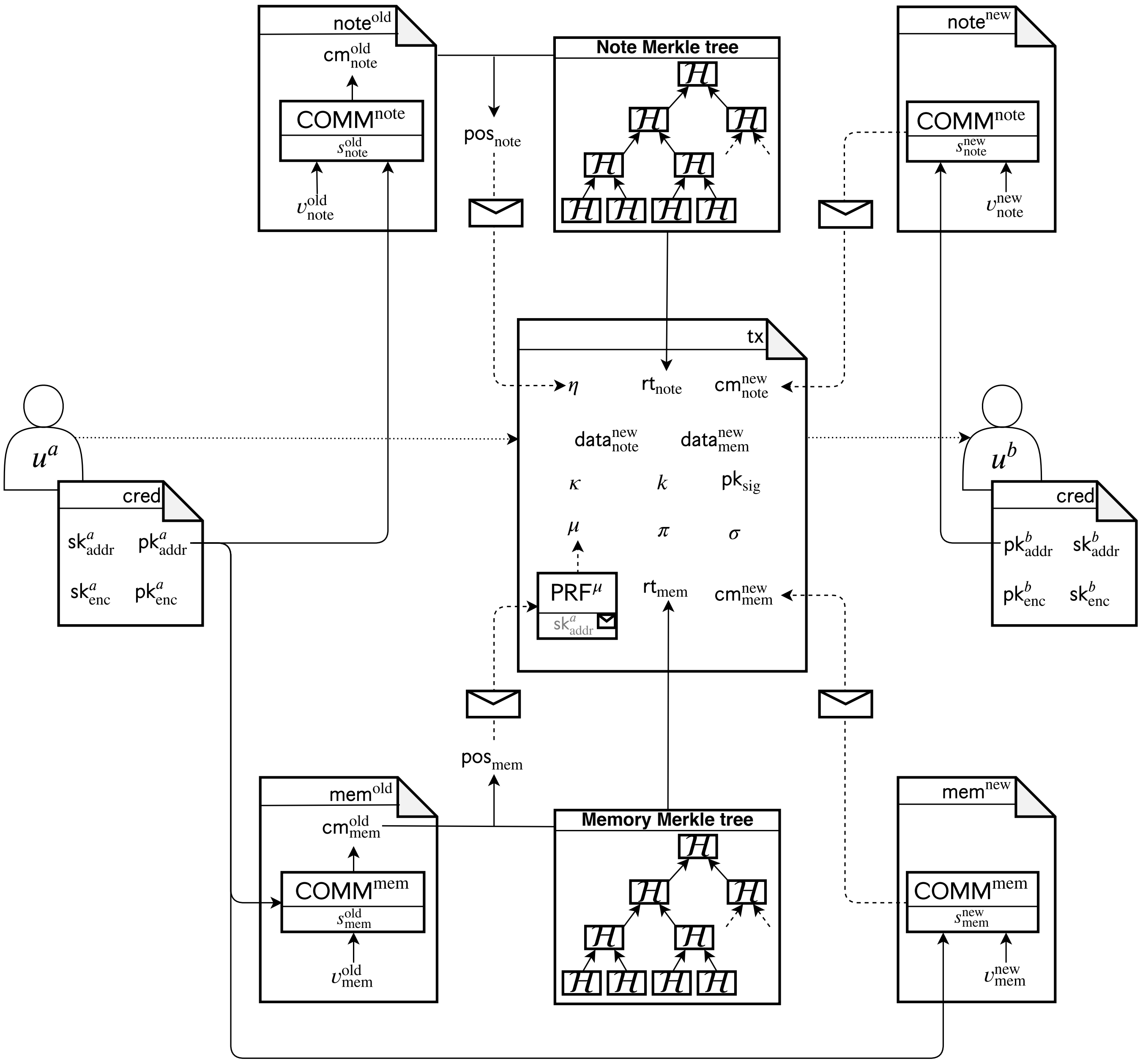
This addition gives rise to several additional and altered steps in the protocol. Where an old transaction had a single input note and a single output note, a new transaction has two inputs and two outputs. These inputs are an old, unused note and the latest memory cell. The outputs are a new note and a new, updated memory cell. Next to the unspent input note $\mathrm{note}^{\text{old}} := (s^{\text{old}}_{\text{note}}, v^{\text{old}}_{\text{note}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}^{\text{old}}_{\text{note}})$, the user also has the latest version of its memory cell $\mathrm{mem}^{\text{old}} := (s^{\text{old}}_{\text{mem}}, v^{\text{mem}}_{\text{old}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}^{\text{old}}_{\text{mem}})$ as input to a transaction. For the construction of the new note everything stays the same except for the fact that the value of this new note $v^{\text{new}}_{\text{note}}$ can now be any positive value, as long as the account balance does not become negative. This updated account balance is calculated as $v^{\text{new}}_{\text{mem}} := v^{\text{old}}_{\text{note}} + v^{\text{old}}_{\text{mem}} - v^{\text{new}}_{\text{note}}$. The commitment to the new memory cell is then computed as $\mathrm{cm}^{\text{new}}_{\text{mem}} := \mathrm{Comm}^{\text{mem}}_{s^{\text{new}}_{\text{mem}}}(\mathrm{pk}_{\text{addr}} \| v^{\text{new}}_{\text{mem}})$, where $s^{\text{new}}_{\text{mem}}$ is a new randomly sampled trapdoor and $\mathrm{pk}_{\text{addr}}$ is the sender's public address key. Obviously, the old memory cell should also be invalidated, hence we need to calculate and publish the nullifier to this. This nullifier μ is computed analogously to the nullifier of the old note as $\mu := \mathrm{PRF}^{\mu}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{mem}})$, where $\mathrm{pos}_{\text{mem}}$ is the position of the commitment to the old memory cell in the Memory Merkle tree. A sketch of this updated construction is given in Figure 4.6.

Finally, the user also needs to proof some more statements in the zk-SNARK proof π. These additional statements are:

" I also know pos_{mem}, v^{new}_{note}, v^{mem}_{old}, s^{old}_{mem}, cm^{old}_{mem}, s^{new}_{mem} such that the following additional statements hold:

- *The account balance is updated properly: $v^{new}_{mem} = v^{old}_{note} + v^{old}_{mem} - v^{new}_{note}$;*
- *The updated account balance is non-negative $v^{new}_{mem} \geq 0$;*
- *Both memory commitments are constructed correctly: $cm^{old}_{mem} = Comm^{mem}_{s^{old}_{mem}}(pk_{addr} \| v^{old}_{mem})$ and $cm^{new}_{mem} = Comm^{mem}_{s^{new}_{mem}}(pk_{addr} \| v^{new}_{mem})$;*
- *The commitment of the old memory cell cm^{old}_{mem} appears at position pos_{mem} in the Memory Merkle tree;*
- *The nullifier to the old note is constructed correctly: $\mu = PRF^{\mu}_{sk_{addr}}(pos_{mem})$."*

Additionally to what is already included in the transaction tx in previous versions, we now also publish (and sign) μ, $\mathrm{cm}^{\text{new}}_{\text{mem}}$ and the updated proof π. It should be noted that in the case that a user does not have a previous memory cell, he or she can define $\mathrm{pos}_{\text{mem}} := -1$ and $v^{\text{old}}_{\text{mem}} := 0$. These additional steps should also be included in the zk-SNARK proof.



note^{old}
$\text{cm}^{\text{old}}_{\text{note}}$
$\text{COMM}^{\text{note}}$
$s^{\text{old}}_{\text{note}}$
$v^{\text{old}}_{\text{note}}$
pos_{note}
Note Merkle tree
$\mathcal{H}$
note^{new}
$\text{COMM}^{\text{note}}$
$s^{\text{new}}_{\text{note}}$
$v^{\text{new}}_{\text{note}}$
tx
η
rt_{note}
$\text{cm}^{\text{new}}_{\text{note}}$
$\text{data}^{\text{new}}_{\text{note}}$
$\text{data}^{\text{new}}_{\text{mem}}$
κ
k
pk_{sig}
μ
π
σ
PRF^{μ}
$\text{sk}^{a}_{\text{addr}}$
rt_{mem}
$\text{cm}^{\text{new}}_{\text{mem}}$
u^a
cred
$\text{sk}^{a}_{\text{addr}}$ $\text{pk}^{a}_{\text{addr}}$
$\text{sk}^{a}_{\text{enc}}$ $\text{pk}^{a}_{\text{enc}}$
u^b
cred
$\text{pk}^{b}_{\text{addr}}$ $\text{sk}^{b}_{\text{addr}}$
$\text{pk}^{b}_{\text{enc}}$ $\text{sk}^{b}_{\text{enc}}$
pos_{mem}
mem^{old}
$\text{cm}^{\text{old}}_{\text{mem}}$
COMM^{mem}
$s^{\text{old}}_{\text{mem}}$
$v^{\text{old}}_{\text{mem}}$
Memory Merkle tree
mem^{new}
COMM^{mem}
$s^{\text{new}}_{\text{mem}}$
$v^{\text{new}}_{\text{mem}}$

Figure 4.6: *Version 5* A transaction tx now also has an input memory cell mem^{old} and an output memory cell mem^{new}. The commitment to a memory cell cm_{mem} is computed from the owner's public address key pk_{addr} and the currenct account balance v_{mem}. These commitments are contained in the newly introduced Memory Merkle tree, and the position in this tree pos_{mem} is used to compute the memory nullifier μ. Finally, this version allows the output note to take an arbitrary value as long as the account balance does not become negative and balance is maintained: $v^{\text{new}}_{\text{mem}} := v^{\text{old}}_{\text{note}} + v^{\text{old}}_{\text{mem}} - v^{\text{new}}_{\text{note}}$. The computations of these new values are all checked in the zero-knowledge proof π and the public values are visible on the blockchain.

Remembering the commitment trapdoor that is used for the new memory commitment is not very practical. Fortunately, there exists a rather simple solution to this. The sender can include the encrypted version of the commitment trapdoor in the transaction, by encrypting it under the sender's own public encryption key. To make this precise, the field $\mathrm{data}^{\mathsf{new}}_{\mathsf{mem}} := \mathrm{Enc}_{\mathrm{pk}_{\mathsf{enc}}}(s^{\mathsf{new}}_{\mathsf{mem}})$ should be added to tx. This ensures that the user can always find and decrypt this field in the case that commitment trapdoor to the latest memory cell is forgotten.

4.5 Access control

Two new versions are presented in this section. Together, these versions implement the *access control* requirement. We will make use of the elements that were presented in previous sections to ensure that only clients of the banks that participate in the system can make transactions. In general terms this means that any potential user needs to be approved by one of the admin(istrator)s of the payment system. The request for an 'account' on the blockchain is also the point were the administrator can perform KYC or customer due diligence (CDD) before granting access. This process is also roughly depicted in the overview in Figure 4.1.

In the first version we will describe the initial process of getting an 'account' on the blockchain. The second version focuses on linking this account to a transaction, such that it can be verified by any user and admin whether or not the creator of a transaction has passed the access control, i.e. holds an 'account'.

Version 6: Access control. The protocol as presented in earlier sections has not yet touched upon the topic of access control. A specific set of actors, i.e. the banks that implement this protocol and control the underlying fiat currency payment system, function as administrators in the protocol. These administrators should be able to have control over who can use this payment scheme and should also be able to perform CDD in case a new user wants to use the system. In this new version, we will add the necessary improvements to achieve this. We will assume for now that the administrators function as one actor, however the protocol as presented can be easily extended to a group of administrators that communicate the relevant information.

In order to let the admin function as a gatekeeper to the digital transaction system, we introduce one more Merkle tree: the Account Merkle Tree – sometimes also called Credential Merkle tree. This Account Merkle tree stores commitments to all the accounts that are allowed to perform transactions in the system. Specifically, it stores commitments of the form $\mathrm{cm}_{\mathsf{cred}} := \mathrm{Comm}^{\mathsf{cred}}_{s_{\mathsf{cred}}}(\mathrm{pk}_{\mathsf{addr}} \| \mathrm{sk}_{\mathsf{addr}})$ for every accepted user u, where $(\mathrm{pk}_{\mathsf{addr}}, \mathrm{sk}_{\mathsf{addr}})$ is u's address key pair, or credentials. By giving the admin control over which credentials are allowed in the Account Merkle tree, the admin can also control which users can perform transactions and which not. Additionally, the admin is required to construct an asymmetric key pair for a strong existential unforgeability against chosen message attack ($\mathrm{SUF\text{-}CMA}$) signature scheme to be able to publish and sign messages as the admin.

Implementation-wise, this means that whenever a new user u wants to join the system, u should ask an admin to get an entry in the Account Merkle Tree. To do this, u first constructs his or her address key pair $(\mathrm{pk}_{\text{addr}}, \mathrm{sk}_{\text{addr}})$ and encryption key pair $(\mathrm{pk}_{\text{enc}}, \mathrm{sk}_{\text{enc}})$. Subsequently, u randomly samples a commitment trapdoor s_{cred} and computes the credential commitment $\mathrm{cm}_{\text{cred}} := \mathrm{Comm}^{\text{cred}}_{s_{\text{cred}}}(\mathrm{pk}_{\text{addr}} \| \mathrm{sk}_{\text{addr}})$. u also constructs a zk-SNARK proof π for the following statement: *"I know s_{cred} and sk_{addr} such that pk_{addr} and cm_{cred} are constructed correctly."*. The user u then sends his or her public address key, commitment, and the proof π to the admin. To give some more insight, this step is also depicted in Figure 4.7. On reception of this message, the admin verifies the proof, checks that u does not have an account yet, and performs the required CDD. When everything is correct, the admin adds the commitment $\mathrm{cm}_{\text{cred}}$ to the Account Merkle tree. In addition, the admin publishes a message on the blockchain, in which the addition of the commitment is stated. The message is signed under the admin's secret signature key, that can be verified by any user using the admin's public key.

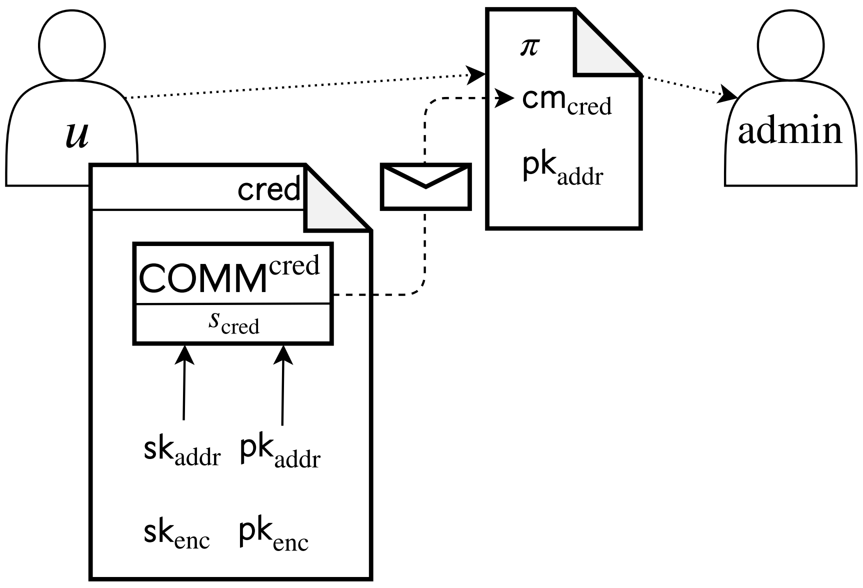

u
cred
$\mathrm{COMM}^{\mathrm{cred}}$
s_{cred}
$\mathrm{sk}_{\mathrm{addr}}$
$\mathrm{pk}_{\mathrm{addr}}$
$\mathrm{sk}_{\mathrm{enc}}$
$\mathrm{pk}_{\mathrm{enc}}$
π
$\mathrm{cm}_{\mathrm{cred}}$
$\mathrm{pk}_{\mathrm{addr}}$
admin

Figure 4.7: *Version 6* User u requests access to the payment scheme by sending a commitment to his credentials to the admin. The commitments is only to u's address key pair. Along with this commitment, u also sends a proof of correct computation of this computation and of the correct construction of $\mathrm{pk}_{\text{addr}}$ from $\mathrm{sk}_{\text{addr}}$.

The admin can revoke an account, if this is needed, by publishing a similar message on the blockchain. The admin can in such a case simply remove the appropriate credential commitment $\mathrm{cm}_{\textsf{cred}}$ from the Account Merkle Tree and publish a message, signed by the admin, that the credentials belonging to that commitment have been revoked.

Version 7: Proof of access. The access control feature as described above does not yet provide linkage with performing a transaction as defined before. This implies that it is currently still possible to publish a transaction without an 'account' in the Account Merkle tree. In this version of the protocol, the connection between both parts is established which leads us to an anonymous digital payment protocol with access control.

Given a user u, with credentials $(\mathrm{pk}_{\textsf{addr}}, \mathrm{sk}_{\textsf{addr}})$ and credential commitment $\mathrm{cm}_{\textsf{cred}}$ present in the Account Merkle tree, who wants to perform a transaction to another user. The steps as described in earlier versions are all still valid, the only thing that really changes in this version is the zk-SNARK proof π. Some additional statements need to be proved there. These additional statements are: *"I know pk_{addr}, sk_{addr}, cm_{cred}, and s_{cred}, such that cm_{cred} is constructed correctly and is contained in the Account Merkle tree."* Since these credential values are also used in other parts of the transaction, i.e. constructing the nullifier, using the old note, et cetera, all steps are tied together by means of the proof π. This connection implies that a user can only perform a valid transaction, i.e. construct a true proof π, when the user's credentials are acknowledged in the Account Merkle tree.

Because of the inclusion of the public address key $\mathrm{pk}_{\textsf{addr}}$ in the credential commitment, one statement of the earlier version of this protocol can be removed from the proof. This concerns the statement: *"I know pk_{addr}, sk_{addr} such that $pk_{addr} = PRF^{addr}_{sk_{addr}}(0)$."* This statement has become redundant since it is already included in the zk-SNARK proof required for getting one's credential acknowledged. We see here that including the user's public address key in the credential commitment and validating the construction before acceptance gives a slight improvement to the performance over not doing this.

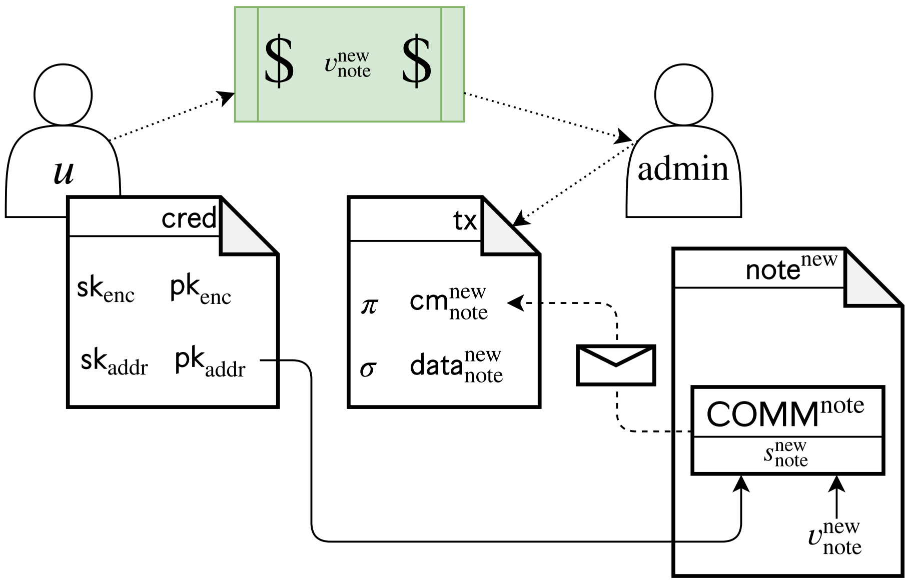
4.6 Conversion to and from fiat currency

The *Conversion to/from fiat currency* requirement is implemented in this section. The backing pool of fiat currency was mentioned in several of the versions in earlier sections. Up to now, we have not discussed how to link fiat currency to this backing pool. Below, we present two new versions that describe how to convert fiat currency into anonymous notes and the other way around.

Version 8: Fiat currency to anonymous notes. In this version we describe how one should actually create a new anonymous note. In earlier version, we assumed that it was possible to pay some amount of fiat currency to a backing pool and then be allowed to construct an anonymous note with the corresponding value. In this version of the protocol, the steps required to create an new anonymous note are set out.

If a user u wants to have more value that can be transferred anonymously, u should obtain a new note with a certain value $v_{\text{note}}^{\text{new}}$. Before we get started on how to do this, it is important to remark that there are two ways to spend such a new note: the note can be used as an input note either when u wants to transfer money to another user, or in a transaction with an output note of value 0 which adds the full value of the input note to u's account balance.

To obtain a new note with value $v_{\text{note}}^{\text{new}}$, the user u pays the amount $v_{\text{note}}^{\text{new}}$ in fiat currency to the admin. Subsequently, the admin constructs a new note with value $v_{\text{note}}^{\text{new}}$ destined for the public address key $\mathrm{pk}_{\text{addr}}$ of the user u. Explicitly, the admin computes $\mathrm{cm}_{\text{note}}^{\text{new}} := \mathrm{Comm}_{s_{\text{note}}^{\text{new}}}^{\text{note}}(\mathrm{pk}_{\text{addr}} \| v_{\text{note}}^{\text{new}})$, with $s_{\text{note}}^{\text{new}}$ a randomly sampled commitment trapdoor. The admin also computes the encrypted commitment secrets, analogously to a user-to-user transfer, as $\mathrm{data}_{\text{note}}^{\text{new}} := \mathrm{Enc}_{\mathrm{pk}_{\text{enc}}}(s_{\text{note}}^{\text{new}} \| v_{\text{note}}^{\text{new}})$, where pk_{enc} is the public encryption key of u.



$ $v_{\text{note}}^{\text{new}}$ $
u
admin
cred
sk$_{\text{enc}}$ pk$_{\text{enc}}$
sk$_{\text{addr}}$ pk$_{\text{addr}}$
tx
π cm$_{\text{note}}^{\text{new}}$
σ data$_{\text{note}}^{\text{new}}$
note$^{\text{new}}$
COMM$^{\text{note}}$
$s_{\text{note}}^{\text{new}}$
$v_{\text{note}}^{\text{new}}$

Figure 4.8: *Version 8* User u sends fiat currency (green dollar bill) of value $v_{\text{note}}^{\text{new}}$ to the admin. In return the admin publishes a transaction tx that refers to a newly created note with commitment $\mathrm{cm}_{\text{note}}^{\text{new}} := \mathrm{Comm}_{s_{\text{note}}^{\text{new}}}^{\text{note}}(\mathrm{pk}_{\text{addr}} \| v_{\text{note}}^{\text{new}})$ and value $v_{\text{note}}^{\text{new}}$.

In addition to these computations, the admin constructs a zk-SNARK proof π for the following statement: *"I know s^{new}_{note}, v^{new}_{note}, pk_{addr} such that cm^{new}_{note} is calculated correctly."*. As a last step, the admin publishes the new commitment, the proof π and the encrypted data $\mathrm{data}^{\text{new}}_{\text{note}}$, all signed under the admin's secret signature key to the blockchain. The new commitment then also gets added to the Note Merkle tree. The user u that sees this publication, can decrypt the field $\mathrm{data}^{\text{new}}_{\text{note}}$ to ascertain that the transaction was sent to u and use this note as input for a future transaction. Figure 4.8 gives a schematic representation of these steps.

Version 9: Anonymous notes to fiat currency. The only building block that is still missing in the basis protocol, is that of converting one's anonymous notes back to fiat currency. The required techniques and computations to do this are already present in the protocol, in this step we will explain how to combine these parts to achieve conversion from anonymous notes to fiat currency.

Given a user u that wants to convert a received note of value $v^{\text{old}}_{\text{note}}$ to fiat currency.[5] u should perform a normal transaction to the public address key of the admin, with one simple addition. The admin should know which user sent the note. This is achieved by adding u's public address key $\mathrm{pk}_{\text{addr}}$ to the field with encrypted secrets as $\mathrm{data}^{\text{old}}_{\text{note}} := \mathrm{Enc}_{\mathrm{pk}^{\text{old}}_{\text{enc}}}(s^{\text{old}}_{\text{note}} \| v^{\text{old}}_{\text{note}} \| \mathrm{info})$, where info contains $\mathrm{pk}_{\text{addr}}$ and possibly some extra information.

The rest of the conversion is now up to the admin who finds this transaction on the blockchain. After parsing the received note, the admin has a couple of computations to make. The admin calculates the nullifier η to the received note in the usual way, and then computes a zero-knowledge proof π for the following statement:

"I know sk_{addr}, cm^{old}_{note}, v^{old}_{note} , pos_{note}, s^{old}_{note} such that the following statements hold:

- $\mathrm{cm}^{\text{old}}_{\text{note}}$ is present in the Note Merkle Tree at $\mathrm{pos}_{\text{note}}$;
- The note nullifier is calculated correctly: $\eta = \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{note}})$;
- The commitment is calculated correctly: $\mathrm{cm}^{\text{old}}_{\text{note}} = \mathrm{Comm}^{\text{note}}_{s^{\text{old}}_{\text{note}}}(\mathrm{pk}_{\text{addr}} \| v^{\text{old}}_{\text{note}})$;
- The secret address key has been used to construct the public address key: $\mathrm{pk}_{\text{addr}} = \mathrm{PRF}^{\text{addr}}_{\mathrm{sk}_{\text{addr}}}(0)$.

As shown in Figure 4.9, the admin then publishes the nullifier η, zero-knowledge proof π and public values in one message signed under the admin's public signature key. Once this message is accepted on the blockchain, the admin transfers the note value $v^{\text{old}}_{\text{note}}$ in fiat currency to the user corresponding to the public address key $\mathrm{pk}_{\text{addr}}$. Remember that u enclosed $\mathrm{pk}_{\text{addr}}$ in the $\mathrm{data}^{\text{new}}_{\text{note}}$ field. The admin can identify the

[5]The explained steps also work for converting (part of) the account balance.

user that belongs to this public address key, since the admin also received the same address key for the user on requesting an account in the payment scheme.

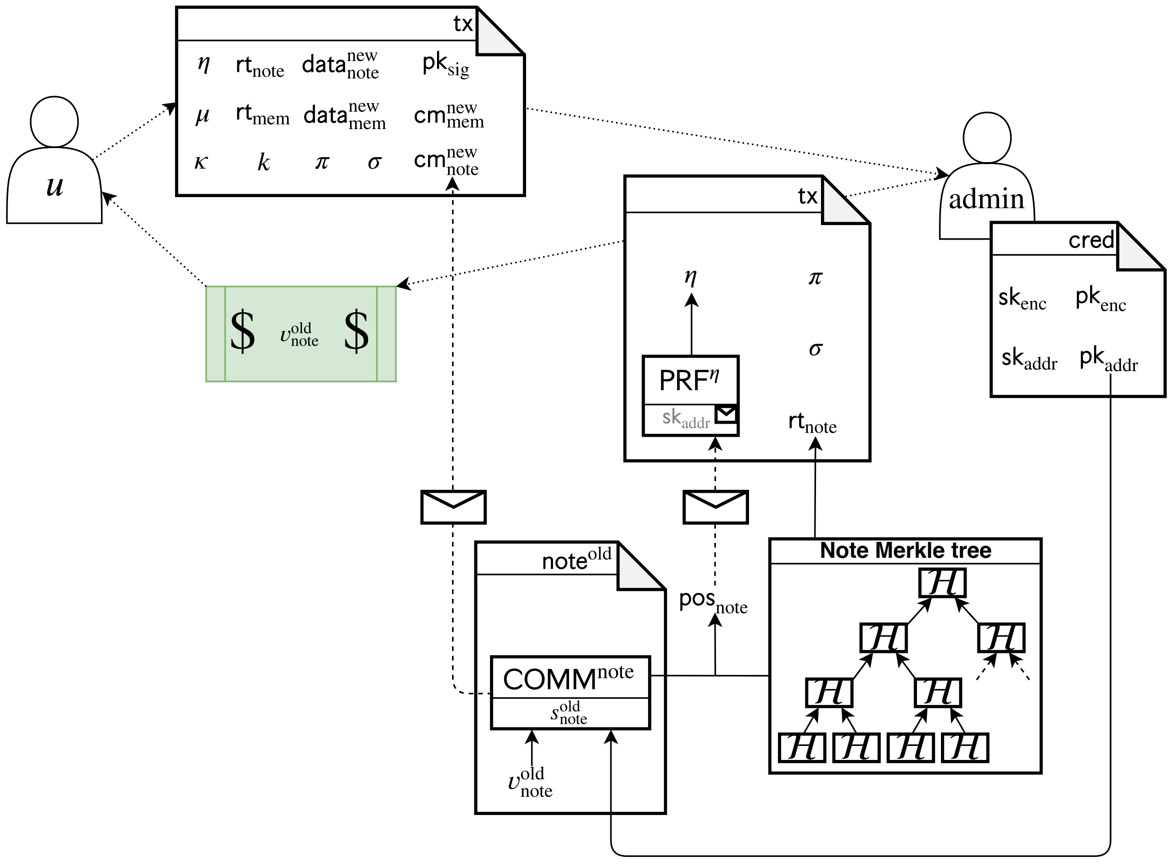

tx
η $\mathsf{rt}_{\mathrm{note}}$ $\mathsf{data}^{\mathrm{new}}_{\mathrm{note}}$ $\mathsf{pk}_{\mathrm{sig}}$
μ $\mathsf{rt}_{\mathrm{mem}}$ $\mathsf{data}^{\mathrm{new}}_{\mathrm{mem}}$ $\mathsf{cm}^{\mathrm{new}}_{\mathrm{mem}}$
κ k π σ $\mathsf{cm}^{\mathrm{new}}_{\mathrm{note}}$
u
admin
$\$$ $v^{\mathrm{old}}_{\mathrm{note}}$ $\$$
tx
η π σ $\mathsf{rt}_{\mathrm{note}}$
PRF^{η}
$\mathsf{sk}_{\mathrm{addr}}$
cred
$\mathsf{sk}_{\mathrm{enc}}$ $\mathsf{pk}_{\mathrm{enc}}$
$\mathsf{sk}_{\mathrm{addr}}$ $\mathsf{pk}_{\mathrm{addr}}$
$\mathsf{note}^{\mathrm{old}}$
$\mathsf{COMM}^{\mathrm{note}}$
$s^{\mathrm{old}}_{\mathrm{note}}$
$v^{\mathrm{old}}_{\mathrm{note}}$
$\mathsf{pos}_{\mathrm{note}}$
Note Merkle tree
$\mathcal{H}$

Figure 4.9: *Version 9* User u sends a note $\mathrm{note}^{\mathsf{old}}$ with commitment $\mathrm{cm}_t^{\mathsf{new}} extnote$ to the admin in the top-left transaction tx. The admin subsequently publishes a new *convert* transaction signed under his key, that publishes the nullifier η to $\mathrm{note}^{\mathsf{old}}$ and a proof π that validates the transaction. The admin also deposits fiat currency with value $v^{\mathsf{old}}_{\mathsf{note}}$ in u's regular bank account to complete the conversion.

4.7 Transaction limit

In this section we present the incorporation of the *spend limit* requirement. The one new version presented below is dedicated to adjusting the protocol such that this requirement is satisfied as well. Remember that the goal of the requirement is to limit the amount of value that any user can transfer anonymously during a predetermined time span.

To explain the required additions to gain this additional feature, we will build upon the latest version of the solution concept as presented in the previous section. Up to now, we have seen four types of transaction: (1) A regular transfer transaction; (2) An account creation transaction; (3) A transaction that converts fiat currency to notes; and (4) A transaction that converts anonymous notes to fiat currency. Below, we explain the additional steps that a user has to take when creating a new regular transfer transaction. In addition, we discuss the few steps of the algorithm that need to be adapted to limit the amount a user can spend in a predefined time frame. Since a user can only transfer notes anonymously by means of a regular transaction, the steps for conversion and account creation transactions need not be altered.

Version 10: Spend limit. To limit the amount of value a user can spend anonymously in a certain time frame, we do not need to add new building blocks. A few adaptations to existing commitments and algorithm steps will do. The only algorithm that really needs adaptations is that of a regular transfer transaction, since this is the type of transaction we actually want to limit.

Say we want to limit the amount that any user can spend during any time span of size T to L. To do this, the user u should proof, in every transaction where u is the sender, that the value of u's new transaction $v^{\text{note}}_{\text{new}}$ plus the amount of all transactions u spent between now (t_{now}) and T time ago ($t_{\text{now}} - T$) is not more than L. In order to proof this, the user needs to add some more information to his or her memory cell and also make use of another old memory cell for construction of the zero-knowledge proof π. We will denote this second memory cell as the 'ceiling' (or 'ceil') memory cell.

Each memory cell gets the following additional fields: c for the total amount of value transferred anonymously up to and including the transaction that caused this memory cell update and t for the time this memory cell got updated. To implement this efficiently, we add c to the commitment of the memory cell, whereas t is added to the Merkle Tree together with the commitment. The commitment of a memory cell is now constructed as $\mathrm{cm}_{\text{mem}} = \mathrm{Comm}^{\text{mem}}_{s_{\text{mem}}}(\mathrm{pk}_{\text{addr}} \| v_{\text{mem}} \| c)$. When a commitment is added to the memory tree, it gets added together with the time the transaction was accepted on the blockchain as a leaf to the Memory Merkle tree, we call this time t. Explicitly, we compute the Merkle leaf as $\mathrm{CRH}^{\text{mem}}(\mathrm{cm}_{\text{mem}} \| t)$ using a new hash function $\mathrm{CRH}^{\text{mem}}$. This is instead of adding just the commitment as a leaf to the Memory Merkle tree as was the case in previous versions. The new Memory Merkle tree and memory cell are depicted in Figure 4.10.

When a user u wants to send a new note with value $v^{\text{new}}_{\text{note}}$ at time t^{new}, u needs to have two memory cells: the latest memory cell $\mathrm{mem}^{\text{old}} = (s^{\text{old}}_{\text{mem}}, v^{\text{old}}_{\text{mem}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}^{\text{old}}_{\text{mem}}, c^{\text{old}}, t^{\text{old}})$ and a memory cell $\mathrm{mem}^{\text{ceil}} = (s^{\text{ceil}}_{\text{mem}}, v^{\text{ceil}}_{\text{mem}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}^{\text{ceil}}_{\text{mem}}, c^{\text{ceil}}, t^{\text{ceil}})$, with $t^{\text{ceil}} < t^{\text{new}} - T$. When the limit is defined as L, u needs to ensure that $c^{\text{old}} - c^{\text{ceil}} + v^{\text{new}}_{\text{note}} \leq L$. u also has to proof this in the zero-knowledge proof π, but we first discuss how u

needs to compute the commitment to the updated memory cell.

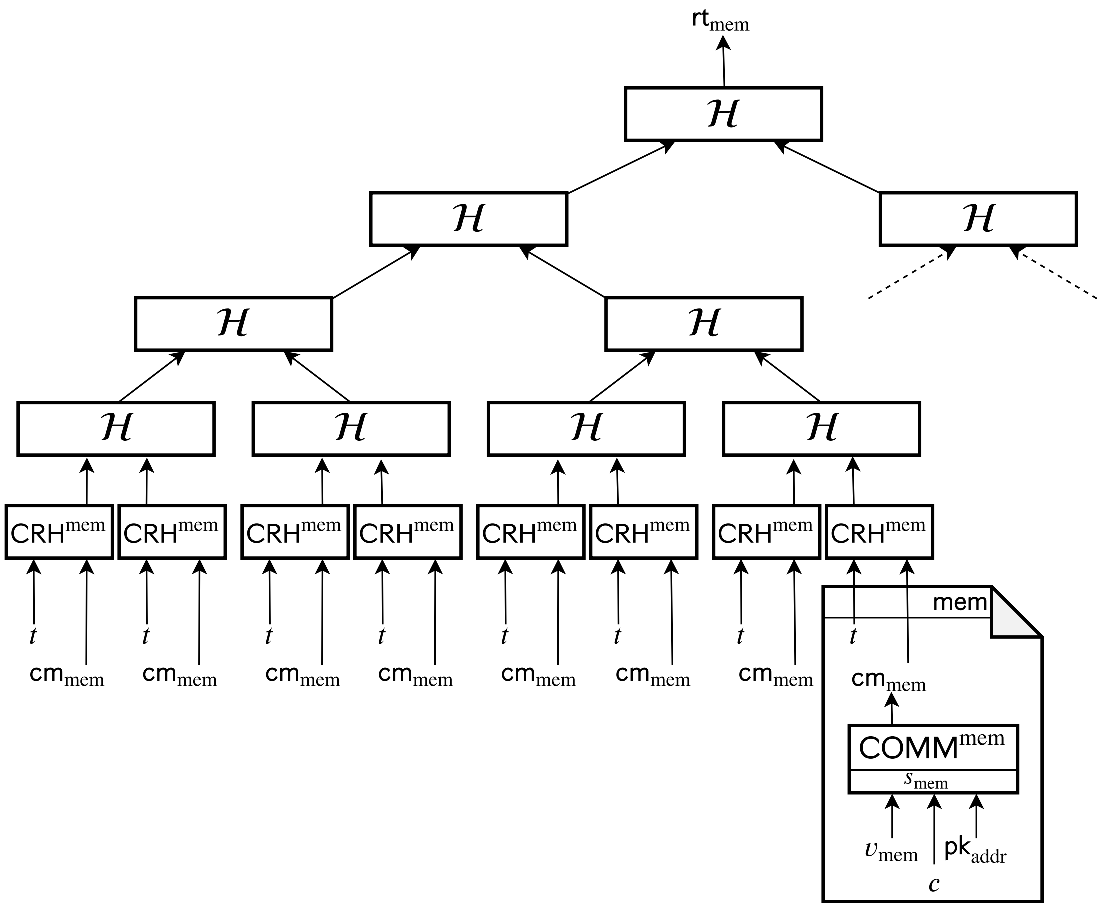


Figure 4.10: *Version 10* The commitment cm_{mem} to memory cell is now also a commitment to the value c: the total amount amount of value transferred up to and including the transaction including this memory cell. Next to this a leaf of the Memory Merkle tree is now the hash of the time t and commitment cm_{mem} instead of just cm_{mem}. t is the time that mem got added to the Memory Merkle tree.

Before the commitment is calculated, u needs to compute $c^{\text{new}} := c^{\text{old}} + v^{\text{new}}_{\text{note}}$. u then computes the commitment to the new memory cell in the same way as before, but includes the additional field c: $\text{cm}^{\text{new}}_{\text{mem}} := \text{Comm}^{\text{mem}}_{s^{\text{new}}_{\text{mem}}}(\text{pk}_{\text{addr}} \| v^{\text{new}}_{\text{mem}} \| c^{\text{new}})$. Finally, u can construct the zk-SNARK proof π, where most statements are the same as before, apart from the (commitments to and membership proofs of) memory cells that have been updated to include the newly defined values: c and t. Specifically, the following additional statements are included in the proof π:

"I also know cm^{ceil}_{mem}, t^{ceil}, c^{old}, c^{ceil}, v^{ceil}_{mem}, cm^{ceil}_{mem}, s^{ceil}_{mem}, c^{new} such that the following holds:

- *The commitment to the ceiling memory cell is constructed correctly:* $cm^{\textit{ceil}}_{\textit{mem}} := Comm^{\textit{mem}}_{s^{\textit{ceil}}_{\textit{mem}}}(pk_{\textit{addr}} \| v^{\textit{ceil}}_{\textit{mem}} \| c^{\textit{ceil}})$*;*
- *The commitment of the ceiling memory cell* $(cm^{\textit{ceil}}_{\textit{mem}}, t^{\textit{ceil}})$ *is included in the Memory Merkle tree;*
- *The amount of value spent is not above the limit:* $c^{\textit{old}} - c^{\textit{ceil}} + v^{\textit{new}}_{\textit{note}} \leq L$*;*
- *The ceiling memory cell is old enough:* $t^{\textit{ceil}} < t^{\textit{new}} - T$*;*
- *Correct calculation of new total outgoing value:* $c^{\textit{new}} = c^{\textit{old}} + v^{\textit{new}}_{\textit{note}}$*."*

It should be noted that u should define t^{new} as the next block time, i.e. the time a next block in which this transaction will be included is published.[6]

4.8 Auditability

In this section we explain how we include the *auditability* requirement. We satisfy the requirement by the inclusion of the verifiable encryption scheme SAVER in our existing protocol, building upon all previous version. For a detailed explanation of SAVER we refer the reader to Section 2.3. The addition of SAVER needs to be discussed along two lines, the adaptation to the transactions and the key generation and sharing amongst judges. We thus introduce two new versions of the protocol. The first new version looks into the topic of enclosing sender identity such that one trusted authority can, if necessary, learn this identity. The second version will focus on replacing this single trusted authority with a group of N judges.

Version 11: Enclosing encrypted transaction details. When a user u wants to send a transaction tx that does not adhere to the spend limit, he should be forced to include the transaction details of tx in such a way that a trusted authority can view these details in case of doubts about u's intentions. Clearly, u does not want to reveal these details to any other parties involved in the transaction scheme, hence he wants to hide the enclosed details in such a way that only the trusted authority can view them. We propose, that u encrypts the transaction details using SAVER and the public key pk_{svr} of the trusted authority.

Specifically, this means that during the setup of the payment scheme the trusted authority should generate a SAVER key triplet $(\mathrm{pk}_{\text{svr}}, \mathrm{vk}_{\text{svr}}, \mathrm{sk}_{\text{svr}})$, using the public parameters of the zk-SNARK scheme. The public key pk_{svr} and verification key vk_{svr} should be made publicly available. The secret SAVER key sk_{svr} should be kept secret

[6]An implementation of this scheme could also define the time a block is released as the block number, since there is a one-on-one relation between the two.

at all times. Since the spend limit only influences the regular transfer transactions, we only need to adapt this type of transactions. The conversion and account creation transaction remain unaltered.

When u surpasses the spend limit on a transaction tx, he should define some additional values that are copies of the details that are relevant to tx. In specific, u defines fields for: (1) his or her (sender) public key $\overline{\mathrm{pk}}_{\mathsf{sndr}} := \mathrm{pk}_{\mathsf{addr}}$; (2) the receiver's public key $\overline{\mathrm{pk}}_{\mathsf{rcvr}} := \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}$; and (3) the value that is transferred $\overline{v}_{\mathsf{xfer}} := v^{\mathsf{new}}_{\mathsf{note}}$. If u does not surpass the spend limit, he should define these values in such a way that they decrypt to rubbish. However, the zk-SNARK proof should still be verifiable, and the trusted authority should be able to distinguish a decryption of rubbish from one of valid contents. Therefore, u should define $\overline{\mathrm{pk}}_{\mathsf{sndr}} := 0$, $\overline{\mathrm{pk}}_{\mathsf{rcvr}} := 0$, and $\overline{v}_{\mathsf{xfer}} := 0$, if the spend limit is not surpassed in tx.[7]

We also need to ensure that a transaction with valid enclosed details does not count towards the spend limit. Therefore, the user u should set $c^{\mathsf{new}} := c^{\mathsf{old}}$, instead of $c^{\mathsf{old}} + v^{\mathsf{new}}_{\mathsf{note}}$, when u's encrypted details are added to the transaction.

The zk-SNARK proof π should again be updated to include the new conditional statements. Specifically, the arithmetic circuit should include the following additional/updated statements:

"In addition to the previous statements EITHER the first statement:

- *The encrypted values are null strings: $\overline{pk}_{sndr} := 0$, $\overline{pk}_{rcvr} := 0$, and $\overline{v}_{xfer} := 0$;*

OR the second set of statements:

- *The encrypted values are correct: $\overline{pk}_{sndr} := pk_{addr}$, $\overline{pk}_{rcvr} := pk^{new}_{addr}$, and $\overline{v}_{xfer} := v^{new}_{note}$;*
- ***updated:** Spend limit is removed: $\xcancel{c^{old} - c^{ceil} + v^{new}_{note} \leq L}$;*
- ***updated:** Total outgoing value remains unchanged: $c^{new} = c^{old} \cancel{+ v^{new}_{note}}$;*

hold."

In the list above, the updated statements (with respect to the previous version) are denoted with '**updated**'. The new inputs $\overline{\mathrm{pk}}_{\mathsf{sndr}}$, $\overline{\mathrm{pk}}_{\mathsf{rcvr}}$, and $\overline{v}_{\mathsf{xfer}}$ are all public inputs to the arithmetic circuit, they will however not be included as plaintext in the transaction message. Instead they are included as a ciphertext after being encrypted using the SAVER encryption method $\mathrm{Enc}^{\mathsf{svr}}$. This method not only returns a ciphertext ct but also the updated zk-SNARK proof π[8]. From now on, we denote the ciphertext as $\mathrm{data}_{\mathsf{tx}} := \mathrm{ct}$. This field also gets added to tx, which is then signed to ensure integrity.

[7]These values are actually zero-strings with the same length as the value they replace.

[8]The proof π is altered in such a way that it works in the context of SAVER.

In all previous versions of the protocol the verifier of a transaction, would simply verify the correctness of the zero-knowledge proof π using $\mathrm{Verify}^{\mathsf{zkp}}$. However, the structure of π is slightly altered now. Moreover, the verifier should also check the validity of the encrypted $\mathrm{data}_{\mathsf{tx}}$ field. Therefore he should execute $\mathrm{VerifyEnc}^{\mathsf{svr}}$, with the ciphertext $\mathrm{data}_{\mathsf{tx}}$, proof π, public transaction values, and public key $\mathrm{pk}^{\mathsf{svr}}$ as input. If this algorithms accepts both the proof and the ciphertext, the transaction tx can be added to the blockchain.

When this is requested, the trusted authority can decrypt the contents of the $\mathrm{data}_{\mathsf{tx}}$ field of a questionable transaction tx on the blockchain. This decryption is performed using the authority's secret key $\mathrm{sk}^{\mathsf{svr}}$ and verification key $\mathrm{vk}^{\mathsf{svr}}$. These keys are used as input for the decryption algorithm $\mathrm{Dec}^{\mathsf{svr}}$, which returns the sender and receiver public key, as well as the transferred value in readable form. Next to this, the algorithm constructs a decryption proof ν which will come in handy in case of transferring the results to the relevant legal authority if this is required. The legal authority wants to be certain that the data it gets from the trusted authority actually corresponds to the transaction at hand. The trusted authority cannot give its secret key $\mathrm{sk}^{\mathsf{svr}}$ to the legal authority, as this key opens the ciphertexts of all transactions on the blockchain. Therefore, the trusted authority hands the plaintext message and the decryption proof ν to the legal authority. The legal authority can use $\mathrm{VerifyDec}^{\mathsf{svr}}$ on the provided values and the requested ciphertext to verify that the decryption process was indeed performed honestly, without learning the trusted authority's secret decryption key.

We note that a user can also choose to enclose transaction details for the trusted authority in a certain transaction, when the transaction would not surpass the limit. This would also entail that the transferred value does not count towards the limit. A user might want to do this for a transaction that is relatively large (when compared to the spend limit), and a lower level of anonymity is acceptable.

Version 12: Key sharing over N judges. In this version we purely discuss replacing the single trusted authority by a group of N judges. No changes to the algorithms or transactions are made.

Replacing the single trusted authority by a group of N judges, requires us to implement a protocol with which the judges can construct a SAVER key triplet together, with the special property that the secret decryption key is divided amongst the N judges. In other words, we need to find a way to execute $\mathrm{KeyGen}^{\mathsf{svr}}$ with N parties in such a way that only the N judges together can recover any information on encrypted ciphertexts. This means that any random values that are used to create the public and verification keys cannot be leaked to one another. We will assume that all judges are honest-but-curious in that they will honestly follow the steps of protocol, but at the same time try to find out as much information as they possibly can.

To begin, each judge $1 \leq j \leq N$ should generate its own shares (additive modulo p) $\{s_i^j\}_{i=1}^n, \{z_i^j\}_{i=1}^n, \{t_i^j\}_{i=0}^n, \rho^j \in_R \mathbb{Z}_p^*$ of the key randomness, i.e. $\{s_i\}_{i=1}^n \equiv \{\sum_j s_i^j\}_{i=1}^n \mod p$, etcetera. Given the common reference string, including g, h, g^δ, g^γ, and $g^{-\gamma}$, each judge j computes the values

$$\left(\left\{g^{\delta s_i^j}\right\}_{i=1}^n, \left\{g_i^{t_i^j}\right\}_{i=1}^n, \left\{h^{t_i^j}\right\}_{i=1}^n, g^{-\gamma\sum_{i=1}^n s_i^j}\right).$$

The judges then share all these values with one another, and individually compute

$$\left(\left\{g^{\delta s_i}\right\}_{i=1}^n, \left\{g_i^{t_i}\right\}_{i=1}^n, \left\{h^{t_i}\right\}_{i=1}^n, g^{-\gamma(1+\sum_{i=1}^n s_i)}\right)$$

by calculating the inner product of all vectors, and multiplying the last term with $g^{-\gamma}$.

Using the values $\left\{g^{\delta s_i}\right\}_{i=1}^n$ and g^δ each judge calculates and shares with the other judges the value $g^{\delta t_0^j}\prod_{i=1}^n g^{\delta s_i t_i^j}$. Each judge can combine all these values to $g^{\delta t_0}\prod_{i=1}^n g^{\delta s_i t_i}$ by a simple multiplication. Together with the previous vector of values the judges have now all calculated the public key $\mathrm{pk}^{\mathsf{svr}}$.

Subsequently, each judge j generates random values $\{r_i^j\} \in_R \mathbb{Z}_p$ that will be used as a blinding factor for some intermediate values. Each judge j then computes and shares the values $\left(h^{\rho^j}, \left\{h^{s_i^j r_i^j}\right\}_{i=1}^n\right)$. Given these values every judge j is able to compute h^ρ, and from this also the values $\left\{h^{\rho z_i^j}\right\}_{i=1}^n$. Moreover, j can compute $\left\{\left\{h^{s_i^{j'} r_i^{j'} z_i^j}\right\}_{i=1}^n\right\}_{j'=1}^N$ from $\left\{h^{s_i^j r_i^j}\right\}_{i=1}^n$. Each judge then shares these newly computed values. Finally, these new shares can be combined to compute $\{h^{\rho z_i}\}$.

Afterwards, each judge j can unmask all the shares of $\left\{\left\{h^{s_i^{j'} r_i^{j'} z_i^j}\right\}_{i=1}^n\right\}_{j'=1}^N$ for which $j' = j$, and share the resulting unmasked values $\left\{\left\{h^{s_i^{j'} z_i^j}\right\}_{i=1}^n\right\}_{j'=1}^N$. As a last step each judge can combine these shares to obtain $\{h^{s_i z_i}\}_{i=1}^n$. Note that the judges will not try to construct the secret key ρ from their individual shares ρ_j, instead they require a separate decryption procedure to decrypt a ciphertext without revealing their individual shares.

The judges can easily decrypt the ciphertext $\mathrm{ct} = (c_0, \ldots, c_n, \psi)$, using the public key $\mathrm{pk}^{\mathsf{svr}}$, the verification key $\mathrm{vk}^{\mathsf{svr}}$, and their respective shares ρ^j of the secret key. Using this, every judge computes $e(c_0, V_i)^{\rho^j}$ and shares this value with the other judges. Afterwards, all the judges can multiply these values to obtain $e(c_0, V_i)^\rho$ and compute the messages $(m_1, \ldots, m_n)$ and decryption proof ν in the regular way as is described in $\mathrm{Dec}^{\mathsf{svr}}$.

The rest of the protocol remains unaltered and works just in the way as presented in all previous versions.

4.9 Timelocks

The only requirement that is still to be included, is the *timelock* requirement. In this section we show how to add this feature to the current protocol. The timelock feature allows the sender of a note to prevent the receiver to spend that note for a certain amount of time. To preserve sender and receiver anonimity, presence of a timelock should not be visible to any other participant in the transaction system. We ensure this, by applying techniques similar to that in the transaction limit as sketched in Section 4.7.

Version 13: Timelock on an output note. Suppose that the sender of a transaction with transaction time t_{new}, wants to lock the output note for time t_δ. In other words, the sender creates a transaction at time t_{new} with output note $\mathrm{note}_{\text{new}}$ and makes the note only spendable at time $t_{\text{new}} + t_\delta$ or later.

We can achieve this by slightly adapting the Note Merkle tree. Namely, we can alter the leaf of a Note Merkle tree to have the value of the hash of the transaction time together with the commitment to the new note $\mathrm{cm}^{\text{new}}_{\text{note}}$. We call this hash function $\mathrm{CRH}^{\text{mem}}$ and define our hash tree analogously to the one presented in Figure 4.10.

The sender should also add the value t_δ to both the note commitment $\mathrm{cm}^{\text{new}}_{\text{note}}$ and the encrypted note data $\mathrm{data}_{\text{note}}$. Thus each note commitment is now constructed as $\mathrm{cm}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s_{\text{note}}}(pk_{\text{addr}} \| v_{\text{note}} \| t_\delta)$. Similarly, the encrypted note data is now computed as $\mathrm{data}_{\text{note}} := \mathrm{Enc}_{\mathrm{pk}_{\text{enc}}}(s_{\text{note}} \| v_{\text{note}} \| t_\delta \| \mathrm{info})$.

When t_δ has passed and the receiver of $\mathrm{note}_{\text{new}}$ wants to spend this note in a new transaction, he or she also needs to prove that the note is allowed to be spent. For this we add an additional statement to the zk-SNARK proof π of a transaction: *"I also know t_δ such that $t^{old}_{note} + t_\delta \leq t_{new}$.* In this statement $t^{\text{old}}_{\text{note}}$ is the time that $\mathrm{note}_{\text{new}}$ got added to the Note Merkle tree and t_{new} is the transaction time of the new transaction. Next to this additional statement, we also update the zk-SNARK statements for the note commitments to the new computation.

Together, these steps allow a sender to lock an output note for a certain amount of time. In the case that the sender of a transaction does not want to lock the output note for a certain amount of time, he or she simply sets $t_\delta := 0$. Furthermore, we force users and administrators to set $t_\delta := 0$ on creating a note as input for one of the conversion transactions. This is achieved by setting this value for t_δ in the computation of the note commitment in the zk-SNARK proofs for both conversions.

Chapter 5

Solution definition

In this chapter we formally define the solution as sketched in the previous chapter. We will first present and explain all the data structures in our scheme. In the subsequent section we will present the four arithmetic circuits that will be used to generate zk-SNARK proofs. Finally, we will define the algorithms that comprise our final payment scheme.

5.1 Data structures

The data structures form the backbone of the protocol and store all transaction details, users of the system, and account balances. Some data structures are also used as inputs and outputs of the algorithms that are defined in Section 5.3.

Blockchain ledger. The blockchain ledger B consists of several connected blocks, where each block contains the transactions that happened between the previous block and the next block. Thus at any time t the blockchain B_t contains a complete overview of all transactions that happened before time t. We further note that B is append-only, meaning that everything that was added to B at some point in time t is still part of B at a later point in time. Next to transactions, the blockchain also contains (references to) other data structures that are relevant for the state of the system. For more information on blockchains in general we refer the reader to Section A.1.

Public parameters. The public parameters pp form a list of values that is available to all participants in the system. This includes public keys of the admin, public parameters of encryption and signature schemes and proof and verifying keys for zk-SNARKs and SAVER. All these parameters are created before the initial launch of the system, and should be constructed by a trusted party/group, e.g. at least all the administrators and possibly some external trusted entities.

Credentials. Every user has a tuple of credentials consisting of four elements. A public address key $\mathrm{pk}_{\text{addr}}$ that is published openly, such that other users can send transactions to that address. A public encryption key pk_{enc} that is also published openly, such that the sender of a transaction can also transfer secrets to the receiver. The secret address key $\mathrm{sk}_{\text{addr}}$ is used to receive and send transactions using the notes and memory cell that belong to the corresponding public key as input. Similarly, the secret encryption key sk_{enc} is used to decrypt the secrets in a transaction that are encrypted under the corresponding public encryption key. There is also a commitment associated to the address credentials, namely $\mathrm{cm}_{\text{cred}}$. The set of all four keys, and related values, or user credentials, is denoted as cred.

Any administrator has the same set of credentials as the user, but also has a asymmetric key pair of signature keys: $\mathrm{sk}_{\text{asig}}$ and $\mathrm{pk}_{\text{asig}}$. All six keys together form the admin credentials, denoted as $\mathrm{cred}_{\text{adm}}$.

Note. A note, our digital representation of value, is connected to a couple of values. Obviously, a note has a monetary note value v_{note} and an owner who is represented with his or her public address key $\mathrm{pk}_{\text{addr}}$. A note also has a publication time t_{note}. Next to this, a note also has a commitment $\mathrm{cm}_{\text{note}}$ to all of these values. Finally, each note has an associated value that prevents double spending, namely its nullifier η. All these values, except for η, are encoded in a single note tuple, denoted as note.

Memory cell. A memory cell contains four values: (1) the owner's public address key $\mathrm{pk}_{\text{addr}}$; (2) current account balance v_{mem}; (3) publication time t_{mem}; and (4) the total amount of value that the owner has transferred, up to and including the transaction that introduced this memory cell, c. A memory cell also has a commitment to its contents: cm_{mem}. cm_{mem}, on its turn, has an associated nullifier μ that is used to invalidate an old memory cell every time a new memory cell is created. All these values, except for μ, are encoded in a single memory cell tuple, denoted as mem.

Transactions. There are five types of transactions, all denoted as tx, all of which contain a different collection of elements:

- *Regular transaction.* In short denoted as *Transfer*. A regular, note-to-note, transaction contains several elements. It contains the Merkle roots rt_* to the three Merkle trees used. Moreover, it contains the values k and κ that tie the signature key pair to the particular message by including it in the zero-knowledge proof π. There are also two nullifiers η and μ to respectively the input note and the old memory cell. Next to this, the transaction contains the creation time t^{new}, commitments to the newly created output note and the updated memory cell.

Finally, there is a zero-knowledge proof π that accompanies all of this for proving validity, as well as three encrypted data_* fields, two of which contain secrets to their respective newly created commitments. The third data field contains the encryption of either the SAVER transaction details or the null string.

- *Account creation.* In short denoted as *NewAccount*. This 'transaction' is originally sent from a user to an administrator and contains a credential commitment $\mathrm{cm}_{\mathsf{cred}}$, public address key $\mathrm{pk}_{\mathsf{addr}}$, and zero-knowledge proof π. The receiving admin publishes this transaction on the blockchain without the public address key and zero-knowledge proof and after signing it.
- *Account revocation.* In short denoted as *RvkAccount*. This transaction is published by an admin on the blockchain when a user's account is revoked. It contains the credential commitment $\mathrm{cm}_{\mathsf{cred}}$ belonging to the public address key that is revoked.
- *Conversion to note from fiat.* In short denoted as *ConvertTo*. This transaction consists of a commitment to a new note $\mathrm{cm}_{\mathsf{note}}$, encrypted secrets of the commitment $\mathrm{data}_{\mathsf{note}}$, and a zero-knowledge proof π for correctness of this transaction.
- *Conversion from note to fiat.* In short denoted as *ConvertFrom*. A conversion in the other direction, consists of a nullifier η to an old note commitment, the root of the current Note Merkle tree $\mathrm{rt}_{\mathsf{note}}$, and the public address key $\mathrm{pk}_{\mathsf{addr}}$ of the admin that performs the conversion. Finally, there is a zero-knowledge proof π that verifies the correctness of certain computations.

Merkle trees. Another frequently used data structure is the (binary) Merkle tree. In our proposed protocol there are three different Merkle trees: the Account Merkle tree, the Note Merkle tree, and the Memory Merkle tree. The roots rt_* of the most recent state of each of the Merkle trees are stored in every new block that is added to the blockchain ledger. Each Merkle tree is used to store a commitment to a different value, either credentials, a note, or a memory cell. Every time a message with a new commitment is added to the blockchain, this commitment also gets added to the corresponding Merkle tree. When adding a value to the Merkle tree as a leaf, this also leads to an update of its parent nodes and eventually the root rt_* of that Merkle tree. Proving membership of a value in a set using zk-SNARKs can be performed by proving presence of a commitment in a Merkle tree. To prove this presence we require knowledge of the position pos (leaf number) of and path path (list of hashes) to this commitment in the Merkle tree state that is represented by the Merkle root rt of the tree at that moment in time.

Nullifier lists. These lists are not a necessity for a working system, but they do help improve efficiency. We make use of two nullifier lists, one for note nullifiers and one for memory cell nullifiers. These lists simply store all the nullifiers that have already been included in a transaction, in order to make them easily findable when verifying a newly published transaction. These values could also be derived directly from the blockchain, however this would take more time since the nullifiers are not sorted efficiently on the ledger.

5.2 Arithmetic circuits

In order to construct a zk-SNARK proof for a statement using the *Groth16* proving scheme (and also many other schemes) the statement needs to be transformed into an arithmetic circuit. For our protocol we will define four of these circuits, all of which are used to prove validity of certain statements and public values in the protocol. In this section we shortly describe the purpose and usage of each circuit. In Chapter 6 the exact definition of the circuits will be set out.

- *CRED.* The circuit CRED is used by a user when he or she first wants to enter the system by sending a commitment to its credentials to the admin. In other words, the circuit is used for the proof in an *Account creation* transaction. These credentials should be accompanied by a proof of knowledge on the input values to the circuit CRED. To be precise, the arithmetic circuit includes the derivation of the public address key from the secret address key, as well as the computation of the credential commitment.

- *CTO.* In CTO only one calculation is included, that of the commitment to the newly created note. This circuit is used to proof correctness of the newly created note commitment by the admin in case of a conversion from fiat currency to an anonymous note.

- *CFROM.* When the admin converts a note to fiat currency a transaction is published on the blockchain. This transaction $\mathtt{tx}$ should also include proof that verifies correct computation of the nullifier that is published in this transaction. Moreover, it should verify that the transaction is performed by the admin that received the note for conversion.

- *XFER.* In the construction of a regular, note-to-note, transaction a lot of values need to be computed honestly. To be able to verify these computations, the XFER circuit should include all the calculations that are performed on the inputs to generate the public values shown in the transaction $\mathtt{tx}$. The computations to check include the construction of the commitments to the new note and updated memory cell, the computation of the nullifiers to the old commitment, and the balance equation. Moreover, the circuit is used to tie the signature key pair to the transaction by means of k and κ, and to check if the sender of the transaction has an acknowledged account in the Account Merkle tree. `XFER` also uses the SAVER plaintext as public input to the circuit. The correct value of the plaintext is ensured by the conditions in the arithmetic circuit. Finally, `XFER` should take into account the possible lack of an input note or old memory cell. This lack of an old memory cell can occur when the user has not yet interacted with the system apart from creating an account.

5.3 Algorithms

The functionality of the payment scheme as presented in Chapter 4 of this report is comprised of several algorithms. In this section we define those algorithms and sketch their intended purposes. We provide an overview of the inputs and outpus for each algorithm and present a rough sketch of the algorithm's steps. The implementation of these algorithms will be discussed in Chapter 6. The presented algorithm are either supposed to be used by administrators or users and possibly by both.

Setup
Input: security parameter λ
Output: public parameters pp; signature secret key $\mathrm{sk}_{\mathsf{asig}}$; address secret key $\mathrm{sk}_{\mathsf{addr}}$; encryption private key $\mathrm{sk}_{\mathsf{enc}}$

The algorithm *Setup* is run by a group that contains at least the (set of) admin(s) and possibly some other actors that together form a trusted party. The main goal of this algorithm is the initialisation of all proof and verification keys for the different arithmetic circuits. Next to this, the algorithm also requires the admin to construct a number of keys that are relevant for the function as admin in the protocol. The algorithm uses the *AddAdmin* algorithm to construct the keys. Finally, the setups for SAVER, the used encryption and the used signature schemes are performed at this point. After completion, the algorithm returns all relevant public parameters for the separate parts of the protocol.

AddAdmin
Input: security parameters λ; public parameters pp
Output: admin credentials $\mathrm{cred}_{\mathsf{adm}}$

This algorithm is called by every admin that participates in the *Setup*, but can also be called by a new admin at any later point in time. *AddAdmin* takes the public parameters of the system as input and constructs the credentials of an administrator. These credentials consist of an address, encryption, and signature key pair. The public keys of these pairs are added to the public parameters.

CreateAccount
Input: public parameters pp
Output: credentials cred

CreateAccount is called by any client that wishes to join the system. The algorithm creates all the credentials that the user needs, and computes a commitment to the address credentials. This commitment and public address key are sent to the admin together with a zero-knowledge proof π of the $\texttt{CRED}$ circuit. The admin is supposed to further handle these credentials, as described in the algorithm *AddAccount*. After the admin has finished *AddAccount*, the user is informed whether or not the credential commitment has been added to the Account Merkle tree. The algorithm returns the user's credentials that are needed for performing future transactions.

AddAccount
Input: public parameters pp; public credential values x; credential proof $\pi_{\texttt{CRED}}$; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: tx

On reception of a *CreateAccount* query, the admin performs CDD and verifies that the user does not have an account yet. If these checks are satisfied, the admin executes the rest of this algorithm, which verifies the construction of the received commitment, adds it to the Merkle tree and posts a message to the blockchain.

RevokeAccount
Input: credential commitment $\mathrm{cm}_{\text{cred}}$; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: tx; $\mathrm{note}^{\text{new}}$

On input of a commitment to a user's credentials, the algorithm *RevokeAccount* removes the commitment from the Account Merkle tree. Moreover, it publishes a signed revocation message to the blockchain that confirms that it was the admin who performed the revocation.

ConvertToNote
Input: public parameters pp; conversion value $v_{\text{note}}^{\text{new}}$; public destination address $\mathrm{pk}_{\text{addr}}^{\text{new}}$; destination encryption key $\mathrm{pk}_{\text{enc}}^{\text{new}}$; extra info info; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: transaction tx

On getting paid with value $v^{\text{new}}_{\text{note}}$ in fiat currency by a user with public address key $\mathrm{pk}^{\text{new}}_{\text{addr}}$, the admin calls the algorithm *ConvertToNote*, to add a new note to the system of the same value destined for public address key $\mathrm{pk}^{\text{new}}_{\text{addr}}$. The algorithm calculates a commitment to the new note, encrypts the necessary secrets under the user's public encryption key, and constructs a zero-knowledge proof for the circuit `CTO`. Finally, the admin signs the transaction tx and publishes it on the blockchain. Subsequently, the new note commitment gets added to the Note Merkle tree which completes the conversion in such a way that the user can now spend a note of value $v^{\text{new}}_{\text{note}}$ anonymously.

ConvertFromNote

Input: public parameters pp; input note $\mathrm{note}^{\text{old}}$; Note Merkle root $\mathrm{rt}_{\text{note}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: transaction tx

When the admin receives a note from a user with public address key $\mathrm{pk}_{\text{addr}}$[1], the algorithm *ConvertFromNote* makes the note unspendable by publishing the nullifier η to the note. The admin publishes the transaction containing this nullifier on the blockchain, with a signature under the admin's secret signing key. This transaction also includes a zero-knowledge proof for the circuit `CFROM`. After having done all this, the admin transfers the value of the note v_{note} in fiat currency to the user with public address key $\mathrm{pk}_{\text{addr}}$ to complete the conversion.

CreateTransaction

Input: public parameters pp; credentials cred; Note Merkle root $\mathrm{rt}_{\text{note}}$; Memory Merkle root rt_{mem}; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; old note $\mathrm{note}^{\text{old}}$; previous memory cell $\mathrm{mem}^{\text{old}}$; ceiling memory cell $\mathrm{mem}^{\text{old}}$; new note value $v^{\text{new}}_{\text{note}}$; new public address $\mathrm{pk}^{\text{new}}_{\text{addr}}$; new public encryption key $\mathrm{pk}^{\text{new}}_{\text{enc}}$; extra info info; new block time t^{new}; boolean value for including SAVER encrypted values b_{saver}
Output: transaction tx

When a user wants to transfer value to another user in an anonymous way he or she uses the algorithm *CreateTransaction*. As input, the sender provides the public address key of the receiver, the user's own credentials and memory cell, an optional input note and some optional extra info info that the user wants to include in the transaction. The algorithm then requires the user to construct an output note, produce an updated memory cell, calculate nullifiers to the input commitments, sample a one-time signature key pair, and compute a zk-SNARK proof that validates all components, ties them together, and verifies that the sender has a valid account. This zk-SNARK proof is also accompanied by a SAVER encryption of either the zero string, or the transaction details depending on the value of b_{saver}. The sender then

[1]The address is included in the $\mathrm{data}_{\text{note}}$ field

signs the message using the one-time signature key pair and publishes the signed message (anonymously) on the blockchain to complete the transaction.

VerifyTransaction
Input: public parameters pp; transaction tx; current blockchain B
Output: boolean value for correctness b

The admins of the blockchain should validate transactions before they get accepted and added to a new block on the blockchain, by entering every transaction as input to *VerifyTransaction*. Moreover, any user can also verify the admins in doing this by checking accepted transactions with this algorithm. *VerifyTransaction* verifies the provided zero-knowledge proof along with the SAVER ciphertext, checks if the public inputs that are used actually exist and are recent and verifies that the transaction has not been tampered with. Finally, the algorithm verifies that the produced nullifiers do not yet appear on their respective nullifier lists. If all tests pass successfully, the transaction is valid and gets added to the blockchain.

ReceiveTransaction
Input: public parameters pp; new transaction tx; credentials cred; current blockchain B
Output: received note $\mathrm{note}^{\text{new}}$

ReceiveTransaction is called by users to check if new transactions on the blockchain were sent to their address. If this is the case the algorithm outputs all relevant details of the received note. Firstly, the algorithm calls *VerifyTransaction* to confirm the validity of the transaction. Secondly, the algorithm tries to decrypt the $\mathrm{data}_{\text{note}}$ field of the transaction. If the decryption output makes sense, the algorithm continues and checks if the the commitment to the note can be reconstructed with the decrypted secrets. If this is possible the user has received a valid transfer and can store the note for later use. If this was not the case, either the transaction was not meant for this user or the secrets have been invalidly encrypted, making the created note unusable.[2]

[2]We note that this is not a security risk, since doing this only negatively affects the sender. Therefore, the sender has more than sufficient incentive to encrypt the secrets honestly.

Chapter 6

Solution construction

In this chapter we present the detailed construction of our payment scheme. We begin with a concrete description of the cryptographic building blocks that will be used. In the next section we present a precise list of zk-SNARK statements. This list is followed by the construction of the algorithm and we end with a discussion on the completeness and security of our construction.

6.1 Building blocks

Before we continue to the construction of the earlier defined algorithms, we need to introduce some of the cryptographic building blocks that will be used by these algorithms. For more details on these building blocks we refer the reader to Chapter 2. Throughout the definitions below, λ denotes the desired security parameter.

Pseudorandom functions. We use a keyed family of pseudo random functions $\{\mathrm{PRF}_k(s) : \{0,1\}^n \to \{0,1\}^{O(\lambda)}\}_k$, where k is the key, s the seed or input, and n the input size. From any keyed pseudorandom function PRF_k we derive four versions that are used directly in different parts of the algorithms: $\mathrm{PRF}_x^{\mathsf{addr}}(s) := \mathrm{PRF}_x(00\| s)$, $\mathrm{PRF}_x^{\eta}(s) := \mathrm{PRF}_x(01\|s)$, $\mathrm{PRF}_x^{\mu}(s) := \mathrm{PRF}_x(10\|s)$, and $\mathrm{PRF}_x^{\kappa}(s) := \mathrm{PRF}_x(11\| s)$. As mentioned before, we require the family of pseudorandom functions to be collision-resistant with respect to (x, s).

Collision-resistant hash function. We use a collision-resistant hash function (CRH) function that takes as input a message of arbitrary length and outputs a fixed length value: $\mathrm{CRH} : \{0,1\}^* \to \{0,1\}^{O(\lambda)}$. We use four instantiations of CRH. The standard CRH is used to compute the parent of two Merkle nodes in a Merkle tree. $\mathrm{CRH}^{\mathsf{mem}}$ is used to compute the leaf of the Memory Merkle tree from a memory commitment $\mathrm{cm}_{\mathsf{mem}}$ and a time t. Analogously, $\mathrm{CRH}^{\mathsf{note}}$ is used to compute the leaf of the Note Merkle tree from a note commitment $\mathrm{cm}_{\text{note}}$ and a time t. Finally, we define $\mathrm{CRH}^{\text{sig}}$ to obtain a fixed length bit string from the public signature key, so that it can be used as input to a function with fixed input length.

Key derivation function. A key derivation function (KDF) is a function that takes as input the public parameters of an encryption scheme and a secret input. The KDF outputs a public and private key pair that are derived from the secret input and satisfy the desired properties of the encryption scheme without revealing any information about the secret input. In our case it is a function $\mathrm{KDF}^{\text{enc}}(\mathrm{pp}_{\text{enc}}, x) \to (\mathrm{pk}_{\text{enc}}, \mathrm{sk}_{\text{enc}})$, where pp is the list of public parameters and x the secret input. The function output is the public and private encryption key pair.

Commitments. We choose to use a commitment scheme that is computationally binding and statistically hiding, since transactions will be visible for a very long time on the blockchain, whereas forging a commitment would cost more than the benefit.[1] A commitment cm is computed as: $\mathrm{cm} = \mathrm{COMM}_r(x) : \{0,1\}^n \to \{0,1\}^{O(\lambda)}$, where r is the commitment trapdoor, x the input and n the input size. The protocol makes use of several different instantiations, which will be discussed in Chapter 7. In accordance with Section 2.1, the different versions are denoted by their respective superscripts.

Digital signature schemes. In our protocol we require two different digital signature schemes. One scheme that is strongly-unforgable, and one that is only one-time strongly-unforgable. Both security definitions should hold against chosen-message attacks, i.e. $\mathrm{SUF\text{-}CMA}$ and $\mathrm{SUF\text{-}1CMA}$. The first of these schemes is used by the admin, who needs to be able to sign many different messages with the same key pair. The second scheme is used by users to sign a transaction, where a signature key pair is generated for only one transaction, hence we only need one-time security here. Both schemes, denoted with a superscript *asig* respectively *sig*, consist of the following algorithms:

- $\mathrm{Setup}^{\text{(a)sig}}(1^\lambda) \to \mathrm{pp}_{\text{(a)sig}}$ constructs the public parameters of the signature scheme given the security parameter.
- $\mathrm{KeyGen}^{\text{(a)sig}}(\mathrm{pp}_{\text{(a)sig}}) \to (\mathrm{pk}_{\text{(a)sig}}, \mathrm{sk}_{\text{(a)sig}})$ generates a public and secret signature key for the signature scheme given its public parameters.
- $\mathrm{Sign}^{\text{(a)sig}}_{\mathrm{sk}_{\text{(a)sig}}}(m) \to \sigma$ computes a signature σ on message m under the secret key $\mathrm{sk}_{\text{(a)sig}}$.

[1]It is not possible for a scheme to be both information-theoretically binding and information-theoretically (or statistically) hiding. [36]

- $\mathrm{Verify}^{\text{(a)sig}}_{\mathrm{pk}_{\text{(a)sig}}}(m, \sigma) \rightarrow b$ verifies if σ is a valid signature on m given the public key $\mathrm{pk}_{\text{(a)sig}}$. If the signature is indeed valid $b = \texttt{true}$, else $b = \texttt{false}$.

Public key encryption. Another important cryptographic building block is public key encryption, which is used to send commitment secrets to the receiver of a transaction. To keep these secrets secret, they need to be encrypted under the receiver's public encryption key. In order to prevent that an encrypted message can be linked to the key it is encrypted under, we require a scheme that is key private. To be precise, we require an asymmetric encryption scheme that is not only $\mathrm{IND\text{-}CCA}$, but also $\mathrm{IK\text{-}CCA}$. We require an encryption scheme that has at least the following functionality:

- $\mathrm{Setup}^{\text{enc}}(1^\lambda) \rightarrow \mathrm{pp}_{\text{enc}}$ construct the public parameters of the encryption scheme given the security parameter.

- $\mathrm{Enc}_{\mathrm{pk}_{\text{enc}}}(m) \rightarrow c$ computes the ciphertext c from the message m given encryption key pk_{enc}.

- $\mathrm{Dec}_{\mathrm{sk}_{\text{enc}}}(c) \rightarrow m$ computes the message m (or $\perp$ if decryption is impossible) from the ciphertext c given the decryption key sk_{enc}.

We note that, in order to generate the encryption key pair, the protocol uses a key derivation function (KDF) as described above.

zk-SNARK scheme. As discussed in Section 2.2, we make use of a zk-SNARK scheme known as *Groth16*. This zk-SNARK scheme contains three algorithms:

- $\mathrm{Setup}^{\text{zkp}}(1^\lambda, C) \rightarrow (\mathrm{pk}, \mathrm{vk})$ is the setup function that generates the proving pk and verifying vk key, given the security parameter and a circuit C over which proofs will be generated.

- $\mathrm{Prove}^{\text{zkp}}(\mathrm{pk}, x, a) \rightarrow \pi$ constructs a zk-SNARK proof π given a proving key pk that encodes the circuit over which the proof is constructed. The function also requires the right amount of public inputs x and auxiliary inputs a.

- $\mathrm{Verify}^{\text{zkp}}(\mathrm{vk}, \pi, x) \rightarrow b$ verifies the proof π over the circuit that is encoded in the verifying key vk, given public inputs x. The result is $b = \texttt{true}$ if verification succeeds, otherwise $b = \texttt{false}$.

Verifiable encryption scheme. In Section 2.3 we discuss the verifiable encryption that is used in our protocol: *SAVER*. In this section we give an exact definition of the relevant algorithms. For completeness we present the relevant algorithms below[2]:

- $\mathrm{Setup}^{\mathsf{svr}}(1^\lambda, C) \rightarrow (\mathrm{pk}, \mathrm{vk})$ is the setup function the generates the proving pk and verifying vk key, given the security parameter and a circuit C over which proofs will be generated. This algorithm calls $\mathrm{Setup}^{\mathsf{zkp}}$ with the same arguments and adds some extra values that are required for SAVER. In the case of multiple administrators we use our multi-party adaptation on this function as presented in Section 4.8.
- $\mathrm{KeyGen}^{\mathsf{svr}}(\mathrm{pk}, \mathrm{vk}) \rightarrow (\mathrm{pk}_{\mathsf{svr}}, \mathrm{sk}_{\mathsf{svr}}, \mathrm{vk}_{\mathsf{svr}})$ constructs the public $\mathrm{pk}_{\mathsf{svr}}$, secret $\mathrm{sk}_{\mathsf{svr}}$, and $\mathrm{vk}_{\mathsf{svr}}$ verifying key for SAVER. These keys are generated using the zk-SNARK proving pk and verifying vk key.
- $\mathrm{Enc}^{\mathsf{svr}}(\mathrm{pk}, \mathrm{pk}_{\mathsf{svr}}, m, x, a) \rightarrow (\pi, c)$ computes the SAVER adapted zk-SNARK proof π and SAVER ciphertext c using the plaintext message m, other public inputs x, and auxiliary variables a. The algorithm also requires the SAVER public key $\mathrm{pk}_{\mathsf{svr}}$ and zk-SNARK prover key pk. This prover key is used as input for $\mathrm{Prove}^{\mathsf{zkp}}$ as well.
- $\mathrm{VerifyEnc}^{\mathsf{svr}}(\mathrm{vk}, \mathrm{pk}_{\mathsf{svr}}, \pi, c, x) \rightarrow b$ verifies the proof π along the SAVER ciphertext c that are encoded in the verifying key vk, given SAVER public key $\mathrm{pk}_{\mathsf{svr}}$ and public inputs x. The result is $b = \texttt{true}$ if verification succeeds, otherwise $b = \texttt{false}$.

The specific instantiations of these building blocks, apart from the zk-SNARK and SAVER scheme, will be discussed in Chapter 7. That Chapter focuses on implementation details, and discusses which schemes have the best efficiency-security trade-off. In that same chapter we also provide more details on input and output size of the specific schemes.

6.2 zk-SNARK statements

In this section we construct the statements that are proved in the four circuits that were defined in Section 5.2. The intuition for the statements is presented in Section 4. A statement has two types of inputs: public and auxiliary. The public inputs are contained in the public transaction tx and are thus visible to the verifier. The auxiliary inputs are only known by the prover, i.e. the user that constructs the proof, and need

[2]We do not present the decryption *Dec* and decryption verification *VerifyDec* algorithm here, since these are not explicitly used in any of the algorithms in our payment scheme.

to stay secret. At the same time, the prover wants to convince the verifier that he or she knows these secret auxiliary inputs.

Additionally, the inputs should satisfy the relations as defined in the statements. Our description of the statements includes these elements as follows: *"Given [public inputs], the prover knows [auxiliary inputs], such that the following statement(s) hold(s): [relations]."*. For more information on the used notation and on zk-SNARK proofs we refer the reader to Chapter 2. Furthermore we note that T and L are respectively time and value limits that are hard coded in the arithmetic circuit, according to Section 4.7.

`CRED`

Given $(\mathrm{cm}_{\text{cred}}, \mathrm{pk}_{\text{addr}})$, the prover knows $(\mathrm{sk}_{\text{addr}}, \mathrm{s}_{\text{cred}})$, such that the following statements hold:

- Correct derivation of public address key from secret address key: $\mathrm{pk}_{\text{addr}} = \mathrm{PRF}^{\text{addr}}_{\mathrm{sk}_{\text{addr}}}(0)$;
- Correct computation of credential commitment: $\mathrm{cm}_{\text{cred}} := \mathrm{Comm}^{\text{cred}}_{s_{\text{cred}}}(\mathrm{pk}_{\text{addr}} \| \mathrm{sk}_{\text{addr}})$.

`CTO`

Given $(\mathrm{cm}^{\text{new}}_{\text{note}})$, the prover knows $(s^{\text{new}}_{\text{note}}, v^{\text{new}}_{\text{note}}, \mathrm{pk}^{\text{new}}_{\text{addr}})$, such that the following statement holds:

- Correct computation of output note commitment: $\mathrm{cm}^{\text{new}}_{\text{note}} = \mathrm{Comm}^{\text{note}}_{s^{\text{new}}_{\text{note}}}(\mathrm{pk}^{\text{new}}_{\text{addr}} \| v^{\text{new}}_{\text{note}} \| 0)$.

`CFROM`

Given $(\mathrm{rt}_{\text{note}}, \eta, \mathrm{pk}_{\text{addr}})$, the prover knows $(\mathrm{sk}_{\text{addr}}, s^{\text{old}}_{\text{note}}, v^{\text{old}}_{\text{note}}, t^{\text{old}}_{\text{note}}, \mathrm{cm}^{\text{old}}_{\text{note}}, \mathrm{path}_{\text{note}}, \mathrm{pos}_{\text{note}})$, such that the following statements hold:

- Correct derivation of public address key from secret address key: $\mathrm{pk}_{\text{addr}} = \mathrm{PRF}^{\text{addr}}_{\mathrm{sk}_{\text{addr}}}(0)$;
- $(\mathrm{pos}_{\text{note}}, \mathrm{path}_{\text{note}})$ is the valid Merkle Tree path from $CRH^{\text{note}}(\mathrm{cm}^{\text{old}}_{\text{note}}, t^{\text{old}}_{\text{note}})$ to $\mathrm{rt}_{\text{note}}$;
- Correct computation of input note commitment: $\mathrm{cm}^{\text{old}}_{\text{note}} = \mathrm{Comm}^{\text{note}}_{s^{\text{old}}_{\text{note}}}(\mathrm{pk}_{\text{addr}} \| v^{\text{old}}_{\text{note}} \| 0)$;
- Correct computation of input note nullifier: $\eta = \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{note}})$.

`XFER`

Given $(\overline{\mathrm{pk}}_{\text{sndr}}, \overline{\mathrm{pk}}_{\text{rcvr}}, \overline{v}_{\text{xfer}}, \mathrm{rt}_{\text{mem}}, \mathrm{rt}_{\text{note}}, \mathrm{rt}_{\text{cred}}, k, \kappa, \eta, \mu, \mathrm{cm}^{\text{new}}_{\text{note}}, \mathrm{cm}^{\text{new}}_{\text{mem}}, t^{\text{new}})$, the prover knows $(\mathrm{rt}^{\text{real}}_{\text{mem}}, \mathrm{rt}^{\text{int}}_{\text{mem}}, \mathrm{rt}^{\text{int}}_{\text{note}}, \eta^{\text{int}}, \mathrm{pk}_{\text{addr}}, \mathrm{sk}_{\text{addr}}, s_{\text{cred}}, \mathrm{cm}_{\text{cred}}, v^{\text{old}}_{\text{note}}, s^{\text{old}}_{\text{note}}, t^{\text{old}}_{\delta}, \mathrm{cm}^{\text{old}}_{\text{note}}, t^{\text{old}}_{\text{note}}, v^{\text{old}}_{\text{mem}}, s^{\text{old}}_{\text{mem}}, c^{\text{old}}, \mathrm{cm}^{\text{old}}_{\text{mem}}, t^{\text{old}}_{\text{mem}}, v^{\text{ceil}}_{\text{mem}}, s^{\text{ceil}}_{\text{mem}}, c^{\text{ceil}}, \mathrm{cm}^{\text{ceil}}_{\text{mem}}, t^{\text{ceil}}_{\text{mem}}, \mathrm{path}_{\text{cred}}, \mathrm{path}_{\text{note}}, \mathrm{path}_{\text{mem}}, \mathrm{path}_{\text{ceil}}, \mathrm{pos}_{\text{note}}, \mathrm{pos}_{\text{mem}}, \mathrm{pos}_{\text{cred}}, \mathrm{pos}_{\text{ceil}}, s^{\text{new}}_{\text{note}}, s^{\text{new}}_{\text{mem}}, v^{\text{new}}_{\text{note}}, v^{\text{new}}_{\text{mem}}, c^{\text{new}}, \mathrm{pk}^{\text{new}}_{\text{addr}}, b_{\text{note}}, b_{\text{mem}}, b_{\text{saver}}, t_{\delta})$, such that the following statements hold:

- Correct computation of credential commitment: $\mathrm{cm}_{\mathsf{cred}} := \mathrm{Comm}^{\mathsf{cred}}_{s_{\mathsf{cred}}}(\mathrm{pk}_{\mathsf{addr}} \| \mathrm{sk}_{\mathsf{addr}})$;
- Correct computation of input note commitment: $\mathrm{cm}^{\mathsf{old}}_{\mathsf{note}} = \mathrm{Comm}^{\mathsf{note}}_{s^{\mathsf{old}}_{\mathsf{note}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{old}}_{\mathsf{note}} \| t^{\mathsf{old}}_{\delta})$;
- Correct computation of previous memory cell commitment: $\mathrm{cm}^{\mathsf{old}}_{\mathsf{mem}} := \mathrm{Comm}^{\mathsf{mem}}_{s^{\mathsf{old}}_{\mathsf{mem}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{old}}_{\mathsf{mem}} \| c^{\mathsf{old}})$;
- Correct computation of ceiling memory cell commitment: $\mathrm{cm}^{\mathsf{ceil}}_{\mathsf{mem}} := \mathrm{Comm}^{\mathsf{mem}}_{s^{\mathsf{ceil}}_{\mathsf{mem}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{ceil}}_{\mathsf{mem}} \| c^{\mathsf{ceil}})$;
- $(\mathrm{pos}_{\mathsf{cred}}, \mathrm{path}_{\mathsf{cred}})$ is the valid Merkle Tree path from $\mathrm{cm}_{\mathsf{cred}}$ to $\mathrm{rt}_{\mathsf{cred}}$;
- $(\mathrm{pos}_{\mathsf{note}}, \mathrm{path}_{\mathsf{note}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathsf{note}}(\mathrm{cm}^{\mathsf{old}}_{\mathsf{note}}, t^{\mathsf{old}}_{\mathsf{note}})$ to $\mathrm{rt}^{\mathsf{int}}_{\mathsf{note}}$;
- $(\mathrm{pos}_{\mathsf{mem}}, \mathrm{path}_{\mathsf{mem}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathsf{mem}}(\mathrm{cm}^{\mathsf{old}}_{\mathsf{mem}}, t^{\mathsf{old}}_{\mathsf{mem}})$ to $\mathrm{rt}^{\mathsf{int}}_{\mathsf{mem}}$;
- $(\mathrm{pos}_{\mathsf{ceil}}, \mathrm{path}_{\mathsf{ceil}})$ is the valid Merkle Tree path from $\mathrm{CRH}^{\mathsf{mem}}(\mathrm{cm}^{\mathsf{ceil}}_{\mathsf{mem}}, t^{\mathsf{ceil}}_{\mathsf{mem}})$ to $\mathrm{rt}^{\mathsf{int}}_{\mathsf{mem}}$ `or` $c^{\mathsf{ceil}} = 0$;
- Correct computation of previous memory cell nullifier: $\mu = \mathrm{PRF}^{\mu}_{\mathrm{sk}_{\mathsf{addr}}}(\mathrm{pos}_{\mathsf{mem}})$;
- Correct computation of input note nullifier: $\eta^{\mathsf{int}} = \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\mathsf{addr}}}(\mathrm{pos}_{\mathsf{note}})$;
- If there is no previous memory cell set the related values accordingly: `if` $b_{\mathsf{mem}} =$ `false, then` $(v^{\mathsf{old}}_{\mathsf{mem}} = 0$ `and` $\mathrm{pos}_{\mathsf{mem}} = -1$ `and` $\mathrm{rt}_{\mathsf{mem}} = \mathrm{rt}^{\mathsf{real}}_{\mathsf{mem}}$ `and` $c^{\mathsf{ceil}} = 0$ `and` $c^{\mathsf{old}} = 0)$, `else if` $b_{\mathsf{mem}} =$ `true, then` $\mathrm{rt}_{\mathsf{mem}} = \mathrm{rt}^{\mathsf{int}}_{\mathsf{mem}}$;
- If there is no input note set the related values accordingly: `if` $b_{\mathsf{note}} =$ `false, then` $(v^{\mathsf{old}}_{\mathsf{note}} = 0$ `and` $\eta = 0$ `and` $\mathrm{rt}_{\mathsf{note}} = 0)$, `else if` $b_{\mathsf{note}} =$ `true, then` $(\eta = \eta^{\mathsf{int}}$ `and` $\mathrm{rt}_{\mathsf{note}} = \mathrm{rt}^{\mathsf{int}}_{\mathsf{note}})$;
- Correct calculation of new account balance in memory cell: $v^{\mathsf{new}}_{\mathsf{mem}} = v^{\mathsf{old}}_{\mathsf{note}} + v^{\mathsf{old}}_{\mathsf{mem}} - v^{\mathsf{new}}_{\mathsf{note}}$;
- The new total outgoing value is correct: $c^{\mathsf{new}} = c^{\mathsf{old}} + v^{\mathsf{new}}_{\mathsf{note}} \cdot (1 - b_{\mathsf{saver}})$;
- The output note value and new account balance lie in the correct range: $0 \le v^{\mathsf{new}}_{\mathsf{mem}}, v^{\mathsf{new}}_{\mathsf{note}} \le v_{\mathsf{max}}$;
- The transaction is not an empty transaction: $0 < v^{\mathsf{new}}_{\mathsf{note}} + v^{\mathsf{old}}_{\mathsf{note}}$.
- The ceiling memory cell is old enough: $t^{\mathsf{ceil}}_{\mathsf{mem}} < t^{\mathsf{new}} - T$;
- The spend limit is not surpassed: $c^{\mathsf{new}} - c^{\mathsf{ceil}} \le L$ `or` $b_{\mathsf{saver}} = \mathtt{true}$;
- The note is unlocked: $t^{\mathsf{old}}_{\mathsf{note}} + t^{\mathsf{old}}_{\delta} \le t^{\mathsf{new}}$.
- Correct computation of output note commitment: $\mathrm{cm}^{\mathsf{new}}_{\mathsf{note}} = \mathrm{Comm}^{\mathsf{note}}_{s^{\mathsf{new}}_{\mathsf{note}}}(\mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}} \| v^{\mathsf{new}}_{\mathsf{note}} \| t_{\delta})$;
- Correct computation of new memory cell commitment: $\mathrm{cm}^{\mathsf{new}}_{\mathsf{mem}} := \mathrm{Comm}^{\mathsf{mem}}_{s^{\mathsf{new}}_{\mathsf{mem}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{new}}_{\mathsf{mem}} \| c^{\mathsf{new}})$;
- The public signature key is tied to this message with the sender's secret address key by means of κ: $\kappa = \mathrm{PRF}^{\kappa}_{\mathrm{sk}_{\mathsf{addr}}}(k)$;
- The saver encrypted values are defined correctly if $b_{\mathsf{saver}} = \mathtt{true}$: $\overline{\mathrm{pk}}_{\mathsf{sndr}} = \mathrm{pk}_{\mathsf{addr}} \cdot b_{\mathsf{saver}}$, $\overline{\mathrm{pk}}_{\mathsf{rcvr}} = \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}} \cdot b_{\mathsf{saver}}$, and $\overline{v}_{\mathsf{xfer}} = v^{\mathsf{new}}_{\mathsf{note}} \cdot b_{\mathsf{saver}}$;
- $b_{\mathsf{note}}, b_{\mathsf{saver}}, b_{\mathsf{mem}} \in \{0, 1\}$.

6.3 Algorithms

After having constructed the arithmetic circuits and discussed the necessary cryptographic building blocks, we continue with the construction of the algorithms. Below, all the steps that an algorithm takes to fulfil its desired functionality are given in complete detail. Each steps either compute a certain value, parses a piece of data, or incorporates functionalities of a certain cryptographic building block.

Setup
Input: security parameter λ
Output: public parameters pp

1. Construct the arithmetic circuits $C_{\mathtt{XFER}}$,$C_{\mathtt{CRED}}$, $C_{\mathtt{CTO}}$, $C_{\mathtt{CFROM}}$ at security level λ.
2. Generate the proving and verification key pairs for all four circuits $(\mathrm{pk}_{\mathtt{XFER}}, \mathrm{vk}_{\mathtt{XFER}}) := \mathrm{Setup}^{\mathsf{svr}}(1^\lambda, C_{\mathtt{XFER}}))$, $(\mathrm{pk}_{\mathtt{CRED}}, \mathrm{vk}_{\mathtt{CRED}}) := \mathrm{Setup}^{\mathsf{zkp}}(1^\lambda, C_{\mathtt{CRED}}))$, $(\mathrm{pk}_{\mathtt{CTO}}, \mathrm{vk}_{\mathtt{CTO}}) := \mathrm{Setup}^{\mathsf{zkp}}(1^\lambda, C_{\mathtt{CTO}}))$, and $(\mathrm{pk}_{\mathtt{CFROM}}, \mathrm{vk}_{\mathtt{CFROM}}) := \mathrm{Setup}^{\mathsf{zkp}}(1^\lambda, C_{\mathtt{CFROM}}))$.
3. The admin(s) together compute $(\mathrm{pk}_{\mathsf{svr}}, \mathrm{sk}_{\mathsf{svr}}, \mathrm{vk}_{\mathsf{svr}}) := \mathrm{KeyGen}^{\mathrm{svr}}(\mathrm{pk}_{\mathtt{XFER}}, \mathrm{vk}_{\mathtt{XFER}})$ and store their individual shares of $\mathrm{sk}_{\mathsf{svr}}$.
4. Create the public parameters for the encryption scheme $\mathrm{pp}_{\mathsf{enc}} := \mathrm{Setup}^{\mathsf{enc}}(1^\lambda)$.
5. Create the public parameters for the signature scheme $\mathrm{pp}_{\mathsf{sig}} := \mathrm{Setup}^{\mathsf{sig}}(1^\lambda)$.
6. Create the public parameters for the admin signature scheme $\mathrm{pp}_{\mathsf{asig}} := \mathrm{Setup}^{\mathsf{asig}}(1^\lambda)$.
7. Set $\mathrm{pp} := (\mathrm{pk}_{\mathtt{XFER}}, \mathrm{vk}_{\mathtt{XFER}}, \mathrm{pk}_{\mathtt{CRED}}, \mathrm{vk}_{\mathtt{CRED}}, \mathrm{pk}_{\mathtt{CTO}}, \mathrm{vk}_{\mathtt{CTO}}, \mathrm{pk}_{\mathtt{CFROM}}, \mathrm{vk}_{\mathtt{CFROM}}, \mathrm{pp}_{\mathsf{enc}}, \mathrm{pp}_{\mathsf{sig}}, \mathrm{pp}_{\mathsf{asig}}, \mathrm{pk}_{\mathsf{svr}}, \mathrm{vk}_{\mathsf{svr}})$.
8. Each admin calls **AddAdmin**$(\lambda.\mathrm{pp})$
9. Publish pp.

AddAdmin
Input: security parameters λ; public parameters pp
Output: admin credentials $\mathrm{cred}_{\mathsf{adm}}$

1. Parse pp as $(\mathrm{pp}_{\mathsf{enc}}, \mathrm{pp}_{\mathsf{asig}}, *)$
2. Generate the public and secret key for the admin signature scheme as $(\mathrm{pk}_{\mathsf{asig}}, \mathrm{sk}_{\mathsf{asig}}) := \mathrm{KeyGen}^{\mathsf{asig}}(\mathrm{pp}_{\mathsf{asig}})$.
3. Randomly sample an address secret key $\mathrm{sk}_{\mathsf{addr}}$ that is also a seed to $\mathrm{PRF}^{\mathsf{addr}}$.
4. Define the address public key as $\mathrm{pk}_{\mathsf{addr}} := \mathrm{PRF}^{\mathsf{addr}}_{\mathrm{sk}_{\mathsf{addr}}}(0)$.
5. Generate the encryption public-private key pair $(\mathrm{pk}_{\mathsf{enc}}, \mathrm{sk}_{\mathsf{enc}}) := \mathrm{KDF}^{\mathsf{enc}}(\mathrm{pp}_{\mathsf{enc}}, \mathrm{sk}_{\mathsf{addr}})$.
6. Add $(\mathrm{pk}_{\mathsf{asig}}\mathrm{pk}_{\mathsf{addr}}, \mathrm{pk}_{\mathsf{enc}})$ to pp.
7. Set $\mathrm{cred}_{\mathsf{adm}} = (\mathrm{pk}_{\mathsf{asig}}, \mathrm{sk}_{\mathsf{asig}}, \mathrm{sk}_{\mathsf{addr}}, \mathrm{pk}_{\mathsf{addr}}, \mathrm{pk}_{\mathsf{enc}}, \mathrm{sk}_{\mathsf{enc}})$.

CreateAccount
Input: public parameters pp
Output: credentials cred

1. Parse pp as $(\mathrm{pk}_{\mathtt{CRED}}, \mathrm{pp}_{\mathsf{enc}}, *)$.
2. Randomly sample an address secret key $\mathrm{sk}_{\mathsf{addr}}$ that is also a seed to $\mathrm{PRF}^{\mathsf{addr}}$.
3. Generate the encryption public and private key pair $(\mathrm{pk}_{\mathsf{enc}}, \mathrm{sk}_{\mathsf{enc}}) := \mathrm{KDF}^{\mathsf{enc}}(\mathrm{pp}_{\mathsf{enc}}, \mathrm{sk}_{\mathsf{addr}})$.

4. Define the address public key $\mathrm{pk}_{\text{addr}} := \mathrm{PRF}^{\text{addr}}_{\mathrm{sk}_{\text{addr}}}(0)$.
5. Randomly sample a commitment trapdoor s_{cred}.
6. Calculate the hiding commitment $\mathrm{cm}_{\text{cred}}$ as $\mathrm{cm}_{\text{cred}} := \mathrm{Comm}^{\text{cred}}_{s_{\text{cred}}}(\mathrm{pk}_{\text{addr}} \| \mathrm{sk}_{\text{addr}})$.
7. Define $x := (\mathrm{cm}_{\text{cred}}, \mathrm{pk}_{\text{addr}})$ and $a := (\mathrm{sk}_{\text{addr}}, s_{\text{cred}})$.
8. The user computes the proof $\pi_{\texttt{CRED}} := \mathrm{Prove}^{\text{zkp}}(\mathrm{pk}_{\texttt{CRED}}, x, a)$.
9. Send x and $\pi_{\texttt{CRED}}$ to the admin, and wait until $\mathrm{cm}_{\text{cred}}$ is included in the Credential Merkle Tree.
10. Define $\mathrm{cred} := (\mathrm{pk}_{\text{addr}}, \mathrm{sk}_{\text{addr}}, \mathrm{pk}_{\text{enc}}, \mathrm{sk}_{\text{enc}}, s_{\text{cred}}, \mathrm{cm}_{\text{cred}})$.

AddAccount
Input: public parameters pp; public credential values x; credential proof $\pi_{\texttt{CRED}}$; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: tx

1. Parse pp as $(\mathrm{vk}_{\texttt{CRED}}, *)$.
2. Parse $\mathrm{cred}_{\text{adm}}$ as $(\mathrm{sk}_{\text{asig}}, *)$.
3. Parse x as $(\mathrm{cm}_{\text{cred}}, *)$.
4. If $\mathrm{Verify}^{\text{zkp}}(\mathrm{vk}_{\texttt{CRED}}, \pi_{\texttt{CRED}}, x)$ outputs `false`, then return $\perp$.
5. Define $m_{\text{add}} := (\textit{“Add credential”}, \mathrm{cm}_{\text{cred}})$.
6. Compute the signature on the message as $\sigma_{\text{add}} := \mathrm{Sign}^{\text{asig}}_{\mathrm{sk}_{\text{asig}}}(m_{\text{add}})$.
7. Publish $\mathrm{tx} := (m_{\text{add}}, \sigma_{\text{add}})$.
8. Add $\mathrm{cm}_{\text{cred}}$ to the Credential Merkle Tree and update $\mathrm{rt}_{\text{cred}}$.

RevokeAccount
Input: credential commitment $\mathrm{cm}_{\text{cred}}$; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: tx

1. Parse $\mathrm{cred}_{\text{adm}}$ as $(\mathrm{sk}_{\text{asig}}, *)$.
2. Define $m_{\text{rvk}} := (\textit{“Revoke credential”}, \mathrm{cm}_{\text{cred}})$.
3. Compute a signature on the message as $\sigma_{\text{rvk}} := \mathrm{Sign}^{\text{asig}}_{\mathrm{sk}_{\text{asig}}}(m_{\text{rvk}})$.
4. Publish $\mathrm{tx} := (m_{\text{rvk}}, \sigma_{\text{rvk}})$.
5. Remove $\mathrm{cm}_{\text{cred}}$ from the Merkle tree and update $\mathrm{rt}_{\text{cred}}$.

ConvertToNote
Input: public parameters pp; conversion value $v^{\text{new}}_{\text{note}}$; public destination address $\mathrm{pk}^{\text{new}}_{\text{addr}}$; destination encryption key $\mathrm{pk}^{\text{new}}_{\text{enc}}$; extra info info; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: transaction tx; $\mathrm{note}^{\text{new}}$

1. Parse pp as $(\mathrm{pk}_{\texttt{CTO}}, *)$.
2. Parse $\mathrm{cred}_{\text{adm}}$ as $(\mathrm{sk}_{\text{asig}}, *)$.
3. Randomly sample the commitment trapdoor $s^{\text{new}}_{\text{note}}$ for the new note.
4. Compute the commitment $\mathrm{cm}^{\text{new}}_{\text{note}}$ for the new note as $\mathrm{cm}^{\text{new}}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s^{\text{new}}_{\text{note}}}(\mathrm{pk}^{\text{new}}_{\text{addr}} \| v^{\text{new}}_{\text{note}}) \| 0$.
5. Define $\mathrm{data}^{\text{new}}_{\text{note}} := \mathrm{Enc}_{\mathrm{pk}^{\text{new}}_{\text{enc}}}(s^{\text{new}}_{\text{note}} \| v^{\text{new}}_{\text{note}} \| \mathrm{info})$.
6. Set $x := (\mathrm{cm}^{\text{new}}_{\text{note}})$.
7. Set $a := (s^{\text{new}}_{\text{note}}, v^{\text{new}}_{\text{note}}, \mathrm{pk}^{\text{new}}_{\text{addr}})$.
8. Obtain a zero-knowledge proof for the transaction $\pi_{\texttt{CTO}} := \mathrm{Prove}^{\text{zkp}}(\mathrm{pk}_{\texttt{XFER}}, x, a)$.
9. Define $m_{\text{cto}} := (x, \pi_{\texttt{CTO}}, \mathrm{data}^{\text{new}}_{\text{note}})$.
10. Compute a signature on the transaction message $\sigma_{\text{cto}} := \mathrm{Sign}^{\text{asig}}_{\mathrm{sk}_{\text{asig}}}(m_{\text{cto}})$.
11. Set $\mathrm{note}^{\text{new}} := (v^{\text{new}}_{\text{note}}, \mathrm{pk}^{\text{new}}_{\text{addr}}, s^{\text{new}}_{\text{note}})$.
12. Publish $\mathrm{tx} := (m_{\text{cto}}, \sigma_{\text{cto}})$.

ConvertFromNote

Input: public parameters pp; input note $\mathrm{note}^{\text{old}}$; Note Merkle root $\mathrm{rt}_{\text{note}}$; admin credentials $\mathrm{cred}_{\text{adm}}$
Output: transaction tx

1. Parse pp as $(\mathrm{pk}_{\texttt{CFROM}}, *)$.
2. Parse $\mathrm{cred}_{\text{adm}}$ as $(\mathrm{sk}_{\text{addr}}, \mathrm{sk}_{\text{asig}}, *)$.
3. Parse $\mathrm{note}^{\text{old}}$ as $(v^{\text{old}}_{\text{note}}, \mathrm{cm}^{\text{old}}_{\text{note}}, t^{\text{old}}_{\text{note}}, s^{\text{old}}_{\text{note}}, *)$.
4. Determine the position $\mathrm{pos}_{\text{note}}$ of $\mathrm{cm}^{\text{old}}_{\text{note}}$ in the Note Merkle Tree and its path $\mathrm{path}_{\text{note}}$ to $\mathrm{rt}_{\text{note}}$.
5. Determine the nullifier to $\mathrm{note}^{\text{old}}$ as $\eta := \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{note}})$.
6. Set $x := (\mathrm{rt}_{\text{note}}, \eta, \mathrm{pk}_{\text{addr}})$.
7. Set $a := (\mathrm{sk}_{\text{addr}}, s^{\text{old}}_{\text{note}}, v^{\text{old}}_{\text{note}}.t^{\text{old}}_{\text{note}}, \mathrm{cm}^{\text{old}}_{\text{note}}, \mathrm{path}_{\text{note}}, \mathrm{pos}_{\text{note}})$.
8. Create a zero-knowledge proof for the transaction $\pi_{\texttt{CFROM}} := \mathrm{Prove}^{\text{zkp}}(\mathrm{pk}_{\texttt{CFROM}}, x, a)$.
9. Define $m_{\text{cfrom}} := (x, \pi_{\texttt{CFROM}})$.
10. Compute a signature on the transaction message $\sigma_{\text{cfrom}} := \mathrm{Sign}^{\text{asig}}_{\mathrm{sk}_{\text{asig}}}(m_{\text{cfrom}})$.
11. Publish $\mathrm{tx} := (m_{\text{cfrom}}, \sigma_{\text{cfrom}})$.

CreateTransaction

Input: public parameters pp; credentials cred; Note Merkle root $\mathrm{rt}_{\text{note}}$; Memory Merkle root rt_{mem}; Credential Merkle root $\mathrm{rt}_{\text{cred}}$; old note $\mathrm{note}^{\text{old}}$; previous memory cell $\mathrm{mem}^{\text{old}}$; ceiling memory cell $\mathrm{mem}^{\text{ceil}}$; new note value $v^{\text{new}}_{\text{note}}$; new public address $\mathrm{pk}^{\text{new}}_{\text{addr}}$; new public encryption key $\mathrm{pk}^{\text{new}}_{\text{enc}}$; extra info info; new block time t^{new}; boolean value for including SAVER encrypted values b_{saver}; lock time new note t_{δ}
Output: transaction tx

1. Parse pp as $(\mathrm{pk}_{\texttt{XFER}}, \mathrm{pp}_{\text{enc}}, \mathrm{pp}_{\text{sig}}, \mathrm{pk}_{\text{svr}} *)$.
2. Parse cred to $(\mathrm{sk}_{\text{addr}}, \mathrm{pk}_{\text{addr}}, \mathrm{cm}_{\text{cred}}, s_{\text{cred}})$.
3. Determine the position $\mathrm{pos}_{\text{cred}}$ of $\mathrm{cm}^{\text{cred}}$ in the Account Merkle Tree and its path $\mathrm{path}_{\text{cred}}$ to $\mathrm{rt}_{\text{cred}}$.
4. Randomly sample two commitment trapdoors $s^{\text{new}}_{\text{note}}$, $s^{\text{new}}_{\text{mem}}$ for the new note.
5. If $\mathrm{note}^{\text{old}} = \perp$, then set $b_{\text{note}} = 0$, $\mathrm{pos}_{\text{note}} = 0$, $\mathrm{path}_{\text{note}} = 0$, $v^{\text{old}}_{\text{note}} = 0$, $s^{\text{old}}_{\text{note}} = 0$, $\eta = -1$, $t^{\text{old}}_{\delta} = 0$, $t^{\text{old}}_{\text{note}} = t^{\text{new}}$, compute $\mathrm{cm}^{\text{old}}_{\text{note}}$ and $\mathrm{rt}^{\text{int}}_{\text{note}}$ accordingly, and go to step 8.
6. Parse $\mathrm{note}^{\text{old}}$ as $(v^{\text{old}}_{\text{note}}, \mathrm{cm}^{\text{old}}_{\text{note}}, t^{\text{old}}_{\delta}, t^{\text{old}}_{\text{note}}, s^{\text{old}}_{\text{note}})$.
7. Determine the position $\mathrm{pos}_{\text{note}}$ of $\mathrm{cm}^{\text{old}}_{\text{note}}$ in the Note Merkle Tree and its path $\mathrm{path}_{\text{note}}$ to $\mathrm{rt}_{\text{note}}$ and set $\mathrm{rt}^{\text{int}}_{\text{note}} = \mathrm{rt}_{\text{note}}$.
8. Determine the nullifier η^{int} to $\mathrm{note}^{\text{old}}$ as $\eta^{\text{int}} := \mathrm{PRF}^{\eta}_{\mathrm{sk}_{\text{addr}}}(\mathrm{pos}_{\text{note}})$.
9. Calculate the commitment $\mathrm{cm}^{\text{new}}_{\text{note}}$ for the new note as $\mathrm{cm}^{\text{new}}_{\text{note}} := \mathrm{Comm}^{\text{note}}_{s^{\text{new}}_{\text{note}}}(\mathrm{pk}^{\text{new}}_{\text{addr}} \| v^{\text{new}}_{\text{note}} \| t_{\delta})$.
10. Define $\mathrm{data}^{\text{new}}_{\text{note}} := \mathrm{Enc}_{\mathrm{pk}^{\text{new}}_{\text{enc}}}(s^{\text{new}}_{\text{note}} \| v^{\text{new}}_{\text{note}} \| t_{\delta} \| \mathrm{info})$.
11. If $\mathrm{mem}^{\text{old}} = \perp$, then set $b_{\text{mem}} = 0$, $\mathrm{pos}_{\text{mem}} = -1$, $t^{\text{old}} = -(T+1)$, $v^{\text{old}}_{\text{mem}} = 0$, $s^{\text{old}}_{\text{mem}} = 0$, $c^{\text{old}}_{\text{mem}} = 0$, $\mathrm{path}_{\text{mem}} = 0$, $\mathrm{rt}^{\text{real}}_{\text{mem}} = \mathrm{rt}_{\text{mem}}$, and compute $\mathrm{cm}^{\text{old}}_{\text{mem}}$ and $\mathrm{rt}^{\text{int}}_{\text{mem}}$ accordingly and go to step 14.
12. Parse $\mathrm{mem}^{\text{old}}$ as $(v^{\text{old}}_{\text{mem}}, \mathrm{cm}^{\text{old}}_{\text{mem}}, s^{\text{old}}_{\text{mem}}, c^{\text{old}}_{\text{mem}}, t^{\text{old}}_{\text{mem}})$.
13. Determine the position $\mathrm{pos}_{\text{mem}}$ of $\mathrm{CRH}^{\text{mem}}(\mathrm{cm}^{\text{old}}_{\text{mem}}, t^{\text{old}}_{\text{mem}})$ in the Memory Merkle Tree and its path $\mathrm{path}_{\text{mem}}$ to rt_{mem} and set $\mathrm{rt}^{\text{real}}_{\text{mem}} = \mathrm{rt}^{\text{int}}_{\text{mem}} = \mathrm{rt}_{\text{mem}}$.
14. If $\mathrm{mem}^{\text{ceil}} = \perp$, then set $t^{\text{ceil}} = -(T+1)$, $v^{\text{ceil}}_{\text{mem}} = 0$, $s^{\text{ceil}}_{\text{mem}} = 0$, $c^{\text{ceil}}_{\text{mem}} = 0$, and compute $\mathrm{cm}^{\text{ceil}}_{\text{mem}}$ accordingly, and go to step 17.
15. Parse $\mathrm{mem}^{\text{ceil}}$ as $(v^{\text{ceil}}_{\text{mem}}, \mathrm{cm}^{\text{ceil}}_{\text{mem}}, s^{\text{ceil}}_{\text{mem}}, c^{\text{ceil}}_{\text{mem}}, t^{\text{ceil}}_{\text{mem}})$.

16. Determine the position $\mathrm{pos}_{\mathsf{ceil}}$ of $\mathrm{CRH}^{\mathsf{mem}}(\mathrm{cm}^{\mathsf{ceil}}_{\mathsf{mem}}, t^{\mathsf{ceil}}_{\mathsf{mem}})$ in the Memory Merkle Tree and its path $\mathrm{path}_{\mathsf{ceil}}$ to $\mathrm{rt}_{\mathsf{mem}}$.
17. Determine the nullifier μ to $\mathrm{mem}^{\mathsf{old}}$ as $\mu := \mathrm{PRF}^{\mu}_{\mathrm{sk}_{\mathsf{addr}}}(\mathrm{pos}_{\mathsf{mem}})$.
18. Determine $v^{\mathsf{new}}_{\mathsf{mem}} := v^{\mathsf{old}}_{\mathsf{note}} + v^{\mathsf{old}}_{\mathsf{mem}} - v^{\mathsf{new}}_{\mathsf{note}}$.
19. Determine $c^{\mathsf{new}} := c^{\mathsf{old}} + v^{\mathsf{new}}_{\mathsf{note}} \cdot (1 - b_{\mathsf{saver}})$.
20. Calculate the new memory commitment $\mathrm{cm}^{\mathsf{new}}_{\mathsf{mem}} := \mathrm{Comm}^{\mathsf{mem}}_{s^{\mathsf{new}}_{\mathsf{mem}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{new}}_{\mathsf{mem}} \| c^{\mathsf{new}})$.
21. Define $\mathrm{data}^{\mathsf{new}}_{\mathsf{mem}} := \mathrm{Enc}_{\mathrm{pk}_{\mathsf{enc}}}(s^{\mathsf{new}}_{\mathsf{mem}})$.
22. Randomly generate a signature public and private key pair $(\mathrm{pk}_{\mathsf{sig}}, \mathrm{sk}_{\mathsf{sig}}) := \mathrm{KeyGen}_{\mathsf{sig}}(\mathrm{pp}_{\mathsf{sig}})$.
23. Compute $k := \mathrm{CRH}^{\mathsf{sig}}(\mathrm{pk}_{\mathsf{sig}})$ and calculate $\kappa := \mathrm{PRF}^{\kappa}_{\mathrm{sk}_{\mathsf{addr}}}(k)$, which ties the signature public key to the secret address key of the sender.
24. Determine the inputs for the SAVER encryption: $\overline{\mathrm{pk}}_{\mathsf{sndr}} = \mathrm{pk}_{\mathsf{addr}} \cdot b_{\mathsf{saver}}$, $\overline{\mathrm{pk}}_{\mathsf{rcvr}} = \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}} \cdot b_{\mathsf{saver}}$, and $\overline{v}_{\mathsf{xfer}} = v^{\mathsf{new}}_{\mathsf{note}} \cdot b_{\mathsf{saver}}$
25. Set $m_{\mathsf{svr}} := (\overline{\mathrm{pk}}_{\mathsf{sndr}}, \overline{\mathrm{pk}}_{\mathsf{rcvr}}, \overline{v}_{\mathsf{xfer}})$.
26. Set $x := (\mathrm{rt}_{\mathsf{mem}}, \mathrm{rt}_{\mathsf{note}}, \mathrm{rt}_{\mathsf{cred}}, k, \kappa, \eta, \mu, \mathrm{cm}^{\mathsf{new}}_{\mathsf{note}}, \mathrm{cm}^{\mathsf{new}}_{\mathsf{mem}}, t^{\mathsf{new}})$.
27. Set $a := (\mathrm{rt}^{\mathsf{real}}_{\mathsf{mem}}, \mathrm{rt}^{\mathsf{int}}_{\mathsf{mem}}, \mathrm{rt}^{\mathsf{int}}_{\mathsf{note}}, \eta^{\mathsf{int}}, \mathrm{pk}_{\mathsf{addr}}, \mathrm{sk}_{\mathsf{addr}}, s_{\mathsf{cred}}, \mathrm{cm}_{\mathsf{cred}}, v^{\mathsf{old}}_{\mathsf{note}}, s^{\mathsf{old}}_{\mathsf{note}}, t^{\mathsf{old}}_{\delta}, \mathrm{cm}^{\mathsf{old}}_{\mathsf{note}}, t^{\mathsf{old}}_{\mathsf{note}}, v^{\mathsf{old}}_{\mathsf{mem}}, s^{\mathsf{old}}_{\mathsf{mem}}, c^{\mathsf{old}}, \mathrm{cm}^{\mathsf{old}}_{\mathsf{mem}}, t^{\mathsf{old}}_{\mathsf{mem}}, v^{\mathsf{ceil}}_{\mathsf{mem}}, s^{\mathsf{ceil}}_{\mathsf{mem}}, c^{\mathsf{ceil}}, \mathrm{cm}^{\mathsf{ceil}}_{\mathsf{mem}}, t^{\mathsf{ceil}}_{\mathsf{mem}}, \mathrm{path}_{\mathsf{cred}}, \mathrm{path}_{\mathsf{note}}, \mathrm{path}_{\mathsf{mem}}, \mathrm{path}_{\mathsf{ceil}}, \mathrm{pos}_{\mathsf{note}}, \mathrm{pos}_{\mathsf{mem}}, \mathrm{pos}_{\mathsf{cred}}, \mathrm{pos}_{\mathsf{ceil}}, s^{\mathsf{new}}_{\mathsf{note}}, s^{\mathsf{new}}_{\mathsf{mem}}, v^{\mathsf{new}}_{\mathsf{note}}, v^{\mathsf{new}}_{\mathsf{mem}}, c^{\mathsf{new}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}, b_{\mathsf{note}}, b_{\mathsf{mem}}, b_{\mathsf{saver}}, t_{\delta})$.
28. Obtain a zero-knowledge proof and ciphertext for the transaction $(\pi_{\mathtt{XFER}}, \mathrm{data}_{\mathsf{saver}}) := \mathrm{Enc}^{\mathsf{svr}}(\mathrm{pk}_{\mathtt{XFER}}, \mathrm{pk}_{\mathsf{svr}}, m_{\mathsf{svr}}, x, a)$.
29. Define $m_{\mathsf{xfer}} := (x, \pi_{\mathtt{XFER}}, \mathrm{data}^{\mathsf{new}}_{\mathsf{note}}, \mathrm{data}^{\mathsf{new}}_{\mathsf{mem}}, \mathrm{data}_{\mathsf{saver}})$.
30. Compute a signature on the transaction message $\sigma_{\mathsf{xfer}} := \mathrm{Sign}^{\mathsf{sig}}_{\mathrm{sk}_{\mathsf{sig}}}(m_{\mathsf{xfer}})$.
31. Set $\mathrm{note}^{\mathsf{new}} := (v^{\mathsf{new}}_{\mathsf{note}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}, s^{\mathsf{new}}_{\mathsf{note}})$.
32. Set $\mathrm{mem}^{\mathsf{new}} := (v^{\mathsf{new}}_{\mathsf{mem}}, \mathrm{pk}_{\mathsf{addr}}, s^{\mathsf{new}}_{\mathsf{mem}})$.
33. Publish $\mathrm{tx} := (m_{\mathsf{xfer}}, \mathrm{pk}_{\mathsf{sig}}, \sigma_{\mathsf{xfer}})$.

VerifyTransaction
Input: public parameters pp; transaction tx; current blockchain B
Output: boolean value for correctness b

1. Parse pp as $(\mathrm{vk}_*, \mathrm{pk}_{\mathsf{svr}}*)$.
2. Parse tx as $(m, \mathrm{pk}_{\mathsf{sig}}, \sigma)$, if transaction type is *not* regular transfer transaction get the public admin key $\mathrm{pk}_{\mathsf{asig}}$.
3. Parse m as $(x, \pi, \mathrm{data}_{\mathsf{saver}}, *)$.
4. If the transaction type is *not* a regular transfer transaction, go to step 10.
5. Parse x as $(\mathrm{rt}_{\mathsf{cred}}, \mathrm{rt}_{\mathsf{mem}}, \mathrm{rt}_{\mathsf{node}}, k, \kappa, \eta, \mu, t^{\mathsf{new}}, *)$.
6. If $\mathrm{rt}_{\mathsf{mem}}$, $\mathrm{rt}_{\mathsf{cred}}$ and $\mathrm{rt}_{\mathsf{node}}$ do not appear in the same block on B, then output `false`.
7. If t^{new} is not close to the current block time, then output `false`.
8. If η or μ does appear on B, then output `false`.
9. If k does not equal $\mathrm{CRH}^{\mathsf{sig}}(\mathrm{pk}_{\mathsf{sig}})$, then output `false`.
10. If $\mathrm{Verify}^{\mathsf{(a)sig}}_{\mathrm{pk}_{\mathsf{(a)sig}}}(m, \sigma)$ outputs `false`, then output `false`.
11. Define $b := \mathrm{VerifyEnc}^{\mathsf{svr}}(\mathrm{vk}_*, \mathrm{pk}_{\mathsf{svr}}, \pi, \mathrm{data}_{\mathsf{saver}}, x)$ if the transaction type is a regular transaction, otherwise $b := \mathrm{Verify}^{\mathsf{zkp}}(\mathrm{vk}_*, \pi, x)$.

ReceiveTransaction
Input: public parameters pp; new transaction tx; credentials cred; current blockchain B
Output: received note $\mathrm{note}^{\mathsf{new}}$

1. If transaction type is not a regular transfer transaction, then output $\perp$.
2. If $\mathrm{VerifyTransaction}(\mathrm{pp}, \mathrm{tx}, B)$ outputs `false`, then output $\perp$.
3. Parse cred as $(\mathrm{sk}_{\mathsf{enc}}, \mathrm{pk}_{\mathsf{addr}}, *)$.
4. Parse tx as $(m_{\mathsf{xfer}}, *)$.
5. Parse m_{xfer} as $(x, \mathrm{data}_{\mathsf{note}}, *)$.
6. Parse x as $(\mathrm{cm}^{\mathsf{new}}_{\mathsf{note}}, *)$.
7. Compute $(s^{\mathsf{new}}_{\mathsf{note}}, v^{\mathsf{new}}_{\mathsf{note}}, t^{\mathsf{new}}_{\delta}, \mathrm{info}) = \mathrm{Dec}_{\mathrm{sk}_{\mathsf{enc}}}(\mathrm{data}^{\mathsf{new}}_{\mathsf{note}})$, if the output was $\perp$, then output $\perp$.
8. If $\mathrm{cm}^{\mathsf{new}}_{\mathsf{note}}$ does not equal $\mathrm{Comm}^{\mathsf{note}}_{s^{\mathsf{new}}_{\mathsf{note}}}(\mathrm{pk}_{\mathsf{addr}} \| v^{\mathsf{new}}_{\mathsf{note}})$, then output $\perp$.
9. Set $\mathrm{note}^{\mathsf{new}} := (v^{\mathsf{new}}_{\mathsf{note}}, s^{\mathsf{new}}_{\mathsf{note}}, \mathrm{info}, \mathrm{cm}^{\mathsf{new}}_{\mathsf{note}})$.

6.4 Completeness and security

The construction as presented in the above Section is also complete and secure. We verify this by showing that all our requirements, as defined in Section 4.1, hold. In Appendix C we prove for each individual requirement that it holds for the concrete scheme as defined in this Chapter. It should be noted that even though all requirements hold separately, we cannot guarantee that they all hold at once. Given that not all of the used cryptographic primitives are universally composable, we were unable to achieve a proof of this more general statement. Nevertheless, we have no reason to doubt that all requirements do hold in a universal setting.

Chapter 7

Implementation

In this chapter we discuss some considerations to take into account when implementing the concrete scheme as presented in the previous chapter. In Section 7.1 we discuss some details regarding key management for both administrators and users. The concrete instantiation of the cryptographic building blocks is discussed in Section 7.2.

Some details regarding the transformation of our zk-SNARK statements into arithmetic circuits as well as implementation details thereof are discussed in Section 7.3. An implementation of our scheme in Rust is discussed in Section 7.4. In this final section we also present performance benchmarks for the key generation, prove, and verification times for all four arithmetic circuits.

7.1 Key management

Since the administrator and regular users have different sets of keys, and require different management of these keys, we discuss the key management separately for both roles. We also, briefly discuss the key management of the secret SAVER decryption key for the judges, as well as the other individually generated randomness. We do not focus on public parameters or proof and verification keys since these public parameters should just be visible for all participants and require no special attention.[1]

Administrator. An administrator has three public-private key pairs, one for a signature scheme, one for an encryption scheme, and one that functions as an address. The keys for the signature scheme can be generated at random by the admin that uses it. In the case of more administrators, each one should have their own key pair to make the sender of each message identifiable. The size of these keys depends on the security parameter and the type of signature scheme that is used.

[1]This also holds for the SAVER public key and verification key.

Each admin should generate a random secret address key with the amount of bits that the seed of a PRF is required to have, from which the public address key can be derived. This secret address key should be kept secret at all times by the administrator. From this secret address key, the admin also derives the asymmetric encryption key pair. The secret encryption key should also be kept secret. The size of the encryption keys depends on the security parameter and the type of encryption scheme used.

After having generated all the keys, the admin should post all public keys in a location that is open to any (potential) user of the system. These keys should also be posted in such a way that it is clear to all other users to which legal entity these keys belong.

User. A regular user has two public-private key pairs: an encryption key pair and an address key pair. These key pairs have the same details as those of the admin and are also obtained in the same way. The user need however only publish its public key pairs to the people that the user wants to receive transactions from. This could either be sent to another user directly, or published openly visible to the world. A user could treat the keys similar to how one would normally deal with a bank account number.

Next to this, the user also has a commitment to its address key pair, which requires a trapdoor s_{cred} to create randomness. The commitment is sent to an admin along with the public address key. This ensures that the admin knows which public address key belongs to which physical entity. This is important for connecting the anonymous values to fiat currency. It also allows the bank to perform KYC for any new user.

We want to ensure that users do not have to remember a large amount of keys, trapdoors to commitments, and other secrets. Therefore, the protocol is designed in such a way that only two values need to be remembered by the user: the secret address key $\mathrm{sk}_{\text{addr}}$ and the credential commitment trapdoor s_{cred}. This secret address key can be used to derive all the other keys, and together with the commitment trapdoor these keys allow a user to proof knowledge of the credential commitment in the Account Merkle tree.

The user could then easily obtain all existing unspent notes by scanning the blockchain for any received notes, to which the nullifiers have not been published yet. Moreover, all the user's memory cells can be found by scanning the blockchain for transactions to which the $\mathrm{data}_{\text{mem}}$ field can be decrypted and leads to a valid trapdoor for the related memory cell commitment. The user does not need to have more information to re-obtain the current state of its account in the payment scheme.

Judge. All judges participate in a secure multi-party computation protocol to perform the joint key generation for SAVER. During this protocol each judge j generates some random values s_i^j, z_i^j, t_i^j, and ρ^j. These first three values should all be kept secret from other judges, as they do leak some information about the exponents in the public and verification keys that can be used to decrypt ciphertexts. The value ρ^j should clearly be kept secret as well, since each value ρ^j is a share to the secret decryption key. Finally, the judges also generate some random values r_i^j as masking for intermediate values. These masking values should also be kept secret as they do reveal some intermediate values that could be used to decrypt ciphertexts.

7.2 Instantiation of cryptographic building blocks

We will now discuss how to instantiate the cryptographic building blocks as were defined above. We will do this by making some choices that give a level of security of at least 128 bits. Many of these choices can also be made differently to try and improve performance, security or make implementation simpler when this is needed. We make no claim that the made choices are optimal considering performance, implementation or security. Though, the choices should prove decent for all three factors.

CRH, PRF, COMM and Merkle trees. These primitives could all be based on several different hash functions, with minor adaptations. Options include: SHA-256, MiMC, Blake2s, Poseidon, Starkad, or the Pedersen hash. Some of these hashes are more efficient in arithmetic circuits, while others provide a higher level of security. SHA-256 and Blake2s are both very secure and well-known hash functions. A downside to both algorithms is that they require a lot of constraints when modelled as an arithmetic circuit, i.e. they are not so efficient. On the other hand we have MiMC, Poseidon, Starkad, and the Pedersen hash. All of these algorithms require a low amount of constraints when modelled as an arithmetic circuits, i.e. they are very efficient. A downside of the first three algorithms is that the constructions are relatively new and security seems weaker. Already, some collision and preimage attacks have been constructed against instantiations of these algorithms [37]. That leaves us with the Pedersen hash. The Pedersen hash is very efficient, and is also known to be rather secure. We do note that it is considered to be less secure than SHA-256 and Blake2s due to its discrete log based security assumption. Interestingly, this assumption is not weaker than the knowledge of exponent based assumptions of our zk-SNARK scheme. Therefore, using a Pedersen hash in a zk-SNARK scheme does not weaken security on that front.

In light of the above discussion we decide to use Blake2s[2] as building block for our PRF. We decide to use a stronger level of security here in order to make *stealing* or *destroying* a note more difficult. Both stealing and destroying a note of another user requires the attacker to publish the true nullifier, i.e. finding a collision or learning $\mathrm{sk}_{\mathsf{addr}}$. The security thereof is thus independent of the security of the zk-SNARK scheme and using Pedersen hash would thus decrease this level of security. *Forgery* and *duplication* of a note is another matter, since an attacker simply produces a valid proof for a random nullifier. This can be done by either breaking the zk-SNARK scheme, or finding a collision in COMM or CRH. Therefore, using Blake2s for our commitment scheme or CRH would not really increase security with regard to forgery or duplication. Practically this implies that we can use the more efficient Pedersen hash as building block for COMM and CRH [38]. We will use the Pedersen hash and Pedersen commitment as defined in the ZCash protocol [32][3] since it is optimised for use in our zk-SNARK scheme.

For our PRF we will use Blake2s with a 512 bit (32 byte) input and a 256 bits output, which provides us with a 128-bits security level against collision attacks and a 256-bits security level against (second-)preimage attacks. We denote our Blake2s function by $\mathcal{H}$.

We define our Note and Memory Merkle trees to have depth 32, this allows for 2^{32} leafs, which should be more than sufficient for the protocol to function a long time without needing a reset of the Merkle trees. This implies that the position of a commitment in a Merkle tree pos can be represented by a 33-bit number (32 bits for the position and 1 bit extra for the case of using -1 as the position). The Credential Merkle tree has depth 20.

Using the hash function $\mathcal{H}$ we define our pseudorandom function as $\mathrm{PRF}_x(s) := \mathcal{H}(x\|s)$, with $x, s \in \{0,1\}^{256}$. Specifically the four versions of our PRF are now defined as follows, with $[\cdot]_x$ being the function that truncates its input to a size of x bits:

- $\mathrm{PRF}_x^{\mathsf{addr}}(s) := \mathcal{H}(x\|00\|s)$, with $x \in \{0,1\}^{256}$ and $s \in \{0,1\}^{254}$;
- $\mathrm{PRF}_x^{\eta}(s) := \mathcal{H}(x\|01\|0^{191}\|s)$, with $x \in \{0,1\}^{256}$ and $s \in \{0,1\}^{33}$;
- $\mathrm{PRF}_x^{\mu}(s) := \mathcal{H}(x\|10\|0^{191}\|s)$, with $x \in \{0,1\}^{256}$ and $s \in \{0,1\}^{3}$;
- $\mathrm{PRF}_x^{\kappa}(s) := \mathcal{H}(x\|11\|[s]_{254})$, with $x \in \{0,1\}^{256}$ and $s \in \{0,1\}^{256}$.

Consistently with the above defined PRFs, we take our public and secret address key to have bit length 256. Using this fact we can define the credential commitment using the Pedersen hash. We will denote the Pedersen hash function as $\mathcal{P}$. Our

[2]It is slightly more efficient than SHA-256 and has similar level of security.

[3]See Section 5.4 *Concrete cryptographic schemes* for the specification.

Pedersen hash function takes as input a bit string of arbitrary length and outputs an element in the Jubjub curve as defined in the ZCash protocol [32]. It should be noted that this hash function is only collision-resistant for fixed-length inputs. Fortunately, given the fixed size keys and randomness this is not an issue for our implementation.

Given this Pedersen hash function and three pre-defined elements on the Jubjub Curve J_1, J_2, and J_3 we define our commitment functions as:

$$\mathrm{COMM}^{\textsf{cred}}_{s_{\textsf{cred}}}(\mathrm{pk}_{\textsf{addr}} \| \mathrm{sk}_{\textsf{addr}}) := \mathcal{P}(\mathrm{pk}_{\textsf{addr}} \| \mathrm{sk}_{\textsf{addr}}) \cdot J_1^{s_{\textsf{cred}}},$$

with $s_{\textsf{cred}} \in \{0,1\}^{252}$. Moreover, we define $v_{\textsf{max}} := 2^{64} - 1$ for convenience. This implies that v will have bit length of 64 which is sufficient to denote any existing amount of fiat currency existing. Consistently, we define c and t_δ to have a bit length of 64 as well. This gives rise to the following construction of a note commitment

$$\mathrm{COMM}^{\textsf{note}}_{s_{\textsf{note}}}(\mathrm{pk}_{\textsf{addr}} \| v_{\textsf{note}}) := \mathcal{P}(\mathrm{pk}_{\textsf{addr}} \| v_{\textsf{note}} \| t_\delta) \cdot J_2^{s_{\textsf{note}}}.$$

Similarly, we define our memory commitment as

$$\mathrm{COMM}^{\textsf{mem}}_{s_{\textsf{mem}}}(\mathrm{pk}_{\textsf{addr}} \| v_{\textsf{mem}}) := \mathcal{P}(\mathrm{pk}_{\textsf{addr}} \| v_{\textsf{mem}} \| c_{\textsf{mem}}) \cdot J_3^{s_{\textsf{mem}}}.$$

Similarly, we define the hash function for our Merkle tree CRH as $\mathcal{P}$, which takes as input two Merkle nodes and outputs their parent. In order to obtain collision-resistance we require our input to be of fixed size. When used to compute a parent Merkle noe, this function is the concatenation of two child Merkle nodes, therefore our requirement is trivially satisfied. We will now show, that our other instantiations of CRH also satisfy the fixed input size requirement.

We define $\mathrm{CRH}^{\textsf{mem}}$ and $\mathrm{CRH}^{\textsf{note}}$ using $\mathcal{P}$ as $\mathrm{CRH}^{\textsf{mem}}(\mathrm{cm}_{\textsf{mem}}, t) = \mathcal{P}(\mathrm{cm}_{\textsf{mem}} \| t)$ and $\mathrm{CRH}^{\textsf{note}}(\mathrm{cm}_{\textsf{note}}, t) = \mathcal{P}(\mathrm{cm}_{\textsf{note}} \| t)$. To ensure collision-resistance we also require these inputs to be of fixed size. The memory commitment is trivially of fixed size, and we can define the length of t as fixed ourselves. To ensure the ability to store enough values of time we define our time t to be stored in a 64 bit value. That should be more than sufficient to store time accurately enough for a long time.

On the other hand, we define our collision-resistant hash $\mathrm{CRH}^{\textsf{sig}}$ function to use $\mathcal{H}$. Because this hash function needs not be encoded inside the arithmetic circuit, efficiency does not matter as much and we simply choose the more secure option.

Signatures. Both signature schemes, the one used by the admin and the one used in anonymous transactions can be instantiated using EdDSA [39]. Normally, EdDSA only provides $\mathrm{EUF\text{-}CMA}$ security, however if one introduces an additional check on the domain of the signature [40] it provides $\mathrm{SUF\text{-}CMA}$ (and thus also $\mathrm{SUF\text{-}1CMA}$) security.

Encryption and KDF. For the key private encryption scheme we choose to use ECIES [41], with any appropriate key derivation to derive the user's private key. Specifically, we use a SHA-512 derived Hash KDF to generate X25519 key pairs. The used encryption algorithm is CHACHA20-POLY1305. We choose to adopt ECIES since it is a standardised and widely adopted scheme with available implementations and fits our security requirement.

7.3 Arithmetic circuit construction

The four arithmetic circuits as presented earlier in this report form the backbone of our transaction scheme. Most of the statements in the circuits follow rather straightforwardly from the algorithms and data structures that are used, however some details require some extra clarification. In this paragraph, we give some more insight in the design choices that might be somewhat unclear and give some details on how one could implement these circuits in such a way that actual proofs can be constructed an verified.

Design choices. One of the choices that was made is allowing a transaction without an input note, this means that the user only pays with value from the account balance. This gives the necessary opportunity to allow a user to spend value, without having an unspent transaction output. One could wonder, if users should also be allowed to perform a transaction without an output note. We choose not to, since this would not give rise to a new feature, i.e. the user can just produce an output note with value 0 to achieve the same thing. This increases the set of notes in the Merkle tree, which is favourable for anonymity. Hence, it is preferable when users produce an output note with value 0 instead of allowing for the option of no output note.[4]

We also choose not to verify the encrypted data fields in the transactions. Firstly, encryption algorithms are rather complex and would thus require a lot of constraints in the arithmetic program. This would result in a significant increase in prover and verifier time. Also, the sender of a transaction has no reason to want to tamper with the encrypted field, since if the receiver can not use the transaction output the transferred value is simply lost by the sender. Next to this, other users cannot tamper

[4]Our scheme functions as a sort of hybrid between the account-based and UTXO model. A full account-based model would also have been possible, by splitting a note-to-note transaction in a spend and a receive transaction, where the spend transaction publishes the new note for the receiver. The receive transaction, subsequently, adds the value of the note to the receiver's account and publishes to note nullifier. However, because of the zk-SNARKs involved, this would be less efficient. Moreover, since sender and receiver are likely to act at almost the same point in time in such a purely account-based system, resilience against side-channel attacks might be reduced.

with the data field due to the non-malleability of the message. This leads us to conclude that there would be no benefits in including this in the zk-SNARK proofs, whereas there are non-negligible disadvantages.

The XFER arithmetic circuit contains a statement that prohibits a transaction from being empty, i.e. when there is no value transferred to either the account balance or the output note. Even though this statement is not strictly necessary, it seems desirable to include this since it prevents flooding of the blockchain with 'empty' transactions. Moreover, this statement is rather efficiently implementable in an arithmetic circuit and thus has negligible influence on prover and verifier time.

In most of the arithmetic circuits we need to prove membership of a commitment in the Merkle Tree. In the statements this is defined as the sender knowing a path from the specific commitment cm to the Merkle root rt of the tree that contains the commitment. We also need pos to be the position of cm in the Merkle tree. This path is actually a sorted list of hashes containing all the values of the nodes that are needed to reconstruct rt from cm. So a path contains as first element the hash (in this case a commitment) of cm's sibling, then it contains the hash value of the sibling of cm's parent, and the hash of that parent's sibling, and so on. By recursively taking hashes over all these values one should eventually end up at rt, which then proves that cm is indeed contained in the Merkle tree with root rt. The position pos comes into play when determining whether the current node in the tree is a left or right child of its parent. In the binary representation of pos a 0 at position i denotes that the child at level i is on the left, and a 1 denotes that the child is on the right. Hence, we can use this fact to calculate rt using cm and path and also verify that cm is indeed at position pos.

The commitment's position pos is used in the construction the nullifier to a commitment. As far is the authors know, this idea was adopted for the first time in ZCash Sapling [32]. Since pos is unique for each commitment, and our PRF is collision-resistant we know that the produced nullifier is unique and deterministic for each commitment. It needs to be deterministic to prevent double usage of a commitment, and it needs to be unique to ensure that every commitment can be used at least once. Together, this implies that each commitment can be used exactly once, which is precisely the goal of the nullifier for memory cells and notes.

Implementation. Another aspect of the arithmetic circuits used is how to implement these circuits such that one can actually construct and verify proofs over these circuits. There is a small number of libraries that implement zk-SNARK logic [42], of which two seem the most promising: `libsnark` and `bellman`. In our opinion `bellman` is the better library, it provides more gadgets, has been updated more recently and has been developed for use in ZCash. Since our protocol is based on the protocol of

ZCash it seems most efficient to use the `bellman` library.

Due to the difficulty of implementing circuits of complex hash functions and other cryptographic primitives, one is mainly limited to using the primitives provided in `bellman`. Fortunately, our use case is similar to that of ZCash so the building blocks that we choose to use are also present in the library.

For completeness, we note that the library uses the BLS12-381 elliptic curve for implementing zk-SNARKs. It also provides us with the possibility of implementing elliptic curve arithmetic in an arithmetic circuit efficiently on an embedded curve inside BLS12-381, otherwise known as *Jubjub*. This last feature is very useful in implementing our use of Pedersen hash and commitment functions.

7.4 Performance

Using the `bellman` library as discussed above, we implemented a proof of concept of our payment scheme in Rust. This implementation, not only focuses on the zk-SNARK proof construction, but also implements all other steps in the protocol. This proof of concept is tested extensively to validate that our protocol works as desired. Next to this, we use the proof of concept to obtain performance metrics on our scheme. We will discuss the performance metrics below.

For each type of transaction we will discuss the proof size, SAVER ciphertext size if present, and the full transaction size. Next to this, we will discuss the relevant time-related metrics for the zk-SNARK scheme *Groth16* and the verifiable encryption scheme SAVER. These two parts of the transaction time together account for the significant part of the time measurements. The time it takes to perform other computations relevant for the payment scheme are implementation dependent and negligible with respect to *Groth16* and SAVER times.

For *Groth16* we will discuss the time cost of the parameter generation, proof construction, and proof validation for all four arithmetic circuits. For the *Transfer* transaction the parameter generation time includes the generation of SAVER parameters. Similarly, the proof generation time of *Transfer* includes the time of the ciphertext encryption. The validation will consider the SAVER adapted validation of the *Groth16* proof. Next to this, we discuss the time at takes to decrypt the SAVER ciphertext and verify that decryption.

In Table 7.1 we list all the relevant sizes of all transactions in our system. Apart from the *RvkAccount* transaction as it plays no major role in our payment scheme, and the size is relatively small compared to the other transactions.

We do note that all size in Table 7.1 are implementation-dependent and could be made slightly smaller with some minor optimisations or different choices of primitives. We have set the block size of the SAVER encryption to 2 bytes (16 bits), giving us a

Transaction type	Proof size	SAVER ciphertext size	Total size
NewAccount	192B	-	320B
ConvertTo	192B	-	376B
ConvertFrom	192B	-	352B
Transfer	192B	1824B	2640B

Table 7.1: The proof size, SAVER ciphertext size (if present), and total transaction size in bytes for all transaction types.

good trade-off between ciphertext size and decryption speed. With a total plaintext size of 72 bytes, this gives us a total ciphertext size of 1824 bytes. This is somewhat larger than we would like, however it is not infeasible. It causes the blockchain to grow quite sizeable, but not so large that it cannot be stored any longer. For comparison we note that a shielded transaction in Zcash also has a size of approximately 2000 bytes [32].

The proof size of our SNARK proof is always constant, irrespective of the size of the statement. This helps in keeping the transaction sizes at a practical level. This especially helps us to keep the size of the complex *Transfer* transaction proof within practical bounds.

The size of a zk-SNARK statement and the time it takes to compute a proof is best expressed in the amount of constraints required to encode the statement and the size of the common reference string. These values shown in Table 7.2 for all four circuits. The size of the SAVER keys for a *Transfer* transaction are approximately 27.9KiB. It is evident that none of the common reference string sizes are so large that it is inconvenient to store them locally. All strings could even be easily stored on a thumb drive or a mobile phone if that were required.

Arithmetic circuit	Number of constraints	CRS size
`CRED`	23,148	11.7MiB
`CTO`	1,972	841.1KiB
`CFROM`	89,305	42.2MiB
`XFER`	239,138	100.6MiB

Table 7.2: The number of constraints and the size of the common reference string (CRS) for each arithmetic circuit.

The number of constraints in a circuit tells us something about the amount of time it takes to compute and validate a zk-SNARK proof for that circuit. The higher the number of constrains, the longer the computations take. We show the computation times of our implementation in Table 7.3. We tried to optimise the computation times in our implementation as much as possible, but not to every extent. There are still some minor optimisations possible regarding the number of constraints and in parallelising smaller computations. We did not implement these optimisations as they would not result in significant improvements in any of the computation times.

The computation times in Table 7.3 are the median computation times of 9 runs. The runs were performed on a Windows 10 desktop PC with a Ryzen 3600 CPU with 6 cores and 12 threads @4.0GHz and dual-channel DDR4 RAM at 3600MHz. The Rust version is 1.46. No particular effort was taken in preventing other processes from running at the same time.

As can be seen in Table 7.3 the time it takes to verify a proof is very small. Tens to hundreds of transactions can be verified in one second, allowing all participants in the payment scheme to verify all new transactions. Next to this, we see that the parameter generation times are quite a bit larger. However, as this parameter generation only needs to be executed once this is not a problem at all.

Table 7.3 also shows us that proof creation does not take up a too large amount of time. This was also a necessary requirement, since users and administrators should not have to wait too long before a transaction is completed. A proof time of 0.5 to 2 seconds is more than reasonable in practice. With older hardware this proof time might increase to 5 to 10 seconds. This would still be acceptable, however not ideal. We can thus conclude that the proof times are practical, but could be improved upon.

Arithmetic circuit	**Parameter generation**	**Proof creation**	**Proof verification**	**SAVER decryption**	**SAVER decryption verification**
`CRED`	2.8170	0.2182	0.0029	-	-
`CTO`	0.2566	0.0535	0.0026	-	-
`CFROM`	9.9939	0.8155	0.0030	-	-
`XFER`	24.5045	1.8742	0.0226	0.8799	0.0345

Table 7.3: The median computation time in seconds for zk-SNARK parameter generation, proof construction, and proof verification for all four circuits. In the `XFER` circuit these times also include the complementary SAVER computations. We also show the decryption time and decryption verification time for the SAVER ciphertext. We take the median value of 9 runs on a desktop PC with 6 core CPU @4.0GHz and RAM at 3600MHz.

The decryption of a SAVER ciphertext can be performed in under a second, and the time it takes to verify this decryption is nearly negligible. We do not expect that both algorithms will be executed many times, and a waiting time of under a second is more than sufficient for the use case. We are thus more than satisfied with the efficiency thereof. Increasing the block size of the SAVER ciphertext to decrease the ciphertext size is unfortunately not possible. The decryption time increases exponentially with the block size. Thus, increasing the block size with only a couple of bits would make the decryption practically infeasible.

All in all, the proof of concept implementation shows that our payment scheme works as desired. Even though we would like to decrease some elements of the transaction in computation time and size, the transaction sizes and computation times are suitable for use in practice.

Chapter 8

Discussion and future work

8.1 Conclusion

In this thesis we have seen that zero-knowledge proofs and in particular zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) are a very suitable primitive for anonymous payment solutions. Especially, when we expect a system to do more than anonymously transferring value. We have seen that zk-SNARKs are useful when making a permissioned, yet anonymous, payment scheme. Moreover, without zero-knowledge proofs it would be very hard to integrate spend limits, verifiable encryption, and timelocks in a system without losing anonymity. One might even say that zero-knowledge proofs should be the primitive of choice for more advanced anonymous payment schemes.

We have been able to incorporate a spend limit in our payment scheme, without giving in on transaction anonimity. This spend limit ensures that users cannot anonymously spend a lot of money in a short amount of time. Limiting spending behaviour, together with the ability for know your customer (KYC) in our permissioned payment scheme, should help a lot in preventing abuse.

When a spend limit turns out to be not sufficient for a user, the user can also use verifiable encryption to spend more money. With the incorporation of the SAVER scheme we enable users to enclose transaction details in any transaction. These transaction details are not visible to any other user of the system. Only a select set of judges can view these transaction details, if at any later point in time certain transactions are mistrusted by authorities. The advantage of using SAVER for this system, is that we do not require a lot of additional computation time for creating a transaction.

Finally, we have also shown how to introduce anonymous timelocks in our payment scheme. This can be used to prevent users of just received funds to immediately spend them. This might be used in future work to enable faster payment schemes, or escrow systems.

We have also proven security and completeness of our decentralised anonymous payment scheme in a formal way. These proofs ensure that no user can forge or steal money, and that published transactions reveal nothing about the sender, receiver, or transferred value. An implementation of our scheme was also provided to show that the system actually works in practice and has a more than acceptable performance. The validation times for our transactions are very low and the creation of a new transaction only lasts a couple of seconds.

8.2 Discussion

The protocol as introduced in this thesis is a first approach to combining auditability and anonimity in a permissioned decentralised payment scheme. Even though all initial requirements are satisfied there is still room for improvement. In this section we briefly discuss the parts of our solution that we feel could be improved.

Security of our protocol is proven extensively in the appendix. However, the security could still be improved. The anonimity of our protocol is not quantum-proof or information-theoretically secure. This implies that a great increase in computational capacity of adversaries anonimity cannot always be guaranteed. Moreover, not all building blocks in our protocol are universally composable. Even though unlikely, this could lead to some compromises in the proved security. Lastly, the used verifiable encryption scheme SAVER is only $\mathrm{IND\text{-}CPA}$ and not $\mathrm{IND\text{-}CCA}$ secure. In the unlikely case that an adversary manages to let the judges decrypt a lot of chosen ciphertexts, this might compromise the security.

A disadvantage of our scheme is the common reference string that is created for our zk-SNARK scheme. This common reference string is the weak spot of security of this scheme. Knowledge of the generation parameters for this common reference string would render the created zero-knowledge proofs completely useless. It is therefore important that this common reference string is created honestly by multiple parties, and that the used randomness never shared or simply destroyed. The reliance on this common reference string is thus not ideal, but relying on a zk-SNARK scheme without such a string would significantly decrease performance. Additionally, we feel that in our use case the honest generation of this common reference string and destroying the randomness could be achieved more easily and is more trustworthy.

Furthermore, the computational efficiency of our scheme especially with regards to the sender of a transaction could be improved upon. We showed in Section 7.4 that the transaction creation and verification times are suitable for use in practice. However, the creation time of a transaction is rather large. Most of this time is spent on generating a zero-knowledge proof. We would like to see the proof generation time decrease significantly, without giving in on the security of the proofs. Further research into other schemes for generating zero-knowledge proofs could possibly help to improve this.

The size of our SAVER ciphertext in a *Transfer* transaction is also a bit larger than desirable with approximately 1.8kB. We would like to decrease this to well under 1kB. However, as their currently does not exist a SNARK-friendly verifiable encryption scheme with smaller proof size this is not possible at the moment. It is the hope of the author that such a scheme will be constructed in the future, as it seems rather unlikely that the same functionality can be achieved without SNARK-friendly verifiable encryption.

Finally, one might wonder whether a blockchain, or distributed ledger, is a strict requirement for our scheme. The short answer would be: no. However, the long answer is a bit more complex. Namely, using a distributed ledger does simplify things in the case of multiple administrators that do not want to or are not allowed to cooperate. This might even be quite likely in the administrators of our case: financial institutions. A distributed ledger can be combined with a suitable consensus mechanism to resolve cooperation issues. This also allows for dividing the verification process amongst multiple parties in order to increase efficiency. Moreover, the use of a distributed ledger generates distributed control. This might increase the trust of regular users in the security of the scheme and the honesty of the administrators. In conclusion, a distributed ledger is not strictly necessary for our scheme but is advised due to the advantages it provides.

8.3 Future work

Throughout the process of this thesis and after obtaining the results, we also constructed some suggestions for future work. The first of these suggestions would be building a post-quantum secure version of this protocol. This would require special focus on the used zk-SNARK scheme as it is currently not post-quantum secure. The adaptation of other primitives should be more straightforward.

On a related note it would also be a good idea to look into simulation-extractable SNARKs. The security proofs for our scheme only hold separately and it would be a great improvement to use universal composability to ensure that our system is secure in general. Simulation-extractable SNARKs are a rather new and interesting way to ensure that zk-SNARKs can also be used in such proofs.

The final security improvement that could be improved in future work is applying a transformation on the SAVER encryption scheme to obtain $\mathrm{IND\text{-}CCA}$ security. One could for example look into the Fujisaki-Okamoto Transformation [43] to achieve this stronger notion of security.

We also have two more suggestions regarding functionality of our payment scheme. The first one is the implementation of anonymous multi- or threshold-signatures. Currently, this is a significant feature in many cryptocurrencies and also certain banks allow for shared accounts. The inclusion of this kind of signatures would be an interesting step towards a more mature payment scheme.

In this thesis we did not yet really touch upon the use of timelocks. Our last suggestion actually concerns the use of timelocks for improving transaction speed. Timelocks could for example be used in realising intermediate off-chain transaction in a Lightning Network-like fashion. Future research into the use of timelocks in a similar fashion as for example the Lightning Network is needed to see what the possibilities are.

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008, online article. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] W. Simons, “OKEx Korea schrapt privacymunten van beurs: Monero, Zcash en Dash,” Sep. 2019. [Online]. Available: https://bitcoinmagazine.nl/2019/09/okex-korea-privacymunten-verbod/

[3] RTLZ, “ING-ceo Hamers: Facebook riskeert ban door banken dankzij libra-plannen,” *RTLZ*, Oct. 2019. [Online]. Available: https://www.rtlz.nl/tech/artikel/4893111/facebook-libra-ban-banken-crypto-ing-ralph-hamers

[4] FATF, “Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers,” 2019, report. [Online]. Available: www.fatf-gafi.org/publications/fatfrecommendations/documents/Guidance-RBA-virtual-assets.html

[5] ——, “International Standards on Combating Money Laundering and the Financing of Terrorism & Proliferation,” 2012, report. [Online]. Available: http://www.fatf-gafi.org/recommendations.html

[6] D. Chaum, “Project Page: DigiCash.” [Online]. Available: https://chaum.com/ecash/

[7] ——, “Blind Signatures for Untraceable Payments,” in *Advances in Cryptology*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Boston, MA: Springer US, 1983, pp. 199–203.

[8] T. Sander and A. Ta-Shma, “Auditable, Anonymous Electronic Cash,” in *Advances in Cryptology — CRYPTO’ 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Berlin, Heidelberg: Springer, 1999, pp. 555–572.

[9] “GNU Taler.” [Online]. Available: https://taler.net/en/index.html

[10] F. Dold, “The GNU Taler system : practical and provably secure electronic payments,” Ph.D. dissertation, l’université de rennes, Feb. 2019.

[11] K. M. Alonso and koe, “Zero to Monero: First Edition,” Jun. 2018, online article. [Online]. Available: https://web.getmonero.org/it/library/Zero-to-Monero-1-0-0.pdf

[12] CryptoRekt, “Blackpaper Verge Currency,” Jan. 2019, blackpaper. [Online]. Available: https://vergecurrency.com/static/blackpaper/verge-blackpaper-v5.0.pdf

[13] “mimblewimble/grin,” Oct. 2019, original-date: 2016-10-21T00:02:23Z. [Online]. Available: https://github.com/mimblewimble/grin

[14] E. Fujisaki and K. Suzuki, “Traceable Ring Signature,” in *Public Key Cryptography – PKC 2007*, ser. Lecture Notes in Computer Science, T. Okamoto and X. Wang, Eds. Berlin, Heidelberg: Springer, 2007, pp. 181–200.

[15] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 397–411.

[16] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized Anonymous Payments from Bitcoin,” in *2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014, pp. 459–474.

[17] J. Poon and T. Dryja, “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments,” Jan. 2016, online article. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[18] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS ’12*. ACM Press, 2012, pp. 326–349.

[19] “ZKProof Community Reference – Version 0.1.x,” Oct. 2019, original-date: 2019-05-29T19:18:12Z. [Online]. Available: https://github.com/zkpstandard/zkreference

[20] J. Groth, “On the Size of Pairing-Based Non-interactive Arguments,” in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, vol. 9666, pp. 305–326.

[21] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly Practical Verifiable Computation,” in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 238–252, iSSN: 1081-6011.

[22] J. Lee, J. Choi, J. Kim, and H. Oh, “SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization,” IACR, Tech. Rep. 1270, 2019. [Online]. Available: https://eprint.iacr.org/2019/1270

[23] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, “Key-Privacy in Public-Key Encryption,” in *Advances in Cryptology — ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Berlin, Heidelberg: Springer, 2001, pp. 566–582.

[24] N. van Saberhagen, “CryptoNote v 2.0,” Oct. 2013, cryptoNote. [Online]. Available: https://cryptonote.org/whitepaper.pdf

[25] “r/Monero - The Strange Birth & History of Monero (4 part series),” 2017. [Online]. Available: https://www.reddit.com/r/Monero/comments/6ud6wh/the_strange_birth_history_of_monero_4_part_series/

[26] “The Bytecoin Scam - A Continuation.” [Online]. Available: https://bitcointalk.org/index.php?topic=4508322.0

[27] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: the second-generation onion router,” in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, ser. SSYM’04. USA: USENIX Association, Aug. 2004, p. 21.

[28] A. Poelstra, “Mimblewimble,” Oct. 2016, online article. [Online]. Available: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf

[29] “BEAM | Mimblewimble-based Privacy Coin.” [Online]. Available: https://beam.mw/

[30] “Zcoin (XZC) Private Financial Transactions Enabled by the Sigma Protocol.” [Online]. Available: https://zcoin.io/

[31] T. Ruffing, S. A. Thyagarajan, V. Ronge, and D. Schroder, “Burning Zerocoins for Fun and for Profit - A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol,” in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, Jun. 2018, pp. 116–119.

[32] S. Bowe, T. Hornby, and N. Wilcox, “Zcash Protocol Specification,” Feb. 2020. [Online]. Available: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf

[33] “PIVX.” [Online]. Available: https://pivx.org/

[34] “Tezos.com.” [Online]. Available: https://tezos.com/

[35] D. Khovratovich and M. Vladimirov, “Full privacy in account-based cryptocurrencies v 0.12,” Jul. 2019, online draft. [Online]. Available: https://dusk.network/uploads/Private-account-based-model.pdf

[36] N. P. Smart, *Cryptography made simple*, ser. Information security and cryptography. Springer, 2016, oCLC: 934618628.

[37] T. Beyne, A. Canteaut, I. Dinur, M. Eichlseder, G. Leander, G. Leurent, M. Naya-Plasencia, L. Perrin, Y. Sasaki, Y. Todo, and F. Wiemer, “Out of Oddity – New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems,” in *Advances in Cryptology – CRYPTO 2020*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 299–328.

[38] E. Tromer, “[Sapling] specify Pedersen hashes for a collision-resistant hash function inside the SNARK · Issue #2234 · zcash/zcash,” Apr. 2017. [Online]. Available: https://github.com/zcash/zcash/issues/2234#issuecomment-292419085

[39] I. Liusvaara and S. Josefsson, “Edwards-Curve Digital Signature Algorithm (EdDSA),” library Catalog: tools.ietf.org. [Online]. Available: https://tools.ietf.org/html/rfc8032

[40] P. Wuille, “Dealing with malleability,” Mar. 2014, bIP 62. [Online]. Available: https://github.com/bitcoin/bips

[41] D. R. L. Brown, “Standards for Efficient Cryptograhpy 1 (SEC 1),” May 2009. [Online]. Available: https://www.secg.org/sec1-v2.pdf

[42] “Zero-Knowledge Proofs | What are they, how do they work, and are they fast yet?” [Online]. Available: http://zkp.science/

[43] E. Fujisaki and T. Okamoto, “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” in *Advances in Cryptology — CRYPTO’ 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Berlin, Heidelberg: Springer, 1999, pp. 537–554.

[44] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, “Exploring the Attack Surface of Blockchain: A Systematic Overview,” *arXiv:1904.03487 [cs]*, Apr. 2019, arXiv: 1904.03487.

[45] F. Tschorsch and B. Scheuermann, “Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies,” *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[46] S. Goldwasser, S. Micali, and C. Rackoff, “The Knowledge Complexity of Interactive Proof Systems,” *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, Feb. 1989.

[47] A. Fiat and A. Shamir, “How To Prove Yourself: Practical Solutions to Identification and Signature Problems,” in *Advances in Cryptology — CRYPTO’ 86*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed. Springer Berlin Heidelberg, 1987, pp. 186–194.

Appendix A

Building Blocks

This chapter of the appendix provides more details to several building blocks that are used in the digital payment scheme as describe in this report. We begin with discussing distributed ledgers and cryptocurrencies in order to provide some more background to the general concept of the payment scheme. Subsequently, some more background on zero-knowledge proofs and quadratic arithmetic programs (QAPs) is given. In the last section we present some theory on commitment schemes and presents an algebraic commitment scheme known as the Pedersen Commitment.

A.1 Distributed ledgers and cryptocurrencies

A distributed ledger is a technology that allows for a consensus on shared data in a decentralised manner. In our application the data consists of financial transactions, and the ledger thus contains a history of all transactions of all users. Probably the most popular way to implement a distributed ledger is by means of a blockchain, as known from the Bitcoin project [1]. For most decentralised currencies the blockchain is open to anyone, we call this a permissionless blockchain.

In a **permissionless** blockchain anyone can join the network, participate in it, and read or write to the network. There is no verification of someone's identity or intentions. This blockchain setting is used in many cryptocurrencies such as Bitcoin and ZCash.

On the contrary, **permissioned** blockchains require anyone who wants to join the network to get approved by the administrators of the network. This option is mostly used by organisations and enterprises, where it is unwanted for just anyone to read from or write to the blockchain.

A permissioned network does however not always have to defined so strictly that you can either do absolutely nothing with the blockchain, or everything. One might for example want to design a network where everyone can view the blockchain, but only a certain set of authorised identities can write to it. This is still considered a permissioned network, even though it is only partly restricted. [44]

We will now shift our focus to a particular usage of distributed ledger technologies, namely virtual currencies. A virtual currency as a form of digital payment has gotten increased popularity and usage in the past decade. The best way to explain how these so called cryptocurrencies works is by explaining it via Bitcoin, as it was the first widely adopted implementation thereof. Bitcoin [1], [44], [45] is a decentralised, i.e. not governed by a single entity or small group of entities, virtual currency. This decentralisation is achieved by making use of a distributed ledger, frequently called blockchain. Before describing how such a blockchain works, we will firstly describe the setup in which Bitcoin can be used.

Suppose Alice wants to transfer a (digital) coin to Bob, she can do this by generating a signed statement stating “I, Alice, transfer 1 coin to Bob.”. She then announces this publicly, i.e. to everyone else in the community that uses the coin, and therefore everyone agrees that Bob got a coin from Alice. In Bitcoin, such a signed contract is called a transaction. The statement contains the address of the sender (Alice), the address of the receiver (Bob), the amount of the currency that is being transferred, and a signature from Alice. This signature is computed using the public-private key pair of Alice, such that everyone in the community can verify that Alice approved this particular transaction, and that no one has altered it. Everyone in the community can store this transaction in his or her copy of the (decentralised) ledger. There is however a subtlety that is missing in this approach. It might be possible for someone to replay this message. Does this mean that Alice wants to send the same amount of currency to Bob again, or is it a malicious replay?

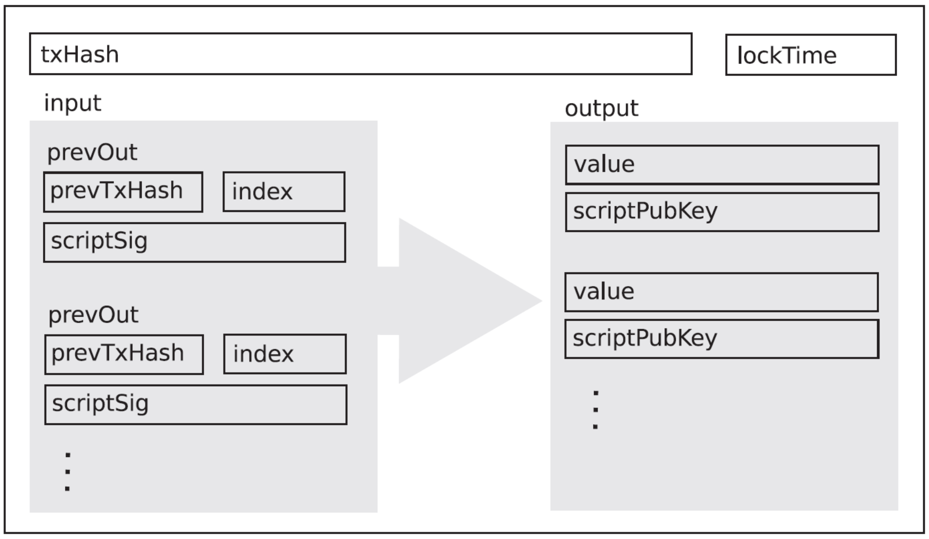


Figure A.1: Schematic depiction of a Bitcoin transaction [45].

In order to prevent this Bitcoin made its transactions a little bit more complex, as can be seen in Figure A.1. Each transaction (`tx`) has a transaction hash `txHash`, which is a hash of all the information in the transaction, and several inputs and outputs. Each input links to the output of a previous transaction by means of the previous transaction hash `prevTxHash` and the index of the output in that previous transaction `index`. Furthermore each input has a signature `scriptSig` which proves that the sender, or set of senders in the case of a multi-signature (multisig) transaction, actually wants to send the money. `lockTime` can be used to specify a timelock, i.e. make the output value only usable from a certain moment in the future. Each output has a value `value`, i.e. the amount of currency, and an address of the receiver of the money `scriptPubKey`. This address might be different for every output. It should always hold that the sum of the input values is larger than or equal to the sum of the output values. The difference between the two is a transaction fee. Once a transaction is published, the peers in the community should check if the transaction is correct, i.e. the hash is correct, the signatures are good, the values add up, and the inputs have not yet been spent before. If all this is the case, they should add the transaction to their own copy of the ledger. This last check is necessary to prevent so called double-spending.

Double-spending a cryptocurrency means spending the same coin or using the same transaction multiple times. With a physical currency this is very hard, because copying a physical banknote is extremely difficult, however digital currencies are rather easy to copy. Fortunately, most cryptocurrencies have a lot of measures in place to prevent this from happening, such as signing your transaction, as well as other nodes checking if a transaction has not been used before. But now the question rises, how do the participants structure the connections between and state of transaction inputs and outputs in a meaningful way?

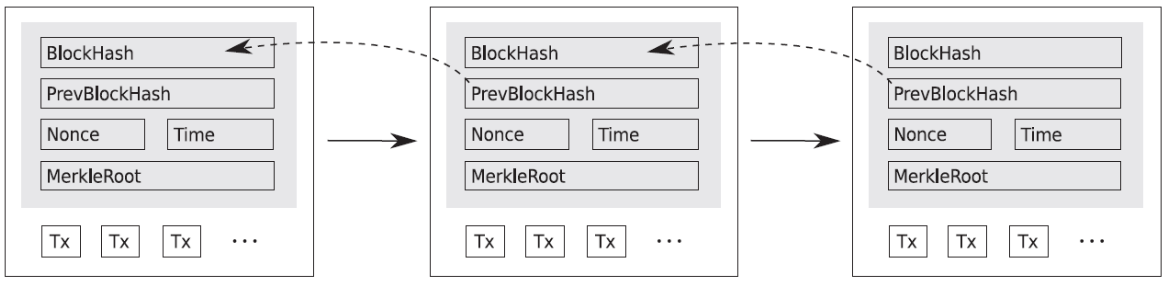


Figure A.2: Schematic (and simplified) depiction of a blockchain [45].

The blockchain was introduced to solve this problem. Simply said, a blockchain aggregates transactions in blocks, whereas these respective blocks together form one big chain representing the ledger of this particular currency. A simplified version of such a blockchain is shown in Figure A.2. A block in a blockchain contains several values. Firstly, it has a `BlockHash`, which is a hash over all values in the block, ensuring integrity. For Bitcoin, a hash should be smaller than a certain number. In order to achieve this, every participant in the network keeps trying different values for the `Nonce` until this is achieved. When this is achieved he or she publicly announces the block as is, and everyone in the network can verify the block and come to the consensus that it should be added to the block chain ledger. They then start to collect new transactions and build a new block. In Bitcoin the transactions are aggregated per 10 minutes.

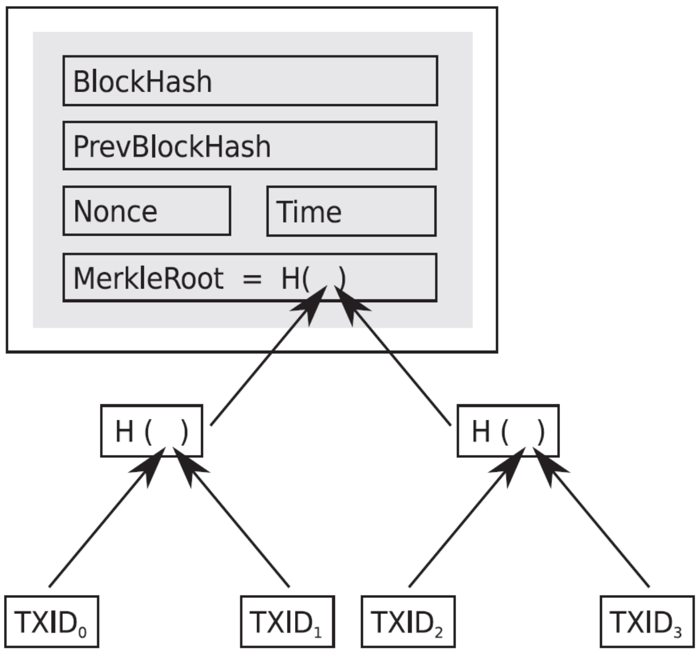


Figure A.3: Schematic (and simplified) depiction of a small Merkle Tree [45].

A block contains several more important values, among others it contains a pointer to the previous block, `PrevBlockHash` and a `time` at which it was announced. Moreover, it contains the transaction details in the form of a Merkle Tree, as shown in Figure A.3. In order to make finding a transaction more easy, every transaction contains a reference `MerkleRoot`, which points to the root of Merkle Hash Tree. But how exactly do peers in the network keep their ledgers consistent with one another?

A new block is added to chain when consensus has been reached. In case of ambiguity, or multiple branches on a chain, the participants select the longest chain of blocks. This method prevents most attacks, since constructing one new block is very time consuming. Namely, one needs to find the right value of the `Nonce`, such that the `BlockHash` value is low enough. Finding this value takes a lot of time, but verifying is relatively simple. This is also the reason why this method is called Proof-of-Work: finding a new block consumes a lot of electricity and therefore money. It is thus economically irrational to try and create a long false chain of blocks, since the costs outweigh the benefits. For completeness, an overview of the entire Bitcoin Protocol is shown in Figure A.4.

The workings of **Bitcoin** [1] as explained above, make it rather clear that it was invented for the sole purpose of transferring value, with the possible addition of some really simple extra requirements on the money by the means of small scripts. Moreover, Bitcoin also provides a weak form of anonymity, through the use of pseudonyms which cannot be directly related to a specific electronic device or person. However, with a bit of analysis it becomes rather easy to trace a subset of transactions back to a single person. Similar to Bitcoin, most of the cryptocurrencies currently out there also have the same goal and level of anonymity, though there is a set of exceptions.

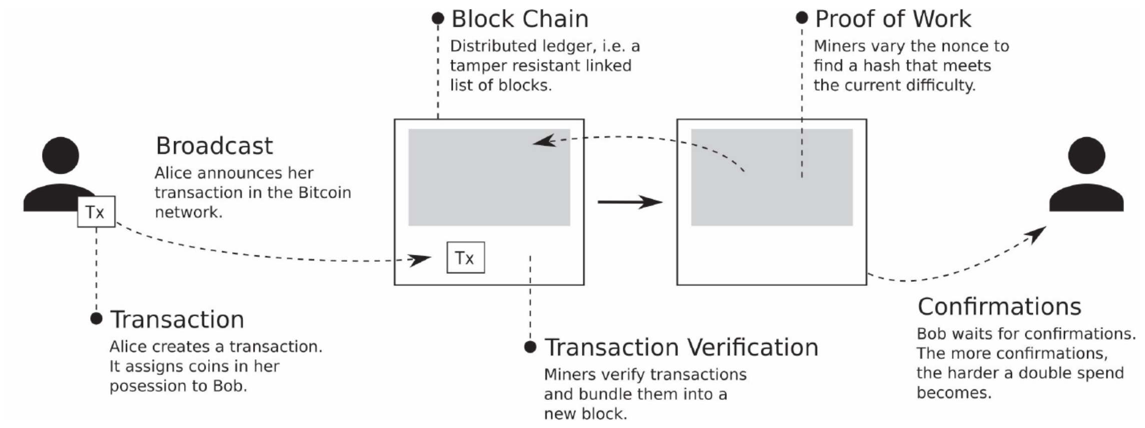

Block Chain
Distributed ledger, i.e. a tamper resistant linked list of blocks.
Proof of Work
Miners vary the nonce to find a hash that meets the current difficulty.
Broadcast
Alice announces her transaction in the Bitcoin network.
Tx
Tx
Transaction
Alice creates a transaction. It assigns coins in her posession to Bob.
Transaction Verification
Miners verify transactions and bundle them into a new block.
Confirmations
Bob waits for confirmations. The more confirmations, the harder a double spend becomes.

Figure A.4: Steps in the Bitcoin Protocol [45].

A.2 Proving of Knowledge

One of the arguments that is generally used for showing that a certain language L is in NP is showing the existence of a *proof system* for that language. Such a proof system can be visualised as a prover (not computationally bounded), who supplies a certain certificate, or proof, π on input x, which attests to the fact that $x \in L$. The verifier should then be able to verify, in polynomial time, with help of the proof π that indeed $x \in L$. Please note that the prover should only be able to verify such a proof π when $x \in L$. A generalisation of such interactive proof systems is provided by the class IP, i.e. the class of languages for which such interactive proof systems exist. It has later been proven that $\mathrm{IP} = \mathrm{PSPACE}$, and thus $\mathrm{NP} \subseteq \mathrm{IP}$. This implies that interactive proof systems exist for all problems in NP, including NP-complete problems. The two required (informal) properties for an interactive proof system are the following:

- **Correctness.** A prover that knows the witness for a given true statement should be able to (efficiently) convince the verifier thereof;
- **Knowledge soundness.** A prover that does not know the witness to a given statement should not be able (with a more than negligible probability) to convince the verifier that he or she does know the witness.

These definitions state explicitly that the prover should (or should not) know a witness. This is needed for the proof to be a proof of knowledge. If the prover does not know a witness, he or she might still be able to proof the correctness of the statement, we call this a proof of membership. An example stating this difference is that of a proof of knowledge of a hash pre-image, i.e. a proof that the prover knows the input to a hash function for a given hash. This is different from the proof of membership that would only proof that a given hash has a pre-image, which for most hash functions is a trivial statement.

Zero-knowledge proofs

In 1989 Goldwasser, Micali, and Rackoff [46] introduced the privacy-preserving proof of knowledge technique, otherwise known as zero-knowledge proofs. These zero-knowledge proofs enable a prover to proof to a verifier his or her knowledge about a witness that will evaluate a given statement to true, without revealing this witness. These zero-knowledge proofs are interactive proofs with one extra requirement. They should in addition to completeness and soundness also satisfy the following (informal) definition:

- **Zero-Knowledgeness.** The proof should not reveal anything other than the fact that the prover has a witness for the given statement, it should especially not reveal any information about the witness.

The addition of this extra property leads to the definition of a new complexity space CZK (Computational Zero-Knowledge). It has been proven that $\mathrm{NP} \subseteq \mathrm{CZK}$. Moreover we know that $\mathrm{CZK} = \mathrm{IP} = \mathrm{PSPACE}$, if one-way functions exist. This fact implies that we can use these proofs to ensure, in zero-knowledge, that someone knows the solution to a problem that is hard, e.g. NP-complete, to calculate. An example of this is proving knowledge of the discrete logarithm, or prime factorisation of a large composite number, which would be computationally infeasible to calculate. A verifiable proof of knowledge implies that someone actually constructed this group element, or composite number. Zero-knowledgeness implies that the constructor, or prover, can prove that he or she constructed the value without revealing how. Usefulness of this comes from the fact that the discrete logarithm or prime factorisation is generally used in cryptographic protocols, such as asymmetric cryptography.

As a side remark, we note that a zero-knowledge proof, or ZKP, can be either transferable or deniable. A transferable proof can be passed on to other verifiers who can also verify that the original prover actually had knowledge of the correct witness. This setting is convenient in a distributed ledger settings, since the prover wants everyone to be able to verify his or her statement. In some cases this might be unwanted, in the sense that a prover wants to proof to a single verifier his or her knowledge, and later plausibly deny that he or she constructed this proof. Such a proof is called deniable.

One of the problems with standard (interactive) zero-knowledge proofs is the amount of communication that is necessary for a single proof. Therefore we want to transform an interactive zero-knowledge proof into a non-interactive zero-knowledge proof, or NIZK. One of the most popular and early techniques to do this is the Fiat-Shamir heuristic [47]. It should be noted that this technique can only be used to transform public-coin protocols. In the article, the authors propose the use of a non-interactive random oracle in order to remove communication. However, since these are uninstantiable, they are replaced by cryptographic hash functions in practical applications.

A.3 Arithmetic circuits and QAPs

An $\mathbb{F}$-arithmetic circuit is a circuit consisting of only addition and multiplication gates for which the inputs are all elements of the finite field $\mathbb{F}$. Each gate has exactly two input wires and one output wire. Some of the input wires are part of the statement, the public inputs, the other wires are all part of the witness, the auxiliary variables. When all wire values, or variables, are chosen properly, i.e. all values fit the gates, we consider the circuit to be satisfied.

A circuit can also be depicted by a set of quadratic arithmetic constraints. We define one input variable as $a_0 = 1$, used for representing a constant wire value, and denote the other variables, the wire values, as $a_1, \dots, a_m$, where m is the number of non-constant wires. The arithmetic circuit that belongs to the relation R can be represented by a set of n equations of the form

$$\sum_i a_i u_{i,q} \cdot \sum_i a_i v_{i,q} = \sum_i a_i w_{i,q},$$

where $u_{i,q}, v_{i,q}, w_{i,q}$ are the constants (in $\mathbb{F}$) of the qth equation.

The multiplication and addition gates that are used in an $\mathbb{F}$-arithmetic circuit can easily be mapped to these equations. A multiplication gate with input a_i, a_j and output a_k, can be modelled with one constraint as $a_i \cdot a_j = k$. An addition gate does not require an extra constraint, due to the sums in the constraints. For example, if

$a_i + a_j = a_k$ and $a_k \cdot a_l = a_m$, we model this with one constraint as $(a_i + a_j) \cdot a_l = a_m$ and neglect the variable a_k.

These quadratic constraints can be transformed in an Quadratic Arithmetic Program (QAP), given that $\mathbb{F}$ contains sufficient unique elements. Given n constraints, as defined above, we pick n distinct values $r_1, \dots, r_n \in \mathbb{F}$ arbitrarily, and define our target polynomial as $t(x) = \prod_q (x - r_q)$. Furthermore, we define $u_i(x), v_i(x), w_i(x)$ to be the $n-1$ degree $n-1$ polynomials such that

$$u_i(r_q) = u_{i,q}, v_i(r_q) = v_{i,q}, w_i(r_q) = w_{i,q}, \text{ for all } 0 \leq i \leq m, 1 \leq q \leq n,$$

by means of Lagrange interpolation. We now have that our previous n constraints can be rewritten to the following n equations in the points $\{r_q\}_{q=1}^{n}$:

$$\sum_i a_i u_i(r_q) \cdot \sum_i a_i v_i(r_q) = \sum_i a_i w_i(r_q).$$

Since, $t(X)$ is the lowest degree monomial with $t(r_q) = 0$ for all r_q, we can reformulate these equations into one big QAP as

$$\sum_i a_i u_i(X) \cdot \sum_i a_i v_i(X) \equiv \sum_i a_i w_i(X) \mod t(X).$$

In this QAP $a_1, \dots, a_\ell \in \mathbb{F}$ are the public inputs and $a_{\ell+1}, \dots, a_m \mathbb{F}$ are the auxiliary inputs. [20], [21]

A.4 Commitments

Suppose Alice wants to play “rock-paper-scissors(-lizard-spock)” over the phone with Bob. When playing this game in real-life, Alice and Bob just make and show their choice simultaneously. However on the phone they have to decide who goes first. There is one problem with this, if Alice (resp. Bob) goes first, Bob (resp. Alice) can just make such a choice that he (resp. she) always wins, since he (resp. she) can adapt to Alice’s (resp. Bob’s) choice. A way to solve this problem is by means of a commitment, i.e. both Alice and Bob tell one another a commitment to their choice, in any order, which hides their choice. They then reveal how they made this commitment, and can figure out who won. Of course, neither player should be able to change their original decision, thus a commitment needs to be binding. In short, a proper commitment scheme $c = C(x, r)$ should satisfy the following the definitions [36]. In these definitions $\mathbb{P}$ denotes the plaintext domain, and $\mathbb{R}$ the randomness domain, i.e. $C(\cdot, \cdot)$ is the commitment function, that takes plaintext x, and randomness r as input, and outputs commitment c.

Definition 1 (Binding). *A commitment C is said to be information-theoretically (resp. computationally) binding if no infinitely powerful (resp. computationally bounded) adversary can win the binding game (with a non-negligible probability):*

- *The adversary outputs values $x \in \mathbb{P}$, $r \in \mathbb{R}$.*
- *The adversary wins the game if she is able to output values $x' \in \mathbb{P} : x' \neq x$, and $r' \in \mathbb{R}$, such that $C(x, r) = C(x', r')$.*

Definition 2 (Hiding). *A commitment scheme C is said to be information-theoretically (resp. computationally) hiding if no infinitely powerful (resp. computationally bounded) adversary can win the following game (with a non-negligible advantage):*

- *The adversary outputs two messages $x_0, x_1 \in \mathbb{P}$ of equal length.*
- *The challenger generates $r \in_R \mathbb{R}$ and a bit $b \in_R \{0, 1\}$.*
- *The challenger computes $c = C(x_b, r)$, and sends c to the adversary.*
- *The adversary wins the game if she correctly guesses the bit b.*

The advantage of the adversary is defined as

$$\textit{Advantage} = 2 \left| \mathbb{P}\left[\textit{Adversary wins the hiding game}\right] - \frac{1}{2} \right|.$$

We should note that there exists no commitment scheme that is both information-theoretically binding as well as information-theoretically hiding. One of the most used, additively homomorphic, commitment schemes is known as the Pedersen commitment. This scheme is computationally binding (under the discrete-log assumption) and information-theoretically hiding. It is defined, for a finite abelian group $G = \langle g \rangle$ of prime order and $h \in G$, as $C(x, r) = h^x g^r$.

Appendix B

Security games and definitions

In this appendix chapter we list the formal definitions of the existing cryptographic building blocks that we use. Some security games, that play a part in our proofs in Appendix C, will also be described. An informal description of all building blocks and their definitions is already provided in Chapter 2.

zk-SNARK.[1] There are four definitions that a zk-SNARK needs to adhere to: completeness, succinctness, knowledge soundness, and zero-knowledgeness. Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable binary relation R. For any pair $(x, a) \in R$ we call x the statement and a the witness. We define $\mathcal{R}_\lambda$ to be the set of possible relations R, that the relation generator may output given 1^λ. $\mathcal{R}$ might output some additional information z, which will be given to the adversary. A zk-SNARK for $\mathcal{R}$ is a tuple of probabilistic polynomial algorithms $(\mathrm{Setup}, \mathrm{Prove}, \mathrm{Verify}, \mathrm{Sim})$ such that:

- $(\mathrm{crs}, \tau) \leftarrow \mathrm{Setup}(R)$: The setup produces a common reference string crs and a simulation trapdoor τ for the relation R;
- $\pi \leftarrow \mathrm{Prove}(R, \mathrm{crs}, x, a)$: The prover takes as input the common reference string crs, statement-witness pair $(x, a) \in R$ and returns an argument (or proof) π;
- $0/1 \leftarrow \mathrm{Verify}(R, \mathrm{crs}, x, a)$: The verification algorithm takes as input the common reference string crs, statement x, and argument π and returns 0 (reject) or 1 (accept);
- $\pi \leftarrow \mathrm{Sim}(R, \tau, x)$: The simulator takes as input the simulation trapdoor τ and statement x and returns an argument π.

A zk-SNARK should satisfy the definitions below. We give the 'computational' forms of the definitions, they can be easily extended to the 'perfect' forms by replacing $\approx$ by $=$.

[1] We take the definition from [20] and [18].

Definition 3 (Completeness)**.** *Completeness implies that, given any true statements, an honest prover should be able to convince an honest verifier. In other words: for all* $\lambda \in \mathbb{N}, R \in \mathcal{R}_\lambda, (x,a) \in R$*:*

$$\mathbb{P}\left[(crs, \tau) \leftarrow \mathit{Setup}(R); \pi \leftarrow \mathit{Prove}(R, crs, x, a) : \mathit{Verify}(R, crs, x, \pi) = 1\right] \approx 1.$$

Definition 4 (Zero-knowledgeness)**.** *An argument is zero-knowledge if it does not leak any information besides the truth of the statement. We say that* $(\mathit{Setup}, \mathit{Prove}, \mathit{Verify}, \mathit{Sim})$ *is zero-knowledge if for all* $\lambda \in \mathbb{N}, (R, z) \leftarrow \mathcal{R}(1^\lambda), (x,a) \in R$ *and all adversaries* $\mathcal{A}$*:*

$$\begin{aligned} &\mathbb{P}\left[(crs, \tau) \leftarrow \mathit{Setup}(R); \pi \mathit{Prove}(R, crs, x, a) : \mathcal{A}(R, z, crs, \tau, \pi) = 1\right] \\ \approx\ &\mathbb{P}\left[(crs, \tau) \leftarrow \mathit{Sim}(R); \pi \mathit{Prove}(R, crs, x) : \mathcal{A}(R, z, crs, \tau, \pi) = 1\right]. \end{aligned}$$

Definition 5 (Knowledge soundness)**.** *We call* $(\mathit{Setup}, \mathit{Prove}, \mathit{Verify}, \mathit{Sim})$ *an argument of knowledge if there is an extractor* $\mathcal{E}$ *that can compute a witness whenever the adversary produces a valid argument. The extractor gets full access to the adversary's state, including any random coins. Formally, we require that for all non-uniform polynomial time adversaries* $\mathcal{A}$ *there exists a non-uniform polynomial time extractor* $\mathcal{E}$ *such that:*

$$\mathbb{P}\left[\begin{array}{c} (R, z) \leftarrow \mathcal{R}(1^\lambda); (crs, \tau) \leftarrow \mathit{Setup}(R); ((x, \pi); w) \leftarrow (\mathcal{A}\|\mathcal{E})(R, z, crs) : \\ (x, a) \notin R \text{ and } \mathit{Verify}(R, crs, a, \pi) = 1 \end{array}\right] \approx 0.$$

Definition 6 (Succinctness)**.** *We call* $(\mathit{Setup}, \mathit{Prove}, \mathit{Verify}, \mathit{Sim})$ *a succinct argument if the proof size and verification time are scale poly-logarithmically in the statement. Let* $y := (M, x, t)$ *be the instance,* $|w| \le t$*, and* M *the Turing machine that accepts* (x, a) *after at most* t *steps. Given* $\pi \leftarrow \mathit{Prove}(R, crs, x, a), R \in \mathcal{R}_\lambda, (x,a) \in R$*, both the length of* π *and the running time of* $\mathit{Verify}(R, crs, x, \pi)$ *should be bounded by*

$$p(k + |M| + |x| + \log t),$$

where p *is a universal polynomial that does not depend on* $\mathcal{R}$*.*

Commitment schemes. There are two definitions that a commitment scheme needs to have: binding and hiding. Below, we repeat the definitions form Appendix A.4 for easy of use. For more information on commitment schemes we refer the reader to that section of the appendix.

Definition 7 (Binding)**.** *A commitment* C *is said to be information-theoretically (resp. computationally) binding if no infinitely powerful (resp. computationally bounded) adversary can win the binding game (with a non-negligible probability):*

- *The adversary outputs values $x \in \mathbb{P}$, $r \in \mathbb{R}$.*
- *The adversary wins the game if she is able to output values $x' \in \mathbb{P} : x' \neq x$, and $r' \in \mathbb{R}$, such that $C(x,r) = C(x',r')$.*

Definition 8 (Hiding)**.** *A commitment scheme C is said to be information-theoretically (resp. computationally) hiding if no infinitely powerful (resp. computationally bounded) adversary can win the following game (with a non-negligible advantage):*

- *The adversary outputs two messages $x_0, x_1 \in \mathbb{P}$ of equal length.*
- *The challenger generates $r \in_R \mathbb{R}$ and a bit $b \in_R \{0,1\}$.*
- *The challenger computes $c = C(x_b, r)$, and sends c to the adversary.*
- *The adversary wins the game if she correctly guesses the bit b.*

The advantage of the adversary is defined as

$$\textit{Advantage} = 2 \left| \mathbb{P}\left[\textit{Adversary wins the hiding game}\right] - \frac{1}{2} \right| .$$

Hash function. In our protocol we often use hash functions. Those hash functions are required to be collision-resistant. Below we give the definition thereof.

Definition 9 (Collision-resistance)**.** *A hash function H is collision resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage in the following game is negligible:*

- *The adversary is given an instantiation of the hash function H should output a pair x and x'.*
- *The adversary wins the game if $x \neq x'$ and $H(x) = H(x')$.*

Pseudo-random functions. These functions form a big part of our protocol. Often, we require them to be collision-resistant next to being a proper PRF. The definition for collision-resistance is the same as that for hash functions. A definition for secure PRF's, that includes the PRF game is given below.

Definition 10 (Secure PRF)**.** *A function family $\{F_k\}_K : D \to C$, with key space K is a secure PRF-family if no probabilistic polynomial-time adversary $\mathcal{A}$ has a non-negligible advantage in the following game:*

- *The challenger randomly selects a bit $b \in_R \{0,1\}$.*
- *The adversary is allowed to query, as many times as desired, an oracle $\mathcal{O}_{F_k}$ that on input $x \in D$ returns the following value for y:(1) if x was seen before, return the same value y as before; (2) if $b = 0$ then $y \in_R C$; or (3) if $b = 1$ then return $y \leftarrow F_k(x)$.*

- *The adversary ends the game by returning a bit guess b'.*
- *If $b' = b$ the adversary wins the game.*

The advantage of the adversary is defined as

$$\textit{Advantage} = 2\left|\mathbb{P}\left[\textit{Adversary wins the } PRF \textit{ game.}\right] - \frac{1}{2}\right|.$$

Signature scheme. Below, we give a definition for a $\mathrm{SUF\text{-}CMA}$ secure signature scheme. Remember that a signature scheme has the following functions:

- $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}()$: The key generator returns a public pk and private sk key pair.
- $\sigma \leftarrow \mathrm{Sign}_{\mathrm{sk}}(m)$: The signature algorithm returns a signature σ, when given a message m ans signature key sk.
- $0/1 \leftarrow \mathrm{Verify}_{\mathrm{pk}}(\sigma, m)$: The verification algorithm returns a 0 (accept) or 1 (reject), when given a verification key pk, signature σ, and message m.

Definition 11 ($\mathrm{SUF\text{-}CMA}$)**.** *We call a signature scheme $SUF\text{-}CMA$ secure if no probabilistic polynomial-time adversary can win the following game with non-negligible probability:*

- *The challenger generates a key pair $(pk, sk) \leftarrow KeyGen()$ and sends pk to the adversary.*
- *The adversary is allowed to sign arbitrary messages m, as many times as desired, using the signature oracle $\mathcal{O}_{sig}$, that on input m returns $\sigma \leftarrow Sign_{sk}(m)$ and stores m.*
- *The adversary ends the game by returning a message-signature pair (m, σ).*
- *The adversary wins the game if $Verify_{pk}(\sigma, m) = 1$ and (m, σ) has not been seen before.*

In our proofs we also use the $\mathrm{SUF\text{-}1CMA}$ definition. The only difference with $\mathrm{SUF\text{-}CMA}$ security is that the adversary is only allowed to call $\mathcal{O}_{\mathrm{sig}}$ once.

Encryption scheme. Below, we give a definition for a $\mathrm{IND\text{-}CCA}$ and $\mathrm{IK\text{-}CCA}$ secure asymmetric encryption scheme. In these definitions we also give the $\mathrm{IND\text{-}CCA}$ and $\mathrm{IK\text{-}CCA}$ game definitions. Remember that an asymmetric encryption scheme is comprised of the following algorithms:

- $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}()$: The key generator returns a public pk and private sk key pair.

- $c \leftarrow \mathrm{Enc}_{\mathrm{pk}}(m)$: The encryption algorithm returns a ciphertext c, when given a plaintext message m and encryption key pk.
- $m \leftarrow \mathrm{Dec}_{\mathrm{sk}}(c)$: The decryption algorithm returns the original plaintext m, when given a decryption key sk and ciphertext c.

Definition 12 (IND-CCA)**.** *We call an encryption scheme $IND\text{-}CCA$ secure, if no probabilistic polynomial-time adversary can win the following game with a non-negligible probability:*

- *The challenger generates a key pair $(pk, sk) \leftarrow KeyGen()$ and sends pk to the adversary.*
- *The adversary outputs two messages m_0, m_1.*
- *The challenger generates a random bit b and returns the ciphertext $c^* \leftarrow Enc_{pk}(m_b)$.*
- *The adversary is allowed to decrypt arbitrary ciphertexts $c \neq c^*$, using the oracle $\mathcal{O}^{dec}$, that on input $c \neq c^*$ returns $m \leftarrow Dec_{sk}(c)$.*
- *The adversary ends the game by returning a bit guess b'.*
- *The adversary wins the game if $b' = b$.*

The advantage of the adversary is defined as

$$\textit{Advantage} = 2\left|\mathbb{P}\left[\textit{Adversary wins the } IND\text{-}CCA \textit{ game.}\right] - \frac{1}{2}\right|.$$

Definition 13 (IK-CCA)**.** *We call an encryption scheme $IK\text{-}CCA$ secure, if no probabilistic polynomial-time adversary can win the following game with a non-negligible probability:*

- *The challenger generates two key pair $(pk_0, sk_0) \leftarrow KeyGen()$ and $(pk_1, sk_1) \leftarrow KeyGen()$ and sends pk_0 and pk_1 to the adversary.*
- *The adversary outputs a messages m.*
- *The challenger generates a random bit b and returns the ciphertext $c^* \leftarrow Enc_{pk_b}(m)$.*
- *The adversary is allowed to decrypt arbitrary ciphertexts $c \neq c^*$, using the oracle $\mathcal{O}_0^{dec}$ respectively $\mathcal{O}_1^{dec}$, that on input $c \neq c^*$ returns $m \leftarrow Dec_{sk_0}(c)$ respectively $m \leftarrow Dec_{sk_0}(c)$.*
- *The adversary ends the game by returning a bit guess b'.*

- *The adversary wins the game if $b' = b$.*

The advantage of the adversary is defined as

$$\textit{Advantage } = 2\left|\mathbb{P}\left[\textit{Adversary wins the } IND\text{-}CCA \textit{ game.}\right] - \frac{1}{2}\right|.$$

Appendix C

Security proofs

Below, we will present the definition of the several security aspects as translations of the requirements. Moreover we will prove that our payment scheme Π satisfies these security definitions one by one. From now on Π will be used to denote the following set of functions $(\texttt{Setup}, \texttt{CreateAccount}, \texttt{AddAccount}, \texttt{RevokeAccount}, \texttt{ConvertToNote}, \texttt{ConvertFromNote}, \texttt{CreateTransaction}, \texttt{VerifyTransaction}, \texttt{ReceiveTransaction})$ that together form the payment scheme as defined in this document. We note that all the proofs as presented below, only hold separately since not all of our building blocks our universally composable. On the other hand, we have no reason to assume that this results in any practical threat against the security of our scheme.

We first present the *completeness*, *ledger indistinguishability*, *transaction non-malleability*, and *balance* proofs and definitions. These proofs and definitions are very similar to the ones provided in the original Zerocash publication [16]. However, we do fix some issues and inconsistencies and work the proofs out in more detail. The subsequent proofs and definitions for *spend limit*, *access control*, *accountability*, and *timelock* are all completely original and were, as far as the author knows, first introduced in this thesis.

Throughout these proofs we will refer to five different types of transactions published on the blockchain, see also Section 5.1. We briefly repeat the types below to improve legibility:

- *Transfer*: A regular note-to-note transaction, produced by `CreateTransaction`;
- *NewAccount*: New credential commitment that is added to the Credential Merkle tree, produced by `AddAccount`;
- *RvkAccount*: Credential commitment that is removed from the Credential Merkle tree, produced by `RevokeAccount`;

- *ConvertTo*: Conversion to anonymous note from fiat currency, produced by `ConvertToNote`;
- *ConvertFrom*: Conversion from anonymous note to fiat currency, produced by `ConvertFromNote`.

C.1 Completeness

To show that Π is *complete*, we show that no polynomial-sized adversary $\mathcal{A}$ has a non-negligible advantage in the completeness game COMP. First we give the definition of completeness, we then describe the completeness game and end with a proof that shows that our scheme is complete.

Definition 14 (Completeness). *We call a payment scheme Π complete if, for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv_{\Pi,\mathcal{A}}^{COMP}(\lambda) < negl(\lambda)$, with $Adv_{\Pi,\mathcal{A}}^{COMP}(\lambda) := \mathbb{P}\left[COMP(\Pi, \mathcal{A}, \lambda) = 1\right]$ being $\mathcal{A}$'s advantage in the completeness game.*

Game (COMP)**.** The definition of the completeness game COMP is as follows. Given a scheme Π, an adversary $\mathcal{A}$, and a security parameter λ, the game consists of interactions between $\mathcal{A}$ and a challenger $\mathcal{C}$. This interaction terminates with an output bit from $\mathcal{C}$. Finally, let $L_{\leq \mathrm{tx}}$ denote the ledger containing tx and all transaction that were posted before tx. At the start of the game the challenger performs the setup of Π, i.e. he computes $\mathrm{pp} :=$ **Setup**(λ), and sends pp to $\mathcal{A}$. Subsequently, $\mathcal{A}$ sends a ledger L, one pair of admin credentials $\mathrm{cred}_{\mathsf{adm}}$ and a proposed transaction tx to $\mathcal{C}$. This transaction tx can be one of the five types as defined in the start of this chapter. The proposed transaction contains the following values, depending on the type:

- *Transfer*: input note $\mathrm{note}^{\mathsf{old}}$, input memory cell $\mathrm{mem}^{\mathsf{old}}$, ceiling memory cell $\mathrm{mem}^{\mathsf{ceil}}$, sender credentials cred, new note value $v_{\mathsf{note}}^{\mathsf{new}}$, receiver credentials $\mathrm{cred}^{\mathsf{new}}$, info string info, boolean value for including SAVER encrypted values b_{saver}, lock time new note t_{δ};
- *NewAccount*: secret address key $\mathrm{sk}_{\mathsf{addr}}$;
- *RvkAccount*: credential commitment $\mathrm{cm}_{\mathsf{cred}}$;
- *ConvertTo*: new note value $v_{\mathsf{note}}^{\mathsf{new}}$, receiver credentials cred, info string info;
- *ConvertFrom*: input note $\mathrm{note}^{\mathsf{old}}$;

On receiving pp, L, $\mathrm{cred}_{\mathsf{adm}}$ and tx, $\mathcal{C}$ first checks the valid construction of pp and $\mathrm{cred}_{\mathsf{adm}}$. Depending on the type of tx, $\mathcal{C}$ does the following additional checks:

- *Transfer* (1) The input note $\mathrm{note}^{\mathsf{old}}$, input memory cell $\mathrm{mem}^{\mathsf{old}}$, and ceiling memory cell $\mathrm{mem}^{\mathsf{ceil}}$ are well-formed or are $\perp$; (2) Not more than one of $\mathrm{note}^{\mathsf{old}}$ and $\mathrm{mem}^{\mathsf{old}}$ equals $\perp$; (3) If $\mathrm{mem}^{\mathsf{old}} = \perp$ then $\mathrm{mem}^{\mathsf{ceil}} = \perp$; (4) The sender and receiver credentials cred and $\mathrm{cred}^{\mathrm{new}}$ are well-formed; (5) The commitment to the input note appears in a transaction tx^* in L; (6) tx^* is well-formed and $\mathrm{note}^{\mathsf{old}} = \mathtt{ReceiveTransaction}(\mathrm{pp}, \mathrm{tx}^*, \mathrm{cred}, L_{\leq \mathrm{tx}^*})$; (7) The balance equation is satisfied $v^{\mathsf{new}}_{\mathsf{note}} \leq v^{\mathsf{old}}_{\mathsf{note}} + v^{\mathsf{old}}_{\mathsf{mem}}$; (8) $\mathrm{mem}^{\mathsf{ceil}}$ is old enough, i.e. $t^{\mathsf{ceil}}_{\mathsf{mem}} < t^{\mathsf{new}} - T$[1], or $\mathrm{mem}^{\mathsf{ceil}} = \perp$, or $b_{\mathsf{saver}} = 1$; (9) The transaction limit is satisfied: $c^{\mathsf{new}} - c^{\mathsf{ceil}} \leq L$ or $b_{\mathsf{saver}} = 1$; (10) The input note is ‘unlocked’: $t^{\mathsf{old}}_{\mathsf{note}} + t^{\mathsf{old}}_{\delta} \leq t^{\mathsf{new}}$; and (11) The input note $\mathrm{note}^{\mathsf{old}}$ and input memory cell $\mathrm{mem}^{\mathsf{old}}$ have not yet been used as an input in any transaction on L[2]. This can be checked using $\mathcal{E}$, the zk-SNARK extractor for $\mathcal{A}$, to compute witnesses for all transactions on L that contain the nullifier for the input note or memory cell. If the extractor fails to output a valid witness for any of the transactions on L, $\mathcal{C}$ aborts and outputs 1;

- *ConvertFrom* (1) The input note $\mathrm{note}^{\mathsf{old}}$ is well-formed; (2) The value t_δ of the input note $\mathrm{note}^{\mathsf{old}}$ is equal to 0; and (3) The input note $\mathrm{note}^{\mathsf{old}}$ has not yet been used as an input in any transaction on L. This can again be checked using $\mathcal{E}$, the zk-SNARK extractor for $\mathcal{A}$, to compute witnesses for all transactions on L that contain the nullifier for the input note. If the extractor fails to output a valid witness for any of the transactions on L, $\mathcal{C}$ aborts and outputs 1;

- For any of the other transaction $\mathcal{C}$ performs no additional checks.

If any of the above tests fail, $\mathcal{C}$ aborts and outputs 0. Else $\mathcal{C}$ computes $\mathrm{rt}_{\mathsf{cred}}$, $\mathrm{rt}_{\mathsf{note}}$, and $\mathrm{rt}_{\mathsf{mem}}$ over all valid transactions on L, i.e. for which `VerifyTransaction` returns `true`. If necessary, $\mathcal{C}$ also obtains the transaction time t^{new}, and computes $v^{\mathsf{new}}_{\mathsf{mem}}$ in such a way that the balance equation is fully satisfied. Subsequently $\mathcal{C}$ uses the algorithm corresponding to the transaction type to compute the transaction tx as proposed by $\mathcal{A}$. For the input values $\mathcal{C}$ uses the values received from $\mathcal{A}$ and if necessary the just computed values. Finally, $\mathcal{C}$ outputs 1 if and only if any of the following conditions hold, otherwise $\mathcal{C}$ outputs 0:

- tx is not well-formed, **Verifytransaction**$(\mathrm{pp}, \mathrm{tx}, L)$ does not return `true`;

- In the case of a *Transfer* transaction, **ReceiveTransaction**$(\mathrm{pp}, \mathrm{tx}, \mathrm{cred}^{\mathsf{new}}, L)$ does not return $\mathrm{note}^{\mathsf{new}}$.

[1]T is the fixed time limit as introduced in Section 4.7.
[2]L is the fixed spend limit as introduced in Section 4.7.

Remark. In the last case, we do not need to check if the receiver is able to spend this newly received note. For otherwise, $\mathcal{A}$ would then have added the proposed transaction to the ledger itself and asked $\mathcal{C}$ to spend the newly received note, and thus winning the game with larger probability.

Theorem 1. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The zk-SNARK scheme has computational knowledge soundness;
- PRF is a collision-resistant function;
- All used primitives, PRF, COMM, CRH, and the zk-SNARK scheme, satisfy their respective completeness properties;

is complete.

Proof. To prove the theorem we make a case distinction based on the transaction type. But before we do so, we note that the probability that $\mathcal{A}$ wins this game by failure of the zk-SNARK extractor $\mathcal{E}$ is negligible in λ. This follows directly from the knowledge soundness of our zk-SNARK scheme. Thus, it only remains to show that probability with which $\mathcal{A}$ wins the game when $\mathcal{E}$ does not fail is negligible. We do this be showing per case that the probability that $\mathcal{A}$ wins is negligible.

Transfer. The computed transaction tx has the following form $(\mathrm{rt}_{\mathsf{mem}}, \mathrm{rt}_{\mathsf{note}}, \mathrm{rt}_{\mathsf{cred}}, k, \kappa, \eta, \mu, \mathrm{cm}^{\mathsf{new}}_{\mathsf{note}}, \mathrm{cm}^{\mathsf{new}}_{\mathsf{mem}}, t^{\mathsf{new}}, \pi_{\mathtt{XFER}}, \mathrm{data}^{\mathsf{new}}_{\mathsf{note}}, \mathrm{data}^{\mathsf{new}}_{\mathsf{mem}}, \mathrm{data}_{\mathsf{saver}}, \mathrm{pk}_{\mathsf{sig}}, \sigma_{\mathsf{xfer}})$. We will first show that tx is well-formed. The Merkle roots rt_* are trivially well-formed, and computed from the same state of the ledger. The correctness of k follows from the correctness of $\mathrm{CRH}^{\mathsf{sig}}$. Similarly, the correctness of κ, η, μ follows from the correctness of PRF. Moreover, the correctness of both output commitments $\mathrm{cm}^{\mathsf{new}}_*$ follows directly from that of their respective commitment schemes COMM^*. Moreover, $\mathcal{C}$ chooses t^{new} on its own and simply selects the correct current time.

The correctness of the above discussed public inputs, combined with the verified assumptions, lead us to conclude that the zero-knowledge proof π is valid as well. Moreover, the correctness of the transaction signature follows directly from the correctness of the used signature scheme. Similarly, the enclosed data in both $\mathrm{data}^{\mathsf{new}}_*$ fields can also be decrypted correctly given the secret key, due to correctness of the encryption scheme and the verified correctness of cred. Analogously, due to the correctness of the SAVER scheme $\mathrm{data}_{\mathsf{saver}}$ can also be decrypted using the SAVER secret and verification key.

From the above argumentation and the initial checks performed by $\mathcal{C}$, we see that the only way left in which $\mathcal{A}$ wins the game is the option that either η or μ appears in a valid transaction tx' on L. Therefore, it remains to show that this probability is negligible in λ. To the contrary, suppose that there exists a valid transaction tx' on L with $\eta' = \eta$, then either one of two cases holds: (1) $\eta' = \mathrm{PRF}^{\eta}_{\mathrm{sk}'_{\text{addr}}}(\mathrm{pos}'_{\text{note}})$; or (2) $\eta' \neq \mathrm{PRF}^{\eta}_{\mathrm{sk}'_{\text{addr}}}(\mathrm{pos}'_{\text{note}})$. Firstly, we know by the initial checks that $\mathrm{note}^{\text{old}}$ has not been spent yet, implying that $\mathrm{pos}'_{\text{note}} \neq \mathrm{pos}_{\text{note}}$. Therefore, case (1) implies that a collision in PRF has been found. Because PRF is collision-resistant this can only happen with a probability negligible in λ. Case (2) implies that the (knowledge) soundness property of the zk-SNARK scheme has been broken, however the used zk-SNARK scheme has computational (knowledge) soundness this can happen with negligible probability. Thus we can conclude that only with negligible probability, $\eta' = \eta$. The proof for negligible occurrence of the event $\mu' = \mu$ is analogous.

From the above argumentation it follows that $\mathcal{A}$ cannot win this case with a non-negligible advantage.

ConvertFrom. The computed transaction tx has the following form $(\mathrm{rt}_{\text{note}}, \eta, \mathrm{pk}_{\text{addr}}, \pi_{\texttt{CFROM}}, \sigma_{\text{cfrom}})$. We will show that tx is well-formed. The Merkle root $\mathrm{rt}_{\text{note}}$ is trivially well-formed. The correctness of η follows from the correctness of PRF and the fact that $\mathrm{pk}_{\text{addr}}$ is a valid admin public address key stems from the initial checks. This fact, together with the initial checks ensures us that $\pi_{\texttt{CFROM}}$ will also verify correctly. Next to this, the validness of the signature follows form the correctness of the signature scheme. The only way in which $\mathcal{A}$ can now the game is the option that η already appears on another transaction tx' on L. From a proof that is identical to the previous case we conclude that this can only happen with negligible probability.

Other. For all other transactions it is easy to check that the fact that $\mathcal{A}$ can only win with negligible probability, follows directly from the correctness of the building blocks and the initial checks performed by $\mathcal{C}$.

From the above case distinction and our initial argument it follows that $\mathcal{A}$ cannot win the COMP game with a non-negligible advantage. □

C.2 Payment oracle

For the remaining security proofs we require an oracle that allows for simulation of honest users of our payment scheme Π. To be more precise, an adversary $\mathcal{A}$ should be able to send queries to this oracle. As response to these queries the oracle should either elicit behaviour of honest users and administrators on the blockchain, or insert transactions of users and administrators under $\mathcal{A}$'s control.

The payment scheme oracle $\mathcal{O}^{\mathsf{pay}}$ is initialised with public parameters pp and one pair of honest admin credentials $\mathrm{cred}_{\mathrm{adm}}$ constructed by a run of **Setup**. In reality, there could me multiple honest admins, however for the proofs it suffices to simulate all these admins as one entity with one key pair. Next to this, the oracle stores five data structures, all of which are initially empty:

- a ledger L;
- a list of admin credentials under $\mathcal{A}$'s control ADMIN;
- a list of credentials ADDR;
- a list of memory cells MEM;
- a list of old memory cells $\mathrm{MEM\text{-}OLD}$
- a list of notes NOTE.

The oracle $\mathcal{O}^{\mathsf{pay}}$ accepts several different query types Q[3], to which we describe the responses below. All queries, apart from **Insert(admin)**, elicit behaviour of honest users that call the algorithm from Π with the same name. An **Insert** query is used to insert any transaction that was created by a user under $\mathcal{A}$'s control. The **InsertAdmin** transaction is used to add a new administrator to the system with credentials $\mathrm{cred}_{\mathsf{adm}}$[4] who is under control of the adversary $\mathcal{A}$. Note that, because the honest admin owns at least one of the shares of the SAVER secret key $\mathrm{sk}_{\mathsf{svr}}$, $\mathcal{A}$ cannot decrypt any of the SAVER ciphertexts $\mathrm{data}_{\mathsf{saver}}$.

- Q = (**InsertAdmin**, $\mathrm{cred}_{\mathrm{adm}}$)
  1. Verify that $\mathrm{cred}_{\mathrm{adm}}$ is a valid set of administrator credentials, else abort.
  2. Add $\mathrm{cred}_{\mathsf{adm}}$ to ADMIN.
- Q = (**CreateAccount**)
  1. Compute $(\mathrm{cred}, \mathrm{tx}_{\mathsf{cred}}) :=$ **AddAccount** $\circ$ **CreateAccount**(pp).
  2. Add cred to ADDR.
  3. Add $\mathrm{tx}_{\mathsf{cred}}$ to L.
- Q = (**RevokeAccount**, $\mathrm{pk}_{\mathsf{addr}}$)
  1. Compute $\mathrm{rt}_{\mathsf{cred}}$.
  2. Compute $\mathrm{tx}_{\mathsf{rvk}} :=$ **RevokeAccount**$(\mathrm{pk}_{\mathsf{addr}}, \mathrm{rt}_{\mathsf{cred}}, \mathrm{cred}_{\mathsf{adm}})$.

[3]The first entry of a query tuple is its type, the entries after that are input values.

[4]These credentials can be generated using the steps of the **AddAdmin** algorithm.

3. Remove cred belonging to $\mathrm{pk}_{\mathsf{addr}}$ from ADDR.
4. Add $\mathrm{tx}_{\mathsf{rvk}}$ to L.

- Q = (**ConvertToNote**, $v^{\mathsf{new}}_{\mathsf{note}}$, $\mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}$, $\mathrm{pk}^{\mathsf{new}}_{\mathsf{enc}}$, info)
  1. Compute $(\mathrm{note}^{\mathsf{new}}, \mathrm{tx}_{\mathsf{cto}}) :=$ **ConvertToNote**$(\mathrm{pp}, \mathrm{v}^{\mathsf{new}}_{\mathsf{note}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{enc}}, \mathrm{info}, \mathrm{cred}_{\mathsf{adm}})$.
  2. Add $\mathrm{note}^{\mathsf{new}}$ to NOTE.
  3. Add $\mathrm{tx}_{\mathrm{cto}}$ to L.

- Q = (**ConvertFromNote**, $\mathrm{idx}^{\mathsf{old}}$)
  1. Let $\mathrm{cm}^{\mathsf{old}}$ be the $\mathrm{idx}^{\mathsf{old}}$-th note commitment on L.
  2. Let $\mathrm{note}^{\mathsf{old}}$ be the first note on NOTE with commitment $\mathrm{cm}^{\mathsf{old}}$.
  3. Verify that the public address of $\mathrm{note}^{\mathsf{old}}$ is that in $\mathrm{cred}_{\mathsf{adm}}$, else abort.
  4. Compute $\mathrm{rt}_{\mathsf{note}}$.
  5. Compute $\mathrm{tx}_{\mathsf{cfrom}} :=$ **ConvertFromNote**$(\mathrm{pp}, \mathrm{note}^{\mathsf{old}}, \mathrm{rt}_{\mathsf{note}}, \mathrm{cred}_{\mathsf{adm}})$.
  6. Verify that **VerifyTransaction**$(\mathrm{pp}, \mathrm{tx}_{\mathsf{cfrom}}, L)$ returns true, else abort.
  7. Remove $\mathrm{note}^{\mathsf{old}}$ from NOTE.
  8. Add $\mathrm{tx}_{\mathsf{cfrom}}$ to L.

- Q = (**CreateTransaction**, $\mathrm{idx}^{\mathsf{old}}$, $\mathrm{pk}^{\mathsf{old}}_{\mathsf{addr}}$, $\mathrm{v}^{\mathsf{new}}_{\mathsf{note}}$, $\mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}$, $\mathrm{pk}^{\mathsf{new}}_{\mathsf{enc}}$, info, b_{saver}, t_δ)[5]
  1. Compute $\mathrm{rt}_{\mathsf{cred}}$, $\mathrm{rt}_{\mathsf{note}}$, and $\mathrm{rt}_{\mathsf{mem}}$.
  2. Let $\mathrm{cm}^{\mathrm{old}}$ be the $\mathrm{idx}^{\mathsf{old}}$-th note commitment on L.
  3. Let $\mathrm{note}^{\mathsf{old}}$ be the first note on NOTE with commitment $\mathrm{cm}^{\mathsf{old}}$.
  4. Verify that $\mathrm{pk}^{\mathsf{old}}_{\mathsf{addr}}$ is the address used in $\mathrm{note}^{\mathsf{old}}$, else abort.
  5. Let cred be the first entry in ADDR containing $\mathrm{pk}^{\mathsf{old}}_{\mathsf{addr}}$.
  6. Let $\mathrm{mem}^{\mathsf{old}}$ be the memory cell belonging to cred in MEM.
  7. Let t^{new} be the new block time.
  8. Let $\mathrm{mem}^{\mathsf{ceil}}$ be the last memory cell in $\mathrm{MEM\text{-}OLD}$ beloning to cred with $t^{\mathsf{ceil}}_{\mathsf{mem}} < t^{\mathsf{new}} - T$.
  9. Compute $(\mathrm{note}^{\mathsf{new}}, \mathrm{mem}^{\mathsf{new}}, \mathrm{tx}) :=$ **CreateTransaction**$(\mathrm{pp}, \mathrm{cred}, \mathrm{rt}_{\mathsf{note}}, \mathrm{rt}_{\mathsf{mem}}, \mathrm{rt}_{\mathsf{cred}}, \mathrm{note}^{\mathsf{old}}, \mathrm{mem}^{\mathsf{old}}, \mathrm{mem}^{\mathsf{ceil}}, v^{\mathsf{new}}_{\mathsf{new}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}, \mathrm{pk}^{\mathsf{new}}_{\mathsf{enc}}, \mathrm{info}, t^{\mathsf{new}}, b_{\mathsf{saver}}, t_\delta)$;
  10. Verify that **VerifyTransaction**$(\mathrm{pp}, \mathrm{tx}, L)$ returns true, else abort.
  11. Remove $\mathrm{note}^{\mathsf{old}}$ from NOTE.

[5] $\mathrm{idx}^{\mathsf{old}}$ could be $\perp$, in case of no input note.

12. Add $\mathrm{note}^{\text{new}}$ to NOTE.
13. Replace $\mathrm{mem}^{\text{old}}$ by $\mathrm{mem}^{\text{new}}$ on MEM.
14. Add $\mathrm{mem}^{\text{new}}$ to $\mathrm{MEM\text{-}OLD}$.
15. Add tx to L.

- Q = (**Insert**, tx)

  1. Verify that **VerifyTransaction**$(\mathrm{pp}, \mathrm{tx}, \mathrm{L})$ returns true, else abort.
  2. Add tx to L.
  3. Run **ReceiveTransaction**$(\mathrm{pp}, \mathrm{tx}, \mathrm{cred}, L)$ for all cred on ADDR, and add any returned note $\mathrm{note}^{\text{new}}$ to NOTE.

C.3 Ledger indistinguishability

In this section, we first give the definition of Ledger Indistinguishability and present the $\mathrm{L\text{-}IND}$ game. We also define what *public consistency* means in the $\mathrm{L\text{-}IND}$ game. Finally, we define a simulation of the $\mathrm{L\text{-}IND}$ game and use this simulation to proof that our scheme satisfies the ledger indistinguishability property.

Definition 15 (Ledger indistinguishability)**.** *We say that a payment scheme Π satisfies the ledger indistinguishability property if, for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv^{L\text{-}IND}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$, with $Adv^{L\text{-}IND}_{\Pi,\mathcal{A}} := 2 \cdot \mathbb{P}\left[L\text{-}IND(\Pi, \mathcal{A}, \lambda = 1\right] - 1.$*

Game ($\mathrm{L\text{-}IND}$)**.** The Ledger Indistinguishability game is defined as follows. Given a scheme Π, adversary $\mathcal{A}$, and security parameter λ, the game consists of interactions between $\mathcal{A}$ and a challenger $\mathcal{C}$. The game terminates with an output bit from $\mathcal{C}$. At the start of the game, the challenger randomly samples $b \in_R \{0, 1\}$, computes $\mathrm{pp} := \mathtt{Setup}(\lambda)$, initialises two separate oracles $\mathcal{O}^{\text{pay}}_0$ and $\mathcal{O}^{\text{pay}}_1$, and sends pp to $\mathcal{A}$. The protocol then proceeds in steps, where in each step $\mathcal{C}$ provides $\mathcal{A}$ with two ledgers $(L_{\text{left}}, L_{\text{right}}) := (L_b, L_{1-b})$, where L_i is the current ledger state of oracle $\mathcal{O}^{\text{pay}}_i$. Subsequently, $\mathcal{A}$ sends a pair of queries (Q, Q'), which must be of the same type. Depending on the type of query, the challenger $\mathcal{C}$ does the following:

- If the query type is **Insert(Admin)**, $\mathcal{C}$ forwards Q to $\mathcal{O}^{\text{pay}}_b$ and Q' to $\mathcal{O}^{\text{pay}}_{1-b}$. In the case of **InsertAdmin** $\mathcal{C}$ first checks if Q and Q' are *publicly consistent* (explained below). This allows $\mathcal{A}$ to insert transactions produced by users or administrators controlled by $\mathcal{A}$ directly into $(L_{\text{left}}, L_{\text{right}})$.
- If the query type is <u>not</u> **Insert(Admin)**, $\mathcal{C}$ forwards Q to $\mathcal{O}^{\text{pay}}_0$ and Q' to $\mathcal{O}^{\text{pay}}_1$. However, before doing this $\mathcal{C}$ checks if Q and Q' are *publicly consistent*. $\mathcal{A}$

uses these types of queries to simulate behaviour of honest users and honest administrators. Since $\mathcal{A}$ does not know the bit b he does not know whether the resulting transaction of query Q can be found on ledger L_{left} or L_{right} (and vice versa for Q').

$\mathcal{A}$ can also decide to end the game by sending $b' \in \{0, 1\}$ which is a guess of the bit b, instead of sending a pair of queries. If $b' = b$, $\mathcal{C}$ aborts and outputs 1, else $\mathcal{C}$ outputs 0.

Definition 16 (Public consistency)**.** *A pair of queries (Q, Q') is publicly consistent if they satisfy the requirements as set out below. The first and foremost requirement being that the queries must both have the same type. Depending on the type of query, there might be more requirements:*

- *If the query is of type **InsertAdmin** we require that $cred_{\textit{adm}}$ is equal in Q and Q';*
- *If the query is of type **CreateAccount** we require that both oracles generate the same address.*
- *If the query is of type **RevokeAccount** we require that the public address key in both queries is the same.*
- *If the query is of type **ConvertToNote** we require that the public information (with respect to $\mathcal{A}$) is consistent: (1) If $pk_{\textit{addr}}^{\textit{new}}$ does not appear on $ADDR$ in Q, then it should equal $pk_{\textit{addr}}^{\textit{new}}$ in Q' (and vice versa for Q') (2) If $pk_{\textit{addr}}^{\textit{new}}$ does not appear on $ADDR$ in Q, then $info$ and $v_{\textit{note}}^{\textit{new}}$ are equal in both Q and Q';*
- *If the query is of type **ConvertFromNote** we require that the input is well-formed: (1) The note commitment referenced by $idx^{\textit{old}}$ appears on $NOTE$; (2) $idx^{\textit{old}}$ refers to a note that is owned by the admin (3) The input note was never locked: $t_{\delta}^{\textit{old}} = 0$ for both Q and Q'.*
- *If the query is of type **CreateTransaction** we require that the input is well-formed: (1) The note commitment referenced by $idx^{\textit{old}}$ appears on $NOTE$ or equals $\perp$; (2) $idx^{\textit{old}}$ refers to a note that is owned by $pk_{\textit{addr}}$; (3) The balance equation is satisfied $v_{\textit{old}}^{\textit{note}} + v_{\textit{old}}^{\textit{mem}} >= v_{\textit{new}}^{\textit{note}}$.*

  *Moreover, the public information (with respect to $\mathcal{A}$) must be consistent and the transaction must succeed: (1) If $idx^{\textit{old}} = \perp$ in Q then $idx^{\textit{old}} = \perp$ in Q' (vice versa for Q'); (2) If $idx^{\textit{old}} \neq \perp$ in Q, then if $note^{\textit{old}}$ was inserted with an **Insert** query, $v_{\textit{note}}^{\textit{old}}$ should be equal on Q and Q' and $note^{\textit{old}}$ in Q' should also be inserted with an **Insert** query (vice versa for Q'); (3) If $pk_{\textit{addr}}^{\textit{new}}$ does not appear on $ADDR$ in Q, then it should equal $pk_{\textit{addr}}^{\textit{new}}$ in Q' (and vice versa for Q'); (4) If $pk_{\textit{addr}}^{\textit{new}}$ does not* *appear on $ADDR$ in Q, then $info$, v^{new}_{note}, and t_δ are equal in both Q and Q'; (5) t_{new} is equal in both Q and Q'; (6) If pk^{new}_{addr} is on $ADMIN$ or in the honest admin credentials $cred_{adm}$, then $t_\delta = 0$ for both Q and Q'; (7) The input notes $note^{old}$ in both Q and Q' are both unlocked, i.e. $t^{old}_{note} + t^{old}_\delta \leq t^{new}$; (8) The spend limit is not surpassed for both Q and Q': if $b_{saver} = \mathtt{false}$ then $c^{new} - c^{ceil} \leq L$; (9) If $idx^{old} \neq \perp$ in Q, then if $note^{old}$ was inserted with an **Insert** query, t^{old}_δ should also be equal on Q and Q' and if $t^{old}_\delta \neq 0$, then t^{old}_{note} should be equal on Q and Q' (vice versa for Q').*

We proof that our payment scheme satisfies the ledger indistinguishability property by showing that our game is computationally indistinguishable from a simulation thereof, through a series of hybrid experiments. In other words, we will show that there is a negligible difference between the $\mathrm{L\text{-}IND}$ game and the simulation. We will first define our simulation, which differs from the $\mathrm{L\text{-}IND}$ game in the fact that all ledger entries computed by honest users are replaced with random values independent of the ledger state. This implies that the advantage of $\mathcal{A}$ in distinguishing the ledgers will be 0 in the simulation.

Definition 17 (Simulation)**.** *The simulation starts by sampling a bit $b \in_R \{0, 1\}$. It then computes the setup of our scheme Π, with the adaptation that we now also store the zk-SNARK trapdoor τ of each key pair, to obtain $pp := \mathtt{Setup}(\lambda)$. We then continue as in the $L\text{-}IND$ argument, with $\mathcal{C}$ sending pp to $\mathcal{A}$. The simulation then continues in steps where $\mathcal{A}$ sends a pair of publicly consistent queries (Q, Q') to $\mathcal{C}$ who passes these queries along to the respective oracles. Then $\mathcal{C}$ returns the state of both ledger as (L_{left}, L_{right}). These steps continue until $\mathcal{C}$ receives a bit guess b' from $\mathcal{A}$.*

The adaptation in the above part concerns how the oracles $\mathcal{O}^{pay}$ handle the queries, we will discuss these alterations per query:

- ***InsertAdmin** is not altered.*
- *In **CreateAccount** step 1 is altered as follows. Generate pk_{addr} uniformly at random with the appropriate size, instead of deriving it from the address secret key. We compute the proof π using $Sim(\tau, x)$ instead of $Prove(x)$. Note that we still require that the values generated for Q and Q' are identical.*
- ***RevokeAccount** is not altered.*
- *In **ConvertToNote** step 1 is altered as follows, only if pk^{new}_{addr} appears on $ADDR$. Compute cm^{new}_{note} as a commitment to a random string of the appropriate length. Next to this, the zero-knowledge proof π is computed using $Sim(\tau, x)$. We also replace $data^{new}_{note}$ by an encryption of a random string of the appropriate length*

under a randomly generated key. If $pk_{\text{addr}}^{\text{new}}$ does not appear on $ADDR$ nothing is altered.

- *In **ConvertFromNote** we alter step 5 as follows, η is replaced by a random string of the appropriate length and π is computed using $Sim(\tau, x)$. The other values in x remain the same.*
- *In **CreateTransaction** step 9 is altered in the following ways:*
  - *κ and μ are replaced by (independent) random strings of the appropriate length. Moreover, $cm_{\text{mem}}^{\text{new}}$ is replaced by a commitment to a random string of the appropriate length, and $data_{\text{mem}}^{\text{new}}$ is replaced by an encryption of a random string of the appropriate length under a randomly generated key. $data_{\text{saver}}$ is replaced by an encryption of a random string of the appropriate length under the same key, independent of the value of b_{saver};*
  - *If there is an input note, η is replaced by a random string of the appropriate length;*
  - *If $pk_{\text{addr}}^{\text{new}}$ appears $ADDR$, $cm_{\text{note}}^{\text{new}}$ is replaced by a commitment to a random string of the appropriate length, and $data_{\text{note}}^{\text{new}}$ is replaced by an encryption of a random string of the appropriate length under a randomly generated key;*
  - *The proof π is computed using $Sim(\tau, x)$.*
- ***Insert** is not altered.*

Theorem 2. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The zk-SNARK scheme has perfect zero-knowledgeness;
- The encryption scheme Enc is $\mathrm{IND\text{-}CCA}$ and $\mathrm{IK\text{-}CCA}$ secure;
- The SAVER encryption scheme with encryption function $\mathrm{Enc}^{\text{svr}}$ is $\mathrm{IND\text{-}CPA}$ secure;
- The function family PRF is a pseudo-random function family;
- The commitment scheme COMM is computationally hiding;

satisfies the ledger indistinguishability property.

Proof. To show that L_{left} and L_{right} both reveal nothing about the bit b in the simulation, we only need show that the new entries to the ledger as induced by one pair (Q, Q') reveal no information about b. The identical distribution of the full ledgers then follows from induction and the fact that the query types of Q and Q' are equal.

- **InsertAdmin** and **Insert** both are applied directly to L_{left} and L_{right} and are thus independent of the bit b.
- **CreateAccount** publishes the same $\mathrm{pk}_{\text{addr}}$ value on both ledgers. Next to this both values of $\mathrm{cm}_{\text{cred}}$ are randomly distributed. Therefore, both proofs π have the same distribution. Therefore, our message and thus also signature are distributed identically.
- **RevokeAccount** trivially reveals nothing about b since it has identical input, and thus removes the commitment that is in the same position in both ledgers.
- In the case that $\mathrm{pk}_{\text{addr}}^{\text{new}}$ does not appear on ADDR then **ConvertToNote** receives identical input from both queries (publicly consistent), thus $\mathrm{cm}_{\text{note}}^{\text{new}}$ and $\mathrm{data}_{\text{note}}^{\text{new}}$ are identically distributed on both ledgers. In the case that $\mathrm{pk}_{\text{addr}}^{\text{new}}$ does appear on ADDR we have that $\mathrm{cm}_{\text{note}}^{\text{new}}$ has the same random distribution on both ledgers. Moreover, $\mathrm{data}_{\text{note}}^{\text{new}}$ is an encryption of a random string under a randomly generated key on both ledgers. Therefore, in both cases also π and σ are identically distributed.
- In **ConvertFromNote** $\mathrm{rt}_{\text{note}}$ is computed in the same way on both ledgers, namely given all the previous note commitments on the ledger. This value is thus deterministic and reveals nothing more about b than the note commitments already present on the ledger. Moreover, $\mathrm{pk}_{\text{addr}}$ is identical on both ledgers, and η is chosen uniformly at random. Therefore all public inputs to the proof are identically distributed, giving us that also π and σ are identically distributed.
- Similarly to the previous case, in **CreateTransaction** all Merkle roots rt and t^{new} are deterministic values revealing nothing about b. Since k is computed in the same way on both ledgers, independently of b, k also reveals nothing about b. Next to this κ and μ all have the same random distribution on both ledgers, and η is either the empty nullifier on both ledgers (publicly consistent) or identically and randomly distributed on both ledgers. Moreover, $\mathrm{cm}_{\text{mem}}^{\text{new}}$ is a commitment to randomly generated data on both ledgers and $\mathrm{data}_{\text{mem}}^{\text{new}}$ is an encryption of random data under a randomly generated key, similarly $\mathrm{data}_{\mathrm{saver}}$ is the encryption of a random string independent of the input of the query. Finally, $\mathrm{cm}_{\text{note}}^{\text{new}}$ is a commitment on the same input if $\mathrm{pk}_{\text{addr}}^{\text{new}}$ is not in ADDR (publicly consistent), otherwise it is randomly distributed and the same holds for $\mathrm{data}_{\text{note}}^{\text{new}}$. This leads us to the conclusion that all public inputs for the zk-SNARK proof and signature scheme are identically distributed, and thus π and σ are also identically distributed.

The above case distinction proves that the results of the queries tell $\mathcal{A}$ nothing about the bit b, which leads us to conclude that $\mathcal{A}$'s advantage in the simulation is 0.

It remains to proof that the simulation is indistinguishable from the real L-IND game. As mentioned before we will prove this through a series of hybrid experiments $(G_{\text{real}}, G_1, G_2, G_3, G_4, G_{\text{sim}})$, with $G_{\text{real}} := \text{L-IND}$ and G_{sim} the simulation as described directly above.

Before we continue with setting out these hybrid experiments, we first define some useful notation. Let $\text{Adv}^{\delta}_{G_x,G_y} := |\text{Adv}^{G_x}_{\Pi,\mathcal{A}} - \text{Adv}^{G_y}_{\Pi,\mathcal{A}}|$. Moreover, let n_{CTo} be the amount of valid **ConvertToNote** queries sent by $\mathcal{A}$, n_{CFr} the amount of valid **ConvertFromNote** queries sent by $\mathcal{A}$, n_{Tr} the amount of valid **CreateTransaction** queries sent by $\mathcal{A}$, and n_{Acc} the amount of valid **CreateAccount** queries sent by $\mathcal{A}$.

We now describe the hybrid experiments by listing the modifications with respect to the previous experiment in the series:

- G_1 differs from G_{real} in one way, namely G_1 simulates all zk-SNARK proofs. I.e. it calls Sim instead of Prove in all places where a proof π is constructed. However, since the used zk-SNARK scheme is perfect zero-knowledge, the distribution of a proof π generated by Sim is the same as one by Prove. Therefore we can conclude $\text{Adv}^{\delta}_{G_1,G_{\text{real}}} = 0$.

- G_2 differs from G_1 by replacing the note and memory ciphertexts in *ConvertToNote* and *CreateTransaction* transactions by encryptions of random strings under random keys if the receiver is an honest user. To be precise, we always replace $\text{data}^{\text{new}}_{\text{mem}}$, and we replace $\text{data}^{\text{new}}_{\text{note}}$ only when $\text{pk}^{\text{new}}_{\text{addr}}$ is in ADDR. Replacing happens in two steps: (1) Generate a new encryption key pair $(\text{sk}_{\text{enc}}, \text{pk}_{\text{enc}})$ (2) Sample a random message r (of the right length) uniformly at random and compute the ciphertext $\text{Enc}_{\text{pk}_{\text{enc}}}(r)$. We will proof below that $\text{Adv}^{\delta}_{G_2,G_1}$ is negligible in λ.

- G_3 differs from G_2 by replacing the SAVER ciphertexts in *CreateTransaction* transactions by encryptions of random strings under the same key. Irrespective of the receiver or the value of b_{saver}. Specifically, replacing $\text{data}_{\text{saver}}$ is done by generating a random message r (of the right length) uniformly at random and compute the ciphertext $\text{Enc}^{\text{svr}}_{\text{pk}_{\text{svr}}}(r)$. We will proof below that $\text{Adv}^{\delta}_{G_3,G_2}$ is negligible in λ.

- G_4 differs from G_3 by replacing all PRF outputs by random strings of the same length. More specifically, in **CreateAddress** queries pk_{addr} is replaced by a random string. Moreover, η is replaced by a random string in **ConvertFromNote** queries, and finally η, μ, and κ are all replaced by independent random strings in **CreateTransaction** queries. Below, we will proof that $\text{Adv}^{\delta}_{G_4,G_3}$ is negligible in λ.

- We already gave a complete definition of $G_{\textsf{sim}}$, nevertheless we will describe the differences with G_3 for clarity. The simulation modifies G_3 by replacing the input to the commitment functions by random strings of the same length. To be precise, we replace the input for $\mathrm{cm}_{\textsf{note}}$ in both **ConvertToNote** and **CreateTransaction** queries with a random string, when $\mathrm{pk}^{\textsf{new}}_{\textsf{addr}}$ appears on ADDR. Additionally, the input for $\mathrm{cm}_{\textsf{mem}}$ is replaced by a random string of the same length in **CreateTransaction** queries. The same holds for the input to $\mathrm{cm}_{\textsf{cred}}$ in **CreateCredentials** queries. We will proof below that $\mathrm{Adv}^{\delta}_{G_{\textsf{sim}},G_4}$ is negligible in λ.

From G_1 to G_2. Let $\mathrm{Adv}_{\textsf{enc}}$ be the maximum of any adversary's advantage in either the $\mathrm{IND\text{-}CCA}$ or the $\mathrm{IK\text{-}CCA}$ game for the used encryption scheme. We use another hybrid game H in between G_1 and G_2 to prove that $\mathrm{Adv}^{\delta}_{G_2,G_1}$ is negligible in λ. This game H alters G_1 by encrypting each ciphertext under a randomly generated key, instead of the receiver's public encryption key. G_2 on its turn modifies H by replacing the plaintext that forms the input the encryption function by a random string of the same length.

We first discuss $\mathrm{Adv}^{\delta}_{H,G_1}$. We show how to build an adversary against the $\mathrm{IK\text{-}CCA}$ experiment using $\mathcal{A}$ by letting the challenger for G_1 act as normal, except for the following. The challengers picks some $j \in [n_{\textsf{Acc}}]$ and when $\mathcal{A}$ makes the j-th **CreateAccount** query, the challenger starts the $\mathrm{IK\text{-}CCA}$ game and thus receives two public encryption keys $(\mathrm{pk}^0_{\textsf{enc}}, \mathrm{pk}^1_{\textsf{enc}})$. $\mathcal{C}$ than sets $\mathrm{pk}_{\textsf{enc}} := \mathrm{pk}^0_{\textsf{enc}}$ in this transaction. Moreover, the challenger also picks an index $i \in [n_j]$, with n_j being the number of encryptions made under the public key of the j-th user. i is the index of the ciphertext that should be encrypted using the $\mathrm{IK\text{-}CCA}$ oracle. When $\mathcal{A}$ sends the query of type **ConvertToNote** or **CreateTransaction** that results in the i-th encryption under j's public key, the challenger queries the $\mathrm{IK\text{-}CCA}$ challenger on the plaintext m and receives $c := \mathrm{Enc}_{\mathrm{pk}^{\bar{b}}_{\textsf{enc}}}(m)$, where $\bar{b}$ is the internal bit of the $\mathrm{IK\text{-}CCA}$ challenger. Effectively, we replace the i-th ciphertext for user j, that belongs to the current transaction, by c and continue as normal. When $\mathcal{A}$ ends the experiment with a guess b' we return this same b' as our guess in the $\mathrm{IK\text{-}CCA}$ game. To see that this makes sense, we note that when $\bar{b} = 0$ the view of $\mathcal{A}$ is identical to that in G_1. When $\bar{b} = 1$ the view of $\mathcal{A}$ is identical to that of an intermediate hybrid H' (between G_1 and H) where the key of only on ciphertext is replaced. Thus we get that $\mathrm{Adv}_{\textsf{enc}}$ is an upper bound for $\mathrm{Adv}^{\delta}_{H',G_1}$. Then by a hybrid argument over all $2n_{\textsf{Tr}} + n_{\textsf{CTo}}$ ciphertexts it follows that $\mathrm{Adv}^{\delta}_{H,G_1} \leq (2n_{\textsf{Tr}} + n_{\textsf{CTo}}) \cdot \mathrm{Adv}_{\textsf{enc}}$.

We now discuss $\mathrm{Adv}^{\delta}_{G_2,H}$. The argument is analogous to the one above. The only difference is that for this step we replace the plaintext message m by a random message of the same length, instead of replacing the encryption key. Using an argument analogous to the above reasoning, we can construct an adversary against the $\mathrm{IND\text{-}CCA}$ challenger instead of of the $\mathrm{IK\text{-}CCA}$ challenge, to arrive at the result that $\mathrm{Adv}^{\delta}_{G_2,H} \leq (2n_{\mathsf{Tr}} + n_{\mathsf{CTo}}) \cdot \mathrm{Adv}_{\mathsf{enc}}$.

Finally, we use the triangle inequality to conclude that $\mathrm{Adv}^{\delta}_{G_2,G_1} \leq (4n_{\mathsf{Tr}} + 2n_{\mathsf{CTo}}) \cdot \mathrm{Adv}_{\mathsf{enc}}$, which is negligible in λ.

From G_2 to G_3. Let $\mathrm{Adv}_{\mathsf{enc}}$ be the maximum advantage of any adversary in the $\mathrm{IND\text{-}CCA}$ game for the SAVER encryption scheme. Below, we will prove that $\mathrm{Adv}^{\delta}_{G_3,G_2}$ is negligible in λ.

We show how to build an adversary against the $\mathrm{IND\text{-}CCA}$ experiment using $\mathcal{A}$ by letting the challenger for G_3 act as normal, except for the following. The challengers picks some $j \in [n_{\mathsf{Tr}}]$ and when $\mathcal{A}$ makes the j-th **CreateTransaction** query, the challenger starts the $\mathrm{IND\text{-}CCA}$ game and obtains the SAVER public encryption key $\mathrm{pk}_{\mathsf{svr}}$. The challenger than sets $m_0 := m_{\mathsf{svr}}$, i.e. to be the plaintext message that would normally be encrypted under $\mathrm{pk}_{\mathsf{svr}}$, and $m_1 := r$ with r a randomly generated message of the appropriate length.

Afterwards, the challenger queries the $\mathrm{IND\text{-}CCA}$ challenger on the plaintexts (m_0, m_1) and receives $c := \mathrm{Enc}_{\mathrm{pk}_{\mathsf{svr}}}(m_{\bar{b}})$, where $\bar{b}$ is the internal bit of the $\mathrm{IND\text{-}CCA}$ challenger. Effectively, we replace the j-th SAVER ciphertext by c and continue as normal. When $\mathcal{A}$ ends the experiment with a guess b' we return this same b' as our guess in the $\mathrm{IND\text{-}CCA}$ game. To see that this makes sense, we note that when $\bar{b} = 0$ the view of $\mathcal{A}$ is identical to that in G_2. When $\bar{b} = 1$ the view of $\mathcal{A}$ is identical to that of an intermediate hybrid H (between G_2 and G_3) where the plaintext of only one ciphertext is replaced. Thus we get that $\mathrm{Adv}_{\mathsf{enc}}$ is an upper bound for $\mathrm{Adv}^{\delta}_{H,G_2}$. Then by a hybrid argument over all n_{Tr} ciphertexts it follows that $\mathrm{Adv}^{\delta}_{G_3,G_2} \leq n_{\mathsf{Tr}} \cdot \mathrm{Adv}_{\mathsf{enc}}$. We can thus conclude that $\mathrm{Adv}^{\delta}_{G_3,G_2}$ is negligible in λ.

From G_3 to G_4. Let $\mathrm{Adv}_{\mathsf{prf}}$ be the maximum advantage of any adversary in the PRF game against the used PRF-family. We show how to build an adversary that distinguishes our PRF from a true random function in the PRF game. We achieve this by using the adversary $\mathcal{A}$ as a subroutine and by making a small adaptation to G_3. Let with $\mathrm{sk}^*_{\mathsf{addr}}$ be the secret address key corresponding to the first **CreateAddress** query. We replace all calls to $\mathrm{PRF}_{\mathrm{sk}^*_{\mathsf{addr}}}$, with calls to the oracle of the PRF game using the same input.

To be precise, on receiving the first **CreateAddress** query we start the game against the PRF challenger, which gives us access to an oracle $\mathcal{O}^{\mathrm{PRF}}$ which either computes $PRF_{\mathrm{sk}_{\mathsf{addr}}}$ for some secret $\mathrm{sk}_{\mathsf{addr}}$ or returns a completely random output of the same length, depending on the value of the secret bit $\bar{b}$. If $\bar{b} = 0$, $\mathcal{O}^{\mathrm{PRF}}$ computes the value using the PRF. When $\bar{b} = 1$, a random value is returned. We use this oracle to compute $\mathrm{pk}^*_{\mathsf{addr}}$. Now on each query of type **ConvertFromNote** or

CreateTransaction we use $\mathcal{O}^{\mathrm{PRF}}$ to compute μ, η, and κ.

When $\mathcal{A}$ ends the game by outputting a guess b', we send this guess to the PRF challenger. To see that this makes sense, we note that when $\bar{b} = 0$, the view of $\mathcal{A}$ is identical to that in G_2. When $\bar{b} = 1$ the view of $\mathcal{A}$ is identical to that of an intermediate hybrid H (between G_3 and G_4) where only the PRF of the 'first user' is replaced by a true random function. We thus get that $\mathrm{Adv}_{\mathsf{prf}}$ is an upper bound for $\mathrm{Adv}^{\delta}_{H,G_3}$. By a hybrid argument over all n_{Acc} generated address secret keys (functioning as PRF keys), we derive that $\mathrm{Adv}^{\delta}_{G_4,G_3} \leq n_{\mathsf{Acc}} \cdot \mathrm{Adv}_{\mathsf{prf}}$, which is clearly negligible in λ.

From G_4 to G_{sim}. Let $\mathrm{Adv}_{\mathsf{comm}}$ denote the maximum advantage of any adversary in the hiding game against the used commitment scheme. We show how to build an adversary in the hiding game by using $\mathcal{A}$ as a subroutine, and making a small adaptation to G_4. We replace one of the commitments in a **ConvertToNote**, **CreateTransaction** or **CreateAccount** transaction by a commitment to a random string of the same length. To be precise we select an index $i \in [n_{\mathsf{CTo}} + 2 \cdot n_{\mathsf{Tr}} + n_{\mathsf{Acc}}]$ representing one of the commitments that is computed, with $\mathrm{pk}^{\mathsf{new}}_{\mathsf{addr}}$ in ADDR. When the transaction containing the i-th commitment is computed, we replace this commitment to a certain value x by the result of a call to the challenger in the hiding game. This call to the hiding game challenger has the inputs x and r where r is a random string of length $|x|$. When $\mathcal{A}$ ends the game by outputting a guess b', we send this guess to the hiding game challenger. To see that this makes sense, we note that when $\bar{b} = 0$ (the secret bit of the hiding game challenger), the view of $\mathcal{A}$ is identical to that in G_3. When $\bar{b} = 1$, the view of $\mathcal{A}$ is identical to that of an intermediate hybrid H (between G_4 and G_{sim}) where only the i-th commitment is replaced by a commitment to a random string. We thus conclude that $\mathrm{Adv}_{\mathsf{comm}}$ is an upper bound for $\mathrm{Adv}^{\delta}_{H,G_4}$. By a hybrid argument over all $n_{\mathsf{CTo}} + 2n_{\mathsf{Tr}} + n_{\mathsf{Acc}}$ commitments that are possibly replaced, we find that $\mathrm{Adv}^{\delta}_{G_{\mathsf{sim}},G_4} \leq (n_{\mathsf{CTo}} + 2n_{\mathsf{Tr}} + n_{\mathsf{Acc}}) \cdot \mathrm{Adv}_{\mathsf{comm}}$, which is clearly negligible in λ.

From the results above and our definition of Adv^{δ} we can conclude that $\mathrm{Adv}^{\delta}_{G_{\mathsf{sim}},G_{\mathsf{real}}}$ is negligible in λ. □

C.4 Transaction non-malleability

This paragraph outlines the definition of Transaction non-malleability, and also gives a definition of the Transaction non-malleability game. Next to this, we show that our payment scheme satisfies the transaction non-malleability definition.

Definition 18 (Transaction non-malleability)**.** *We say that a payment scheme Π has the transaction non-malleability property if, for every poly(λ)-time adversary $\mathcal{A}$ and* *sufficiently large* λ, $Adv^{TR\text{-}NM}_{\Pi,\mathcal{A}} < negl(\lambda)$, *with* $Adv^{TR\text{-}NM}_{\Pi,\mathcal{A}} := \mathbb{P}\left[TR\text{-}NM(\Pi, \mathcal{A}, \lambda) = 1\right]$.

Game $(\mathrm{TR\text{-}NM})$**.** Given a scheme Π, adversary $\mathcal{A}$, and security parameter λ, the game consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. At the start of the game the challenger performs the setup of Π, computes $\mathrm{pp} := \mathtt{Setup}(\lambda)$ and initialises an oracle $\mathcal{O}^{\mathsf{pay}}$. $\mathcal{A}$ is then allowed to send queries to $\mathcal{O}^{\mathsf{pay}}$, where after each query, $\mathcal{O}^{\mathsf{pay}}$ returns the added transaction to $\mathcal{A}$. At the end of the game, $\mathcal{A}$ sends $\mathcal{C}$ a transaction tx^*, and $\mathcal{C}$ returns a 1 if and only if tx^* satisfies the following requirements:

- tx^* is of type *Transfer*;
- tx^* does not appear on L as maintained by the oracle $\mathcal{O}^{\mathsf{pay}}$;
- $\exists \mathrm{tx}' \in L$ such that at least one of η and μ appears both on tx' and tx^*;
- **VerifyTransaction**$(\mathrm{pp}, \mathrm{tx}^*, L_{\leq \mathrm{tx}'})$ returns true.

Otherwise, $\mathcal{C}$ aborts and outputs 0.

Theorem 3. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The used signature scheme Sig is $\mathrm{SUF\text{-}1CMA}$ secure;
- $\mathrm{CRH}^{\mathsf{sig}}$ is a collision resistant hash function;
- The used zk-SNARK scheme is perfect zero-knowledge;
- The function family PRF is a pseudo random function family;
- PRF is a collision resistant function;
- The used zk-SNARK has computational knowledge soundness;

satisfies the transaction non-malleability property.

Proof. We begin with defining the set $\mathcal{T}_{\mathrm{pk}}$ as the collection of all public signature keys created as a response to a **CreateTransaction** query. The event that $\mathcal{A}$ wins this game can be split in four parts:

- EVENT$_{\mathsf{forge}}$ (forgery): $\mathcal{A}$ wins and $\exists \mathrm{pk}''_{\mathsf{sig}} \in \mathcal{T}_{\mathrm{pk}}$ such that $\mathrm{pk}''_{\mathsf{sig}} = \mathrm{pk}^*_{\mathsf{sig}}$;
- EVENT$_{\mathsf{col}}$ (collision in CRH): $\mathcal{A}$ wins, the above event does not occur, and $\exists \mathrm{pk}''_{\mathsf{sig}} \in \mathcal{T}_{\mathrm{pk}}$ such that $k^* = \mathrm{CRH}^{\mathsf{sig}}(\mathrm{pk}''_{\mathsf{sig}})$;
- EVENT$_{\mathsf{mac}}$ (PRF^{κ} is broken): $\mathcal{A}$ wins, the above two events do not occur, and $\kappa^* = \mathrm{PRF}^{\kappa}_{\mathrm{sk}'_{\mathsf{addr}}}(k^*)$;

- $\text{EVENT}_{\text{null}}$ ($\mathrm{PRF}^{\eta/\mu}$ or ZKP-scheme is broken): $\mathcal{A}$ wins, and the above three events do not occur.

Clearly, $\mathrm{Adv}_{\Pi,\mathcal{A}}^{\text{TR-NM}}$ is equal to the sum of the probabilities of all these events. Thus showing that the probabilities of all these events are negligible in λ suffices to argue that $\mathrm{Adv}_{\Pi,\mathcal{A}}^{\text{TR-NM}}$ is negligible in λ.

Bounding forgery event. Define $\epsilon := \mathbb{P}[\text{EVENT}_{\text{forge}}]$. Let (m^*, σ^*) and (m'', σ'') be the message signature pairs in tx^* respectively tx'', where tx'' is the transaction containing $\mathrm{pk}''_{\text{sig}}$. Moreover, note that $\mathrm{pk}^*_{\text{sig}} = \mathrm{pk}''_{\text{sig}}$ and that σ^* is a valid signature on the message m^*. We will first proof that $\mathrm{tx}^* \neq \mathrm{tx}''$. Firstly note that tx^* and tx' share at least one nullifier, and that $\mathrm{tx}^* \neq \mathrm{tx}'$. Now, suppose for the sake of contradiction that $\mathrm{tx}^* = \mathrm{tx}''$. This would imply that tx' and tx'' also share at least one nullifier. However, we do not allow two transactions on the blockchain to have the same nullifier (η or μ), hence we must have $\mathrm{tx}^* \neq \mathrm{tx}''$. We can now conclude that $(m^*, \sigma^*) \neq (m', \sigma')$, since $\mathrm{tx}^* \neq \mathrm{tx}''$.

We now describe how to construct an adversary for the $\mathrm{SUF\text{-}1CMA}$ game against the proposed signature scheme using $\mathcal{A}$. Firstly, we select a random index $j \in_R [\mathrm{n}_{\mathsf{Tr}}]$. We then perform the $\mathrm{TR\text{-}NM}$ game as usual, apart from the j-th **CreateTransaction** query. When this query is sent by $\mathcal{A}$ we start the $\mathrm{SUF\text{-}1CMA}$ game and query the challenger to obtain both a public signature key $\mathrm{pk}''_{\mathrm{sig}}$ and a signature σ'' on the created transaction tx''. When $\mathcal{A}$ wins the game by outputting a transaction tx^*, and if $\mathrm{pk}^*_{\text{sig}} = \mathrm{pk}''_{\text{sig}}$ we return (m^*, σ^*) as a forgery, otherwise we abort and lose the game.

Since j is selected uniformly at random from the set $[\mathrm{n}_{\mathsf{Tr}}]$, we win the game with probability $\epsilon / n_{\mathsf{Tr}}$. However, since our signature scheme satisfies the $\mathrm{SUF\text{-}1CMA}$ property we know that $\epsilon / n_{\mathsf{Tr}}$ and thus also ϵ must be negligible in λ.

Bounding collision event. Define $\epsilon := \mathbb{P}[\text{EVENT}_{\text{col}}]$. When this event occurs, we know that $\mathrm{pk}''_{\text{sig}} \neq \mathrm{pk}^*_{\text{sig}}$. However we also have that $\mathrm{CRH}(\mathrm{pk}^*_{\text{sig}}) = k^* = \mathrm{CRH}^{\text{sig}}(\mathrm{pk}''_{\text{sig}})$. Together this implies that $\mathcal{A}$ found a collision in CRH. However, because $\mathrm{CRH}^{\text{sig}}$ is collision-resistant this only happens with negligible probability. We thus conclude that ϵ is negligible in λ.

Bounding mac event. Define $\epsilon := \mathbb{P}[\text{EVENT}_{\text{mac}}]$. When this event happens we have $\mathrm{pk}'_{\text{sig}} \neq \mathrm{pk}^*_{\text{sig}}$, $k^* \neq k'$, and $\kappa^* = \mathrm{PRF}^{\kappa}_{\mathrm{sk}'_{\text{addr}}}(k^*)$. We also adapt the $\mathrm{TR\text{-}NM}$ game slightly, by simulating all zk-SNARK proofs instead of actually computing the proof. Notice that, since the used zk-SNARK scheme is perfect zero-knowledge, $\mathcal{A}$'s view is not altered, and hence ϵ is not influenced by this minor alteration.

We will show how to construct an adversary in the PRF game using $\mathcal{A}$. In the PRF game we get access to an oracle $\mathcal{O}^{\text{pay}}_{\text{prf}}$ that computes PRF_k for some secret key

k when the internal bit of the PRF game is $b = 0$. When $b = 1$ it returns a completely random value.

Firstly, we randomly select an index $j \in_R [n_{\mathsf{Acc}}]$. Secondly, we use this index to select the PRF calls that are to be replaced, namely all calls using the function $\mathrm{PRF}_{\mathrm{sk}''_{\mathsf{addr}}}$, with $\mathrm{sk}''_{\mathsf{addr}}$ being the secret address key generated in the j-th **CreateAccount** query. All these calls are replaced with a call to $\mathcal{O}^{\mathrm{PRF}}$, the oracle of the PRF game. Note, that we do not need to know the value $\mathrm{sk}''_{\mathsf{addr}}$ since all zk-SNARK proofs are simulated.

Finally, when $\mathcal{A}$ outputs a transaction tx^* we first check if $\mathrm{PRF}_{\mathrm{sk}''_{\mathsf{addr}}}(k^*)$ has been requested from $\mathcal{O}^{\mathrm{PRF}}$ before. When this is the case, we abort and output $b' = 0$. Otherwise, we request $\mathrm{PRF}_{\mathrm{sk}''_{\mathsf{addr}}}(k^*)$ from $\mathcal{O}^{\mathrm{PRF}}$ and if the result equals κ^*, we output our bit guess $b' = 0$ otherwise we return $b' = 1$.

Note that our advantage in the PRF game can be defined as $\mathbb{P}[b' = 0|b = 0] - \mathbb{P}[b' = 0|b = 1]$. We can analyse both terms separately, when defining ω to be the length of the PRF output:

- $\mathbb{P}[b' = 0|b = 0]$: Note that j is selected uniformly at random from the set $[n_{\mathsf{Acc}}]$, which implies that $\mathbb{P}[b' = 0|b = 0 \cap \text{no abort}] = \epsilon/n_{\mathsf{Acc}}$. This results in the following equality $\mathbb{P}[b' = 0|b = 0] := \mathbb{P}[\text{no abort}] \cdot \epsilon/n_{\mathsf{Acc}} + \mathbb{P}[\text{abort}]$.
- $\mathbb{P}[b' = 0|b = 1]$: Note that $\mathbb{P}[b' = 0|b = 1 \cap \text{no abort}] = 2^{-\omega}$. This implies that $\mathbb{P}[b' = 0|b = 1] := \mathbb{P}[\text{no abort}] \cdot 2^{-\omega} + \mathbb{P}[\text{abort}]$.

Observe that, we only abort if $\mathrm{PRF}_{\mathrm{sk}''_{\mathsf{addr}}}(k^*)$ is requested before $\mathcal{A}$ sends the forged transaction tx^*. However an evaluation of this exact value, implies that either $\mathrm{EVENT}_{\mathsf{forge}}$ or $\mathrm{EVENT}_{\mathsf{col}}$ happened before, which contradicts $\mathrm{EVENT}_{\mathsf{mac}}$. So we conclude that $\mathbb{P}[\text{abort}] = 0$. Moreover, it is trivial to see that $2^{-\omega}$ is negligible in λ. Hence, the probability that we win the PRF game is $\epsilon/n_{\mathsf{Acc}}$. However, since we selected a proper PRF-family this probability and thus also ϵ must be negligible in λ.

Bounding nullifier event. Define $\epsilon = \mathbb{P}[\mathrm{EVENT}_{\mathsf{null}}]$ and let $\mathcal{E}$ be the zk-SNARK extractor for $\mathcal{A}$. We show how to use $\mathcal{A}$ to construct a collision for either PRF^{μ} or PRF^{η}.

First, we run $\mathcal{A}$ and wait until if finishes and returns a transaction tx^*. We then apply the zk-SNARK extractor $\mathcal{E}$ for $\mathcal{A}$ on tx^* to obtain a witness a^* for the proof π^* in tx^*. With probability negligible in λ, $\mathcal{E}$ fails in obtaining a valid witness a, in that case we abort. Otherwise we parse a^* as $(\mu^*, \eta^*, \mathrm{sk}^*_{\mathsf{addr}}, *)$ and check whether η respectively μ appears on tx'. Now, if $\mathrm{sk}'_{\mathsf{addr}} \neq \mathrm{sk}^*_{\mathsf{addr}}$ we have clearly found a collision in PRF^{η} respectively PRF^{μ}.

We now recall that π^* is a valid proof and a^* a valid witness. Moreover, since tx^* is a valid transaction we know that $\kappa^* = \mathrm{PRF}^{\kappa}_{\mathrm{sk}^*_{\mathsf{addr}}}(k^*)$. And by the conditions of the event we know that $\kappa^* \neq \mathrm{PRF}^{\kappa}_{\mathrm{sk}'_{\text{addr}}}(k^*)$. Hence, we can conclude that for this event it always holds that $\mathrm{sk}'_{\text{addr}} \neq \mathrm{sk}^*_{\text{addr}}$. And hence our probability of finding a collision equals $\epsilon -$ negl(λ), where the last term comes from the possibility that $\mathcal{E}$ fails. Thus, since PRF^{η} and PRF^{μ} are both collision-resistant we conclude that ϵ must be negligible in λ. □

C.5 Balance

This section outlines the definition of Balance, and also gives a definition of the Balance game. We also show that our payment scheme has the balance property.

Definition 19 (Balance)**.** *We say that a payment scheme Π satisfies the balance property if , for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv^{BAL}_{\Pi,\mathcal{A}} <$ negl(λ), with $Adv^{BAL}_{\Pi,\mathcal{A}} = \mathbb{P}\left[BAL(\Pi,\mathcal{A},\lambda) = 1\right]$.*

Game (BAL)**.** Given a scheme Π, adversary $\mathcal{A}$, and security parameter λ, the game consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. At the start of the game the challenger performs the setup of Π, computes $\mathrm{pp} :=$ **Setup**(λ) and initialises an oracle $\mathcal{O}^{\text{pay}}$. $\mathcal{A}$ is then allowed to send queries to $\mathcal{O}^{\text{pay}}$, where after each query $\mathcal{O}^{\text{pay}}$ returns the added transaction to $\mathcal{A}$. At the end of the game, $\mathcal{A}$ sends $\mathcal{C}$ a set of notes N and a set of memory cells M. $\mathcal{C}$ then computes the following quantities:

- v_{Unspent} is the total value of all spendable notes in N which belong to a public address key $\mathrm{pk}_{\text{addr}}$ not in ADDR. Recall, that a note is spendable if its nullifier has not yet been published. The challenger can verify spendability of a note by constructing a **ConvertFromNote** query with the note as input.

- v_{Memory} is the total value of all unused memory cells in M that belong to a public address key $\mathrm{pk}_{\text{addr}}$ not in ADDR. Recall, that a memory cell is unused if its nullifier has not yet been published. The challenger can verify usability of a memory cell by constructing a **CreateTransaction** query consuming the old memory cell, using no input note, and setting the value of the new note equal to the value of the memory cell.

- $v_{\text{ConvertFrom}}$ is the total value of all input notes of *ConvertFrom* transaction that were placed on the ledger by an **Insert** query.

- $v_{\mathcal{A}\to\mathrm{ADDR}}$ is the total value of all transfer transaction from addresses controlled by $\mathcal{A}$ to addresses in ADDR. This can be computed by determining the set of notes N' that contains all notes in NOTE that are newly created in a transfer transaction placed on the ledger by an **Insert** query.

- $v_{\text{ConvertTo}}$ is the total value of all *ConvertTo* transactions on the ledger (either added via a **ConvertToNote** or **Insert** query), where the receiver of the note is not in ADDR.

- $v_{\mathrm{ADDR}\to\mathcal{A}}$ is the total value of all transfer transaction from addresses in ADDR to an address controlled by $\mathcal{A}$. This can be computed by looking at all transfer transactions that were added to the ledger via a **CreateTransaction** query and summing up the value of all notes that were transferred to an address not in ADDR.

After having computed this, $\mathcal{C}$ outputs 1 only if $v_{\text{Unspent}} + v_{\text{Memory}} + v_{\text{ConvertFrom}} + v_{\mathcal{A}\to\mathrm{ADDR}} > v_{\text{ConvertTo}} + v_{\mathrm{ADDR}\to\mathcal{A}}$. Otherwise $\mathcal{C}$ outputs 0.

Theorem 4. The payment scheme Π as defined in Section 6 that satisfies the following assumptions:

- The used zk-SNARK has computational knowledge soundness;

- CRH is a collision-resistant hash function;

- The used commitment scheme COMM is computationally binding and computationally hiding;

satisfies the balance property.

Proof. We modify the experiment slightly, but in such a way that $\mathcal{A}$'s view is not altered in any way. For all zero-knowledge proofs constructed by the oracle we store the witness a. Next to this, we use $\mathcal{E}$, the zero-knowledge extractor for $\mathcal{A}$, to obtain the witness a on all zk-SNARK proofs on L that were added as result of an **Insert** query. If $\mathcal{E}$ fails to obtain a valid witness, we abort and output 1. Because this only happens with probability negligible in λ, this plays no contradictory role in proving that $\mathrm{Adv}^{\mathrm{BAL}}_{\Pi,\mathcal{A}}$ is negligible in λ. In other words, to show that $\mathrm{Adv}^{\mathrm{BAL}}_{\Pi,\mathcal{A}}$ is negligible, it suffices to show that the probability that $\mathcal{A}$ wins our slightly altered experiment is negligible in λ.

We will proof this by first defining properties of a so called *balanced ledger*. Then, we show that a *balanced ledger* implies that $\mathcal{A}$ always loses. To complete our proof, we show that the probability that the ledger in our experiment is not a *balanced ledger* is negligible in λ. Combining these steps allows us to conclude that that $\mathrm{Adv}^{\mathrm{BAL}}_{\Pi,\mathcal{A}}$ is negligible in λ.

Balanced ledger properties. We begin by defining the properties that a ledger needs to satisfy, to be a *balanced ledger*.

1. no note forgery: Each transaction-witness pair (tx, a) of type *Transfer* or *ConvertFrom* that contains a $\mathrm{cm}^{\text{old}}_{\text{note}}$, has a valid opening thereof. Moreover, $\mathrm{cm}^{\text{old}}_{\text{note}}$ appears as output (note) commitment on a preceding *ConvertTo* or *Transfer* transaction. If there is no $\mathrm{cm}^{\text{old}}_{\text{note}}$ then $v^{\text{old}}_{\text{note}} = 0$.

2. no memory forgery: Each (tx, a) of type *Transfer* contains a valid opening of $\mathrm{cm}^{\text{old}}_{\text{mem}}$. Moreover, one of the following two statements must hold: (1) $\mathrm{cm}^{\text{old}}_{\text{mem}}$ appears as output (memory) commitment on a preceding *Transfer* transaction; or (2) $v^{\text{old}}_{\text{mem}} = 0$.

3. no note duplication: There exist no two (tx, a) and (tx', a') of type *Transfer* or *ConvertFrom* (both might have different types), with the same value of $\mathrm{cm}^{\text{old}}_{\text{note}}$.

4. no memory duplication: There exist no two (tx, a) and (tx', a') of type *Transfer*, with the same value of $\mathrm{cm}^{\text{old}}_{\text{mem}}$.

5. 'intra'-transaction balance: For every (tx, a) of type *Transfer* the balance equation holds: $v^{\text{old}}_{\text{note}} + v^{\text{old}}_{\text{mem}} = v^{\text{new}}_{\text{note}} + v^{\text{new}}_{\text{mem}}$.

6. 'inter'-transaction note consistency: For every (tx, a) of type *Transfer* or *ConvertFrom* the following holds. If $\mathrm{cm}^{\text{old}}_{\text{note}}$ is the output note commitment of $(\overline{\mathrm{tx}}, \overline{a})$ (of type *ConvertTo* or *Transfer*) then $\overline{v^{\text{new}}_{\text{note}}} = v^{\text{old}}_{\text{note}}$.

7. 'inter'-transaction memory consistency: For every (tx, a) of type *Transfer* the following holds. If $\mathrm{cm}^{\text{old}}_{\text{mem}}$ is the output memory commitment of $(\overline{\mathrm{tx}}, \overline{a})$ then $\overline{v^{\text{new}}_{\text{mem}}} = v^{\text{new}}_{\text{mem}}$.

8. no note steal: For every (tx, a) where tx was added to L by an **Insert** query and with tx of type *Transfer* or *ConvertFrom*, if $\mathrm{cm}^{\text{old}}_{\text{note}}$ appears on a preceding transaction $\overline{\mathrm{tx}}$ then $\overline{\mathrm{pk}^{\text{new}}_{\text{addr}}}$ does not appear on ADDR.

9. no memory steal: For every (tx, a) where tx was added to L by an **Insert** query and with tx of type *Transfer*, if $\mathrm{cm}^{\text{old}}_{\text{mem}}$ appears on a preceding transaction $\overline{\mathrm{tx}}$ then $\overline{\mathrm{pk}^{\text{new}}_{\text{addr}}}$ does not appear on ADDR.

Balanced ledger $\Rightarrow$ $\mathcal{A}$ loses. We will now prove by induction, over the queries issued by $\mathcal{A}$, that if L is a balanced ledger then $v_{\text{Unspent}} + v_{\text{Memory}} + v_{\text{ConvertFrom}} + v_{\mathcal{A}\to\mathrm{ADDR}} \leq v_{\text{ConvertTo}} + v_{\mathrm{ADDR}\to\mathcal{A}}$.

For the base case, we note that the empty ledger is clearly a balanced ledger. We have $v_{\text{Unspent}} + v_{\text{Memory}} + v_{\text{ConvertFrom}} + v_{\mathcal{A}\to\mathrm{ADDR}} \leq v_{\text{ConvertTo}} + v_{\mathrm{ADDR}\to\mathcal{A}}$, since all terms equal 0.

For the induction step, assume that we have a balanced ledger after n queries issued by $\mathcal{A}$ and moreover $v_{\text{Unspent}} + v_{\text{Memory}} + v_{\text{ConvertFrom}} + v_{\mathcal{A}\to\mathrm{ADDR}} \leq v_{\text{ConvertTo}} +$

$v_{\mathrm{ADDR}\to\mathcal{A}}$. We will now show that for the following query issued by $\mathcal{A}$ such that the balanced ledger properties do not become violated, we still have $v_{\text{Unspent}} + v_{\text{Memory}} + v_{\text{ConvertFrom}} + v_{\mathcal{A}\to\mathrm{ADDR}} \leq v_{\text{ConvertTo}} + v_{\mathrm{ADDR}\to\mathcal{A}}$. From now on we will refer to this inequality as the *ledger balance inequality*.

We will proof the induction step by a case distinction on the type of the $n+1$-th query:

- **InsertAdmin**, **CreateAccount**, **Revokeaccount**, and **ConvertFromNote** do not influence any of the terms in the *ledger balance inequality*, thus for those transaction types our *ledger balance inequality* will still hold. For clarity, we emphasise that **ConvertFromNote** queries do not count towards $v_{\text{ConvertFrom}}$ as this value only includes transactions placed on the ledger by an **Insert** query.
- **ConvertToNote**: A transaction of this type always increases $v_{\text{ConvertTo}}$ with $v^{\text{new}}_{\text{note}}$, and it possibly increases v_{Unspent} with the same value $v^{\text{new}}_{\text{note}}$ (follows from *property 6*). This implies that either only the right-hand side of the inequality increases, or that both sides of the inequality increase with the same value. In both cases the *ledger balance inequality* will still be satisfied.
- **CreateTransaction**: A transaction of this type clearly cannot influence any terms other than $v_{\mathrm{ADDR}\to\mathcal{A}}$ and v_{Unspent}, since the input note can only be owned by an address in ADDR. Moreover, these terms are only influence when $\mathrm{pk}^{\text{new}}_{\text{addr}}$ is not in ADDR. When that happens, it follows from *property 6* that both values are increased with the same value $v^{\text{new}}_{\text{note}}$. Hence, the *ledger balance inequality* will still hold.
- **Insert** We make a case distinction on the type of transaction that is inserted:
  - *NewAccount* and *RvkAccount* clearly have no influence on any of the terms in the *ledger balance inequality*, thus the inequality will still hold.
  - *ConvertTo*: A transaction of this type increases $v_{\text{ConvertTo}}$ with $v^{\text{new}}_{\text{note}}$, and it increases v_{Unspent} with the same value $v^{\text{new}}_{\text{note}}$ (follows from *property 6*). It trivially follows that the *ledger balance inequality* will still be satisfied.
  - *ConvertFrom*: An inserted *ConvertFrom* transaction increases $v_{\text{ConvertFrom}}$ with $v^{\text{old}}_{\text{note}}$, however by *property 1*, *property 3*, *property 6* and *property 8* the value of v_{Unspent} is also decreased with this same value. Hence, the left-hand side of *ledger balance inequality* keeps the same value, and therefore the inequality still holds.
  - *Transfer*: The input memory cell of an inserted transfer transaction has value $v^{\text{old}}_{\text{mem}}$. By *property 2*, *property 4*, *property 7*, and *property 9* we know that the value of v_{Memory} is decreased with exactly this value $v^{\text{old}}_{\text{mem}}$.

Remember that, if there is no input memory cell $v^{\text{old}}_{\text{mem}}$ equals 0, so nothing happens. Next to this, the input note has value $v^{\text{old}}_{\text{note}}$, and by *property 1*, *property 3*, *property 6*, and *property 8*, the value of v_{Unspent} is decreased with exactly this value $v^{\text{old}}_{\text{note}}$. Combined, these inputs decrease the value of the left hand side of the inequality with $v^{\text{old}}_{\text{mem}} + v^{\text{old}}_{\text{note}}$.

Concerning the outputs, the value v_{Memory} increases by $v^{\text{old}}_{\text{mem}}$, since this is the value of the new memory cell, that will be unused after this transaction. Moreover, the output note with value $v^{\text{new}}_{\text{note}}$ influences either v_{Unspent} or $v_{\mathcal{A}\to\text{ADDR}}$, depending on the receiver address of the note $\text{pk}^{\text{new}}_{\text{addr}}$. If this address key is in ADDR, the value $v_{\mathcal{A}\to\text{ADDR}}$ increases with $v^{\text{new}}_{\text{note}}$, otherwise v_{Unspent} increases with $v^{\text{new}}_{\text{note}}$. Irrespective thereof, the left hand side of the inequality is increased with $v^{\text{new}}_{\text{mem}} + v^{\text{new}}_{\text{note}}$.

By *property 5* $v^{\text{old}}_{\text{mem}} + v^{\text{old}}_{\text{note}} = v^{\text{new}}_{\text{mem}} + v^{\text{new}}_{\text{note}}$, and therefore the value of the left hand side of the *ledger balance inequality* has the same value as before this transaction. The right hand side of the inequality remains unaltered. Thus we can conclude that the inequality is still satisfied.

Violating balanced properties. It now remains to show that the probability of violating any of the balanced ledger properties is negligible in λ in order to conclude that $\text{Adv}^{\text{BAL}}_{\Pi,\mathcal{A}} = \text{negl}(\lambda)$. We proof this by showing per condition that the probability of violating that condition is negligible.

1. By construction of the oracle $\mathcal{O}^{\text{pay}}$, all transactions that are not inserted via **Insert** queries automatically satisfy this property. Therefore the only way to violate this property is by an **Insert** transaction tx of type *Transfer* or *ConvertFrom*. This transaction should then satisfy two conditions: (1) $\text{cm}^{\text{old}}_{\text{note}}$ does not appear as output note commitment on any preceding transaction on L; and (2) $v^{\text{old}}_{\text{note}} \neq 0$. Note that tx must be a valid transaction, thus for $v^{\text{old}}_{\text{note}}$ to be unequal to 0, we must have a valid Note Merkle tree path $\text{path}_{\text{note}}$. Otherwise the soundness of our zk-SNARK scheme would be broken. However, if $\text{cm}^{\text{old}}_{\text{note}}$ does not appear in the Note Merkle tree preceding this transaction, $\mathcal{A}$ must have found a collision in the hash function CRH. Since CRH is collision-resistant by definition, the probability that $\mathcal{A}$ violates this condition is negligible in λ.

2. The proof for this condition is analogous to the previous one, but only considers transactions of type *Transfer*.

3. For this property to be violated, L must contain two transactions tx and tx' of type *Transfer* or *ConvertFrom*. Both should use the same input note $\text{cm}^{\text{old}}_{\text{note}}$ but publish different values for η, i.e. $\eta \neq \eta'$, since both transactions are valid. These values η are both computed as $\eta = \text{PRF}_{\text{sk}_{\text{addr}}}(\text{pos}_{\text{note}})$, implying that

either (1) $\mathrm{sk}_{\textsf{addr}} \neq \mathrm{sk}'_{\textsf{addr}}$ or (2) $\mathrm{pos}_{\textsf{note}} \neq \mathrm{pos}'_{\textsf{note}}$. Otherwise, the soundness of the zk-SNARK scheme would be broken. We will show that both cases can only happen with probability negligible in λ:

(a) Together with the fact that $\mathrm{cm}^{\textsf{old}}_{\textsf{note}}$ is equal in both transactions, this implies one of the following: (1) $\mathrm{pk}_{\mathrm{addr}} \neq \mathrm{pk}'_{\mathrm{addr}}$; or (2) not the previous, but $\mathrm{cm}_{\textsf{cred}} \neq \mathrm{cm}'_{\textsf{cred}}$; or (3) neither of the previous two, but $(\mathrm{path}_{\textsf{cred}}, \mathrm{pos}_{\textsf{cred}}) \neq (\mathrm{path}'_{\textsf{cred}}, \mathrm{pos}'_{\textsf{cred}})$. However, all of these can only happen with negligible probability: (1) because of the binding property of $\mathrm{COMM}^{\textsf{note}}$; (2) because of the binding property of $\mathrm{COMM}^{\textsf{cred}}$; (3) because of collision-resistance of CRH.

(b) As both pairs $(\mathrm{path}_{\textsf{note}}, \mathrm{pos}_{\textsf{note}})$ and $(\mathrm{path}'_{\textsf{note}}, \mathrm{pos}'_{\textsf{note}})$ are valid Note Merkle tree path for the commitment, $\mathcal{A}$ must have found a collision in CRH. Due to CRH being collision-resistant this can only happen with negligible probability.

4. The proof for this condition is analogous to the previous one, but only considers transactions of type *Transfer*.

5. The fact that this condition can only be broken with negligible probability follows directly from the fact that all transactions on L are valid, and that the balance equation $v^{\textsf{old}}_{\textsf{note}} + v^{\textsf{old}}_{\textsf{mem}} = v^{\textsf{new}}_{\textsf{note}} + v^{\textsf{new}}_{\textsf{mem}}$ is enforced in the zk-SNARK proof. From the soundness of this proof we conclude that this condition cannot be broken with a probability non-negligible in λ.

6. If $\mathrm{cm}^{\textsf{old}}_{\textsf{note}} = \overline{\mathrm{cm}^{\textsf{old}}_{\textsf{note}}}$, but $v^{\textsf{old}}_{\textsf{note}} \neq \overline{v^{\textsf{old}}_{\textsf{note}}}$ we contradict the binding property of $\mathrm{COMM}^{\textsf{note}}$. Obviously, this can only happen with probability negligible in λ.

7. The proof for this condition is analogous to the previous one, but only considers transactions of type *Transfer*.

8. If this condition is broken, $\mathcal{A}$ is able to generate a valid proof using $\mathrm{pk}_{\textsf{addr}}$ for $\mathrm{cm}^{\textsf{old}}_{\textsf{note}}$ belonging to $\overline{\mathrm{pk}^{\textsf{new}}_{\textsf{addr}}}$ in ADDR. This can happen in one of the following ways: (1) $(\overline{v^{\textsf{new}}_{\textsf{note}}}, \overline{\mathrm{pk}^{\textsf{new}}_{\textsf{addr}}}) \neq (v^{\textsf{old}}_{\textsf{note}}, \mathrm{pk}_{\textsf{addr}})$; or (2) $\overline{\mathrm{sk}_{\textsf{addr}}} \neq \mathrm{sk}_{\textsf{addr}}$; or (3) $(\mathrm{path}_{\textsf{cred}}, \mathrm{pos}_{\textsf{cred}}) \neq (\mathrm{path}'_{\textsf{cred}}, \mathrm{pos}'_{\textsf{cred}})$; or (4) $\mathcal{A}$ knows $\overline{sk}_{\textsf{addr}}, \overline{pk}_{\textsf{addr}}, \overline{s}_{\textsf{cred}}$. We will now show that all of this cases can only occur with probability negligible in λ: (1) because of the binding property of $\mathrm{COMM}^{\textsf{note}}$; (2) because of the binding property $\mathrm{COMM}^{\textsf{cred}}$; (3) because of the collision-resistance of CRH; (4) because of the hiding property of $\mathrm{COMM}^{\textsf{cred}}$.

9. The proof for this condition is analogous to the previous one, but only considers transactions of type *Transfer*.

□

C.6 Access control

In this section we define the access control property and game and show that our payment scheme satisfies this property.

Definition 20 (Access control)**.** *We say that a payment scheme Π satisfies the access control property, if for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv^{AC}_{\Pi,\mathcal{A}} <$ negl(λ), with $Adv^{AC}_{\Pi,\mathcal{A}} = \mathbb{P}\left[AC(\Pi, \mathcal{A}, \lambda) = 1\right]$.*

Game (AC)**.** The Access Control game AC is defined as follows. Given a scheme Π, an adversary $\mathcal{A}$, and security parameter λ, the game consists of a series of interactions between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. The game starts with the challenger performing the setup of Π and initialising the oracle $\mathcal{O}^{\text{pay}}$. Subsequently, $\mathcal{A}$ receives the public parameters pp from the challenger. $\mathcal{A}$ is then allowed to send all types of queries to $\mathcal{O}^{\text{pay}}$, apart from the **Insert** and **InsertAdmin** query type. Effectively, $\mathcal{A}$ is only allowed to elicit behaviour of honest users. After receiving a query, $\mathcal{O}^{\text{pay}}$ updates the ledger accordingly and sends the resulting ledger L to $\mathcal{A}$. $\mathcal{A}$ ends the game by sending a transaction tx to the challenger. $\mathcal{A}$ wins the game only if this transaction is a valid transaction and if there is no other transaction tx' on the blockchain for which $\mathrm{tx}' = \mathrm{tx}$. The challenger can verify this by using tx as input to an **Insert** query, that should update the ledger accordingly.

Theorem 5. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The used signature scheme Sig is $\mathrm{SUF\text{-}CMA}$ secure;
- The used zk-SNARK scheme has computational knowledge soundness;
- The used commitment scheme COMM is binding;
- CRH is a collision resistant hash function;

satisfies the access control property.

Proof. We distinguish the event that $\mathcal{A}$ wins the AC game in two cases.

The first case considers the transaction tx having type *Transfer*. It should be noted that $\mathcal{A}$ does not know the opening to any of the commitments on the Credential Merkle Tree. In order to show that $\mathcal{A}$ can only construct a transaction tx of type *Transfer* with negligible advantage, we slightly adapt the experiment. On receiving tx, $\mathcal{C}$ also uses the zk-SNARK extractor $\mathcal{E}$ for $\mathcal{A}$ to obtain the witness a for tx. If $\mathcal{E}$ fails in doing so, we abort and output 1, otherwise $\mathcal{C}$ continues as above. We note two things, (1) the view of $\mathcal{A}$ is not altered, and hence $\mathcal{A}$'s advantage in this game, when $\mathcal{E}$ does not fail, is the same as in AC, (2) $\mathcal{E}$ only fails with negligible advantage. Thus we only need to show that $\mathcal{A}$ has negligible advantage in this altered game, when $\mathcal{E}$ was able to extract a valid witness.

A valid witness a must contain the values $(\mathrm{cm}_{\mathsf{cred}}, \mathrm{pk}_{\mathsf{addr}}, \mathrm{sk}_{\mathsf{addr}}, s_{\mathsf{cred}})$, such that either (i) $\mathrm{cm}_{\mathsf{cred}} = \mathrm{Comm}^{\mathsf{cred}}_{s_{\mathsf{cred}}}(\mathrm{pk}_{\mathsf{addr}}, \mathrm{sk}_{\mathsf{addr}})$ for some $\mathrm{cm}_{\mathsf{cred}}$ in the Credential Merkle tree; or (ii) there exists a valid Merkle path to $\mathrm{rt}_{\mathsf{cred}}$ for some $\mathrm{cm}_{\mathsf{cred}}$ not in the Credential Merkle tree. In the case of (i), since there is no pair $(\mathrm{cm}_{\mathsf{cred}}, \mathrm{pk}_{\mathsf{addr}})$ for which $\mathcal{A}$ knows $(\mathrm{sk}_{\mathsf{addr}}, s_{\mathsf{cred}})$. Thus by the hiding and binding property of $\mathrm{Comm}^{\mathsf{cred}}$ this can only happen with probability negligible in λ. In case (ii) the adversary has found a collision in CRH, which can only happen with negligible probability. Because of this, the first case can only occur with probability negligible in λ.

The second case tx has any of the other types: *RvkAccount*, *NewAccount*, *ConvertTo*, *ConvertFrom*. All these transactions must be signed using a secret signature key of (one of the) admins. Since $\mathcal{A}$ does not know such a key, and since tx does not appear on the blockchain yet, we can use $\mathcal{A}$ to construct an adversary in the $\mathrm{SUF\text{-}CMA}$ game. Since the used signature scheme has the $\mathrm{SUF\text{-}CMA}$ property, $\mathcal{A}$ cannot construct tx in this case with a more than negligible advantage.

Since both cases only occur with negligible probability, also the event that $\mathcal{A}$ wins occurs only with probability negligible in λ. □

C.7 Spend limit

Definition 21 (Spend limit). *We say that a payment scheme Π satisfies the spend limit property if, for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv^{LIM}_{\Pi,\mathcal{A}} <$ negl(λ), with $Adv^{LIM}_{\Pi,\mathcal{A}} = \mathbb{P}\left[LIM(\Pi, \mathcal{A}, \lambda) = 1\right]$.*

Game (LIM). Given a scheme Π, adversary $\mathcal{A}$, and security parameter λ, the game consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. At the start of the game the challenger sends to $\mathcal{A}$ a time span T and spend limit value L. Subsequently, the challenger performs the setup of Π, computes $\mathrm{pp} := $ **Setup**(λ) and initialises an oracle $\mathcal{O}^{\mathsf{pay}}$. $\mathcal{A}$ is then allowed to send queries to $\mathcal{O}^{\mathsf{pay}}$, where after each query $\mathcal{O}^{\mathsf{pay}}$ returns the added transaction to $\mathcal{A}$. At the end of the game, $\mathcal{A}$ sends $\mathcal{C}$ a set of *Transfer* transactions $\mathcal{TX}$. $\mathcal{C}$ first computes the witness a for each *Transfer* transaction on the ledger, using the witness extractor $\mathcal{E}$ for $\mathcal{A}$. If $\mathcal{E}$ fails to produce any of the valid witnesses $\mathcal{C}$ aborts and outputs 1. Then, $\mathcal{C}$ outputs a 1 if and only if all of the following statements hold:

- All transactions in $\mathcal{TX}$ have type *Transfer*;

- For any witness a belonging to a transaction $\mathrm{tx} \in \mathcal{TX}$, we have $b_{\text{saver}} = 0$;
- The input notes of all transactions in $\mathcal{TX}$ are owned by the same public address key. This can be verified for any transaction using the corresponding witness a;
- The transaction time of all transactions in $\mathcal{TX}$ falls in the range $[T_{min}, T_{min} + T]$, with T_{min} defined as the minimum transaction time t^{new} of all transactions in $\mathcal{TX}$. ;
- The sum of the output notes of all transactions in $\mathcal{TX}$ is larger than L.

Otherwise $\mathcal{C}$ aborts and outputs 0.

Before we prove that our payment scheme Π satisfies this property, we first give some useful lemmas.

Lemma 1. For any $\mathrm{pk}_{\text{addr}}$ the first *Transfer* transaction tx, with witness a, of that address uses no input memory cell and returns an output memory cell with value $c^{\text{new}} = v^{\text{new}}_{\text{note}} \cdot (1 - b_{\text{saver}})$. Additionally, this transaction outputs a memory nullifier equal to $\mu = \mathrm{PRF}^{\mu}_{\mathrm{sk}_{\text{addr}}}(-1)$. The above holds will but negligble probability.

Proof. We will first show that the transaction has no input memory cell by means of contradiction. Suppose to the contrary, that the first *Transfer* transaction of $\mathrm{pk}_{\text{addr}}$ does have a value for cm_{mem}. Clearly, we must have that $\mathrm{cm}^{\text{old}}_{\text{mem}}$ is a commitment to $\mathrm{pk}_{\text{addr}}$, and some other inputs and randomness. This implies that one of the following conditions hold: (1) $\mathrm{cm}^{\text{old}}_{\text{mem}}$ is present in the Memory Merkle tree; or (2) $\mathrm{cm}^{\text{old}}_{\text{mem}}$ is not present in the Memory Merkle tree. Both cases give rise to a contradiction: (1) contradicts the binding property of $\mathrm{Comm}^{\text{mem}}$; and (2) contradicts collision-resistance of CRH.

It now remains to show that the output memory cell will have value $c^{\text{new}} = v^{\text{new}}_{\text{note}} \cdot (1 - b_{\text{saver}})$. This follows directly from the fact that, due to there not being an input memory cell, the zk-SNARK proof ensures we have $c^{\text{old}} = 0$. By the soundness of our proof we immediately conclude that $c^{\text{new}}_{\text{note}} = v^{\text{new}}_{\text{note}} \cdot (1 - b_{\text{saver}})$, which concludes the proof of our lemma. □

Lemma 2. For any $\mathrm{pk}_{\text{addr}}$, any *Transfer* transaction tx, with witness a, that is not the first *Transfer* transaction of that address, uses the memory cell of the previous transaction of $\mathrm{pk}_{\text{addr}}$ as input and outputs a new memory cell with $c^{\text{new}} = c^{\text{old}} + v^{\text{new}}_{\text{note}} \cdot (1 - b_{\text{saver}})$. The above holds will but negligble probability.

Proof. Given $\mathrm{pk}_{\text{addr}}$, we first prove that any of these transactions must have an input memory cell. Suppose, for the sake of contradiction, that there is a transaction tx^* (not the first) for which there is no input memory cell. By the previous lemma, this implies that there are two transaction with no input memory cell: the first transaction tx^0 and tx^*. For both transactions to be accepted it most hold that $\mu^0 \neq \mu^*$. Since there is no input memory cell, the zero-knowledge proof implies that $\mathrm{sk}^0_{\text{addr}} \neq \mathrm{sk}^*_{\text{addr}}$. Given the above, we consider two cases that both lead to a contradiction: (1) $\mathrm{cm}_{\text{cred}}$ is present in the Memory Merkle tree; or (2) $\mathrm{cm}_{\text{cred}}$ is not present in the Memory Merkle tree. Case (1) contradicts the binding property of $\mathrm{Comm}^{\text{cred}}$; and (2) contradicts collision-resistance of CRH.

Next, we show that the input memory cell must be the output memory cell of the previous transaction of $\mathrm{pk}_{\text{addr}}$. Suppose to the contrary, that there exists a transaction tx^* for which the input memory cell is not the output memory cell of the previous transaction. Due to the previous part of this lemma, we know that we must have an input memory cell, so that leaves us with the following possibilities: (1) the commitment $\mathrm{cm}^{\text{old}}_{\text{mem}}$ belongs to another transaction tx' of $\mathrm{pk}_{\text{addr}}$; or (2) the commitment $\mathrm{cm}^{\text{old}}_{\text{mem}}$ belongs to another public address key $\overline{\mathrm{pk}_{\text{addr}}}$; or (3) the commitment $\mathrm{cm}^{\text{old}}_{\text{mem}}$ is not present in the Memory Merkle tree. In case (1) we must have $\mu' \neq \mu^*$, which implies that either $\mathrm{sk}'_{\text{addr}} \neq \mathrm{sk}^*_{\text{addr}}$ or $\mathrm{pos}'_{\text{mem}} \neq \mathrm{pos}^*_{\text{mem}}$. The first contradicts either the binding property of $\mathrm{Comm}^{\text{cred}}$ or collision-resistance of CRH. The second also contradicts collision-resistance in CRH. In case (2) we directly contradict the binding property of $\mathrm{Comm}^{\text{mem}}$. Finally, case (3) contradicts the collision-resistance of CRH.

Finally, we show that $c^{\text{new}} = c^{\text{old}} + v^{\text{new}}_{\text{note}} \cdot (1 - b_{\text{saver}})$. This follows directly from the fact that we have an input memory cell, and from the soundness of our proof. This also concludes the proof of this lemma. □

Theorem 6. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The used commitment scheme COMM satisfies the binding property;
- CRH is a collision-resistant hash function;
- The used zk-SNARK scheme has computational knowledge soundness;

satisfies the spend limit property.

Proof. Assume that $\mathcal{A}$ wins the game with probability non-negligible in λ. Let tx^* be the last, with regards to t_{new}, transaction in $\mathcal{TX}$ with ceiling memory cell commitment $\mathrm{cm}^{\text{ceil}}_{\text{mem}}$. The ledger contains another transaction tx' of $\mathrm{pk}_{\text{addr}}$ for which the new memory commitment is equal to $\mathrm{cm}^{\text{ceil}}_{\text{mem}}$. By definition of $\mathcal{TX}$, we know that $\mathrm{tx}' \notin \mathcal{TX}$. Now, remember that the sum of the output notes of all transactions, i.e. $v^{\text{new}}_{\text{note}}$ in $\mathcal{TX}$ is larger than L. We combine this fact with Lemma 1, Lemma 2, and the soundness of zk-SNARK proofs to conclude that $c^{\text{new}} - c^{\text{ceil}} > L$, with all but negligible probability.

This clearly contradicts the soundness of our zk-SNARK proof, also with all but negligible probability. Therefore, $\mathcal{A}$ can not win with a more than negligible probability. □

C.8 Accountability

In this section we define the accountability property and game and show that our payment scheme satisfies this property.

Definition 22 (Accountability)**.** *We say that a payment scheme Π satisfies the accountability property, if for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv_{\Pi,\mathcal{A}}^{ACC} <$ negl(λ), with $Adv_{\Pi,\mathcal{A}}^{ACC} = \mathbb{P}\left[ACC(\Pi, \mathcal{A}, \lambda) = 1\right]$.*

Game (ACC)**.** The Accountability game ACC is defined as follows. Given a scheme Π, an adversary $\mathcal{A}$, and security parameter λ, the game consists of a series of interactions between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. The game starts with the challenger performing the setup of Π and initialising the oracle $\mathcal{O}^{\mathsf{pay}}$. Subsequently, $\mathcal{A}$ receives the public parameters pp from the challenger. $\mathcal{A}$ is then allowed to send all types of queries to $\mathcal{O}^{\mathsf{pay}}$. After receiving a query, $\mathcal{O}^{\mathsf{pay}}$ updates the ledger accordingly and sends the resulting ledger L to $\mathcal{A}$. $\mathcal{A}$ ends the game by sending a transaction tx to the challenger. $\mathcal{C}$ then uses the witness extractor $\mathcal{E}$ for $\mathcal{A}$ to obtain the witness a for tx. $\mathcal{A}$ wins the game only if either the witness extractor fails to produce a valid witness a, or if $b_{\mathsf{saver}} \neq 0$ and $\mathrm{data}_{\mathsf{saver}}$ cannot be decrypted to $\mathrm{pk}_{\mathsf{addr}}$. $\mathrm{pk}_{\mathsf{addr}}^{\mathsf{new}}$, and $v_{\mathsf{new}}^{\mathsf{note}}$.

Theorem 7. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The used zk-SNARK scheme has computational knowledge soundness;
- The used SAVER encryption scheme computationally satisfies the encryption knowledge soundness and decryption soundness properties.

satisfies the accountability property.

Proof. Suppose that $\mathcal{A}$ does win the ACC game with non-negligible probability. $\mathcal{A}$ can do so in two ways. The first is, when $\mathcal{E}$ fails to provide a valid witness a. This only happens with negligible probability. Hence, $\mathcal{A}$ should succeed in the second case with non-negligible probability. In the second case we must have that $b_{\mathsf{saver}} = 0$ and $\mathrm{data}_{\mathsf{saver}}$ does not decrypt to $\mathrm{pk}_{\mathsf{addr}}^{\mathsf{new}}$, and $v_{\mathsf{new}}^{\mathsf{note}}$. However, by definition of $\mathrm{data}_{\mathsf{saver}}$ our zero-knowledge proof π, the knowledge-soundness of our zk-SNARK scheme, and the encryption and decryption soundness of SAVER this cannot happen with non-negligible probability.

Therefore we have reached contradiction, and must thus conclude that $\mathcal{A}$ cannot win the ACC game with more than negligible probability. $\square$

C.9 Timelock

In this section we define the timelock property and game and show that our payment scheme satisfies this property.

Definition 23 (Timelock)**.** *We say that a payment scheme Π satisfies the accountability property, if for every poly(λ)-time adversary $\mathcal{A}$ and sufficiently large λ, $Adv^{TL}_{\Pi,\mathcal{A}} <$ negl(λ), with $Adv^{TL}_{\Pi,\mathcal{A}} = \mathbb{P}\left[TL(\Pi,\mathcal{A},\lambda) = 1\right]$.*

Game (TL)**.** The Timelock game TL is defined as follows. Given a scheme Π, an adversary $\mathcal{A}$, and security parameter λ, the game consists of a series of interactions between $\mathcal{A}$ and a challenger $\mathcal{C}$, which terminates with an output bit from $\mathcal{C}$. The game starts with the challenger performing the setup of Π and initialising the oracle $\mathcal{O}^{\text{pay}}$. Subsequently, $\mathcal{A}$ receives the public parameters pp from the challenger. $\mathcal{A}$ is then allowed to send all types of queries to $\mathcal{O}^{\text{pay}}$. After receiving a query, $\mathcal{O}^{\text{pay}}$ updates the ledger accordingly and sends the resulting ledger L to $\mathcal{A}$. $\mathcal{A}$ ends the game by sending two transactions tx' and tx^* to the challenger. $\mathcal{C}$ then uses the witness extractor $\mathcal{E}$ for $\mathcal{A}$ to obtain the witness a for both transactions. $\mathcal{A}$ wins the game only if either the witness extractor fails to produce a valid witness a, or if all of the following statements hold:

- Both transactions are valid, i.e. **VerifyTransaction** returns true on both tx^* and tx'.
- $\mathrm{cm}^{\text{new}}_{\text{note}}$ in tx' is equal to $\mathrm{cm}^{\text{old}}_{\text{note}}$ in tx^*;
- For the values t_δ and t^{new} in tx' and $\overline{t_{\text{new}}}$ in tx^* the following holds: $t_\delta + t^{\text{new}} > \overline{t^{\text{new}}}$.

Theorem 8. The payment scheme Π as defined in Chapter 6 that satisfies the following assumptions:

- The used zk-SNARK scheme has computational knowledge soundness;
- CRH is a collision-resistant hash function;
- The used commitment scheme COMM satisfies the binding property;

satisfies the timelock property.

Proof. Suppose that $\mathcal{A}$ wins the game with non-negligible probability. The first of two cases in which this happens is when $\mathcal{E}$ fails to produces a valid witness for both transactions. By the soundness of our zk-SNARK scheme this is not possible with a more than negligible probability. Hence the other case must occur with non-negligible probability.

By soundness of the zk-SNARK scheme and the condition that the input note for tx^* must be unlocked, $(t_\delta, t^{\textsf{new}})$ in tx' cannot be equal to $(t_\delta^{\textsf{old}}, t_{\textsf{note}}^{\textsf{old}})$ in tx^* with a more than negligible probability. Hence, we must have that both tuples are unequal. However, by the condition of our case we must have that $\mathrm{cm}_{\textsf{note}}^{\textsf{new}} = \mathrm{cm}_{\textsf{note}}^{\textsf{old}}$, if the values of t_δ are to be unequal with more than negligible probability, we contradict the binding property of COMM. When $t^{\textsf{new}}$ in tx' is not equal to $t_{\textsf{note}}^{\textsf{old}}$ in tx^*, but $\mathrm{cm}_{\textsf{note}}^{\textsf{new}} = \mathrm{cm}_{\textsf{note}}^{\textsf{old}}$ with more than negligible probability we contradict the collision-resistance of CRH, since the Merkle roots in both transactions must be valid.

Since, there is no other case in which $\mathcal{A}$ can win, we have reached contradiction. We can now conclude that $\mathcal{A}$ cannot win the TL game with more than negligible probability. $\square$