Lindenbauer Dominik, BSc

Integrating Confidential Transactions into Distributed Ledger Technologies

Master's Thesis

to achieve the university degree of
Diplom-Ingenieur

submitted to
Graz University of Technology

Supervisor
Dipl.-Ing. Daniel Kales BSc.

Univ.-Prof. Dipl.-Ing. Dr.techn. Christian Rechberger

Institute for Applied Information Processing and Communications

Faculty of Computer Science and Biomedical Engineering

Graz, April 2020

Abstract

The interest in cryptocurrencies has significantly increased in the last couple of years, and with it also the various other applications of blockchains have become more prominent. One of the main differences of cryptocurrencies to conventional systems is that no dedicated authority, usually in the form of banks, is required. Distributed ledger-based systems are administered by their underlying data-structure, the *blockchain*.
Similar to account numbers, users are identified by a unique value called *address*. Users may transfer their assets to any other user by proposing transactions to the system, which can then be verified by anyone with access to the blockchain at a later point in time.
Due to the fact that the blockchain is public, a curious party can gather information about transferred amounts and balances of different users, even though their identities are unknown. Hence such systems trade the necessity of a central authority with the user's confidentiality.
In this thesis, we will describe how distributed ledger-based payment systems can be enhanced in a way such that no sensitive information needs to be stored on the ledger in plain text anymore.
We show how cryptographic primitives, such as encryption schemes and zero-knowledge proofs, can be used to construct a *confidential* transaction protocol. Our protocol then allows users to transfer their assets secretly to others without disclosing any sensitive information, while the fundamental concept of public verification is preserved.

Kurzfassung

Das Interesse an Kryptowährungen ist in den letzten Jahren immer mehr gestiegen, dabei wurden auch andere Anwendungsmöglichkeiten von Blockchains abseits von Währungen diskutiert. Eine der größten Unterschiede von Kryptowährungen zu konventionellen Währungssystemen ist, dass keine zentrale Verwaltungsbehörde, normalerweise in der Form von Banken, benötigt wird.
Systeme die auf einem verteilten Register beruhen verwalten sich eigenständig durch die Verwendung von *Blockchains*.
Ähnlich wie mit Kontonummern sind auch Nutzer in solchen Systemen durch einen eindeutigen Wert, hier *Adresse* genannt, identifiziert. Benutzer haben weiters die Möglichkeit Beträge an Dritte zu überweisen, indem sie Transaktionen an das System schicken. Diese können dann von jedem mit Zugriff auf das Register zu einem späteren Zeitpunkt verifiziert werden.
Infolgedessen, dass die Blockchain öffentlich erreichbar ist, kann ein Benutzer mit böswilligen Absichten sensible Informationen, wie Beträge und Kontostände, vom Register ableiten auch wenn die tatsächlichen Identitäten verborgen bleiben. In diesen Systemen wird also die Abwesenheit einer zentralen Verwaltung mit der Geheimhaltung von Nutzerinformationen getauscht.
In dieser Arbeit wird beschrieben, wie verteilte Währungssysteme verbessert werden können, sodass keine sensiblen Informationen mehr einsehbar sind. Es wird gezeigt wie kryptographische Primitive, wie Verschlüsselungen und Zero-Knowledge-Beweise, verwendet werden können um ein *geheimes* Überweisungsprotokoll zu definieren. Das entwickelte Protokoll erlaubt es Benutzern Beträge geheim an andere zu überweisen, während dabei das grundlegende Prinzip der öffentlichen Verifikation erhalten bleibt.

Contents

List of Figures

List of Tables

1 Introduction and Ledger Overview

Today there exist a lot of crypto-currencies such as *Bitcoin* [Nak08], *Ethereum* [But14], *Monero* and many more. The interest in these technologies has also significantly grown in the last couple of years, especially due to the increasing presence in the media.
In contrast to other currency systems, crypto-currencies do not require a central authority because all necessary information about the users balances is stored in an immutable data-structure. Taking proven solutions into account, the first choice for storing this information is almost always a *blockchain*.
Users may transfer amounts of assets to another party by proposing *transactions* to the system. For this purpose users are identified by a unique value called an *address*, similar to an account number. Another significant difference to *traditional* digital payment systems is that anyone who has access to the blockchain may verify new transactions. The blockchain is *distributed* among all users, so everyone holds a copy of it. A valid transaction will cause a state change, which needs to be broadcast to all participants, so that everyone is kept up to date. Anyone with access to the current state is then able to *replay* all occurred transactions to verify that everything is correct and consistent.
By this design an honest but curious party may be able to track a users activity even though the identities behind these addresses are unknown. At a later point in time an account may be linked to a person, which leads to a total break of privacy.
Another problem would be that if a user receives his salary by blockchain-transactions, his landlord is able to see/track this persons salary and as an effect may charge more rent.

A distributed-ledger based asset system consists of the following actors:

1. **Users**: Users hold assets and can transfer them to other users by proposing a transaction. A user is identified by a unique value called an address.
2. **Nodes**: Nodes verify the validity of these transactions, and will on success add them to the ledger. If the system allows it, users are also able to act as nodes.

The goal of this thesis is to describe a transaction protocol that enhances the confidentiality of users by disabling the possibility of deducing sensitive information with a focus on the transferred amounts.

1.1 Ledger Description

There exist multiple design options for a distributed ledger, which mainly differ in how transactions are handled and also in the information stored in every block. Standard transaction models are the *UTXO*[1] and the account model. The most prominent distributed ledger system using the UTXO approach is Bitcoin. Ethereum [Woo17] [But14] on the other hand is a representative using the second approach. In the following, we will briefly describe the ideas of both principles.

1.1.1 UTXO

Transactions in the UTXO approach are defined in the following way: they consist of transaction-inputs and transaction-outputs; outputs describe the receivers and inputs are references to previous outputs. The balance of a user is implicitly stored on the ledger and is defined by the sum of all unspent transaction outputs. A transaction output is considered unspent, if it has not been referenced by any transaction input. Users may only spend outputs which belong to them. Projecting this approach into the real world, transactions in the UTXO model could be seen as paper bill transactions. Every user possess a wallet holding multiple bills. When a user wants to spend money, he needs

[1]Unspent transaction output.

to use enough bills to cover the expense and may also receive some change in return.
To make this process more clear consider the following example: a user *owns* two outputs with a respective amount of 5 and 7. To now compose a transaction with an expense of 8 he declares both outputs as transaction-inputs and defines two new outputs. One of this outputs sends 8 *coins* to the recipient and the other one, which defines the change, defines the spending user as receiver.

1.1.2 Account Model

The account model, like used in Ethereum [But14], is a more intuitive approach because it is similar to how banks handle digital transactions. Transactions will be proposed to the verifiers, which will then check that the spending party's balance is large enough and that the receivers account exists. When all requirements are fulfilled, the authority will update the involved balances.
In this setting, a transaction could be seen as a global state change, where the balances of all users define the current state. In contrast to the UTXO model described before, the account model is more ore less an analogy to the credit/debit card system banks use. However, the bank setting requires a third party, which needs to be trusted by all users. In the distributed ledger setting, no central authority is needed because everyone will be able to check the integrity of the current state by replaying all occurred transactions.
For the rest of this thesis, we will consider the **account model** as the reference transaction approach[2].

1.1.3 Ledger

In the setting of a distributed ledger-based payment system, almost always, a *blockchain* is used to keep track of the current system-state. A blockchain is essentially a single linked list, where each entry or *block* holds a reference to the previous one in the list, as depicted in Figure 1.1. The list itself is designed in a way, s.t once a block has been inserted it cannot be deleted anymore without

[2]Keep in mind that all descriptions may be adapted to also work with a UTXO based transaction system.

destroying the list. Due to the immutability of the list, a blockchain may be used to store system information which is then protected against tampering by design. A more detailed explanation on how this data-structure can be utilized in the case of a payment system will be described in section 2.5

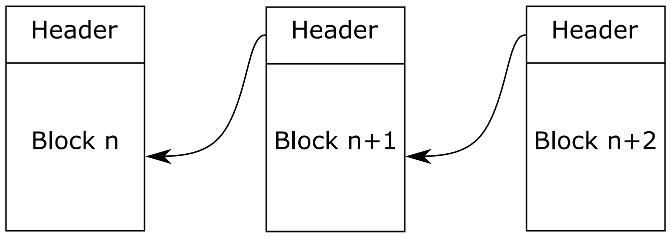


Figure 1.1: A common blockchain structure.

In the following we will shorty revisit the general parts of a distributed ledger-based payment system:

- To keep track of the current state a blockchain, as described above, is used.
- As previously stated a distributed ledger-based payment system defines two actors: Users and Nodes. Users are the actual owners of the assets, and nodes will take care of the transaction verifications.
- A transaction is a message, which proposes a state transfer. When Alice wants to engage an asset transfer with Bob, she will send a message saying: “I Alice want to transfer n units from my account to Bob’s”, to the nodes. Consider a more practical example:
  At the current state Alice holds 10 and Bob holds 5 units. Alice wants to transfer 5 of them to Bob; she will compose a transaction holding this information and sends it to the nodes. One of the nodes will verify that Alice has enough units and the claimed spending account is actually owned by her[3]. If the verification succeeds, the nodes will execute a state transfer by updating the balances of Alice and Bob.

[3]In addition also other checks may be necessary, which will be described in later sections.

1.1.4 Asset Transfer

Transactions are messages which transfer assets from the spending account to the receiver's. As with *conventional* digital money transfers the following conditions must hold:

- A user may only transfer as many assets as he currently posses. This statement is equivalent to the inequality $b \geq a$, where a denotes the amount to be transferred, and b denotes the current balance of the spender.
- The spender must prove the ownership of the sending account; otherwise, every user would be able to propose transactions from any other account.

User accounts are stored on the system; a valid transaction will lead to a state change by updating the involved balances. An illustration of the balance update, regarding the account model, can be found in Figure 1.2.

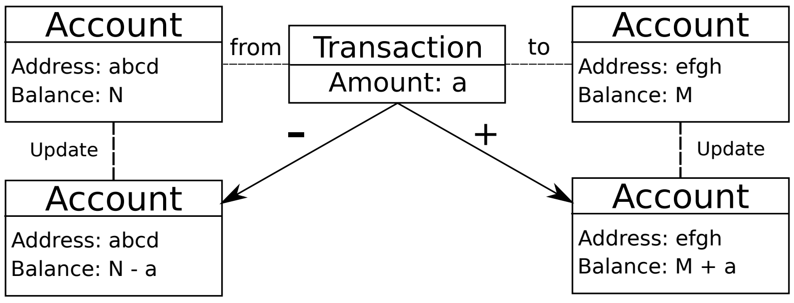


Figure 1.2: Balance update in the account model setting.

1.2 Changes for a Confidential Distributed-Ledger

With this basic structure described in section 1.1, we will now propose some necessary adaptions for a confidential distributed ledger-based payment system.

In the basic design, the amounts involved in a transaction are represented in plain form, which allows nodes to check the transaction validity. In a confidential setting no information about the amounts should be disclosed in any form, neither direct nor indirect, hence all sensitive information must be obfuscated, but should still be recoverable later. Due to this requirements one would quickly think about the use of encryption schemes. These schemes allow one to hide information in a sense that only parties holding a secret *key* will be able to recover the plain information again.

There exist two main type of encryption schemes: symmetric and asymmetric. In a symmetric setting the same key is be used during encryption and decryption. In the setting of a crypto-currency this would mean that every user would need to hold a symmetric key with every other user of the system; otherwise a user will not be able to verify the validity of a proposed transactions from another party, which clearly destroys the possibility of public verifiability. The effort of key exchange and key revocation makes this approach highly impractical for this setting. In addition every user would than be able to decrypt every amount of every transaction, which would have the same effect as using no encryption at all.
An asymmetric cipher on the other hand works as follows: the scheme is defined by two different keys, a private and a public part. The public part is used for encryption and the respective private counter-part is used during decryption. Such a scheme allows one to encrypt information for a dedicated recipient. This approach is way more practical compared to the symmetric approach, because every user only needs to hold one key-pair instead of a list of keys, which could grow really large.
When projecting this idea into our setting, this would mean that a spender will be able to encrypt the transferred amount with the receivers public-key, but another significant problem arises: with this approach only the receiver will be able to decrypt the amount again, which further destroys the paradigm of public

verification. Also the users balances will not be updatable without decryption. A transaction protocol should preserve the ability of public verifiability, while at the same time no decryption during the process is required.

Even though this idea looks quite impractical at first glance, there exist cryptosystems which suite these requirements. They allow one to perform defined operations on encrypted values, yielding valid encryptions of the operation performed on the plain values, without the necessity of decryption. Encryption schemes which allow this fall into the group of **homomorphic**-encryption schemes. A more detailed explanation of these schemes and why they are useful in our setting, will be presented in later parts of this thesis.

The amounts are now represented in a way where no information about the plain value can be deduced anymore, while still allowing dedicated operations. The use of a homomorphic public-key encryption scheme solves the problem of sensitive information disclosure, but creates a new one: by definition cipher-texts do not leak anything about the enclosed plain values, so a classic verification without decryption will not be possible anymore.
To still allow public verification of transactions, encryption must be combined with another cryptographic primitive that allows one to prove the truth a statement, while the secret remains hidden.

Proof of Knowledge We previously stated that encryption alone will not be sufficient enough to fulfill all of the predefined requirements. In addition to an encrypted amount, also cryptographic proofs need to be present in a transaction. The necessity of such proofs is a significant difference compared to the basic design of a distributed ledger-based payment system.

Recap the actual verification process: the integrity of the ledger is preserved by validating that the spender is actually in possession of the assets and that the spending account's balance is large enough.
In the classical design, the balance verification is trivial, because all amounts are available in clear text. When considering the confidential design, the nodes will no longer be able to check this property by performing a simple comparison. As a result another way of describinp and verifying these properties needs to be found. This is where proof systems come into play.
A cryptographic proof-protocol consists of two parties: the prover and the

verifier. The prover wants to convince the verifier that a given statement is true without disclosing any other information than the truth of the statement.
Appropriate proof-schemes can be used to prove statements of arbitrary complexity, which will fit our requirements. By utilizing such proof-schemes, the spender can prove that the spent amount is smaller than his current balance, without disclosing these values.
The basic approach of these schemes is usually built on the challenge-response paradigm, which requires interaction of both parties. The possibility of verification at a later point in time should be preserved, so the used protocol must be non-interactive, otherwise spender and receiver need to be online at the same time.
However the chosen proof system should be capable of efficiently expressing all required statements; in our case non-interactive Σ-protocols will be used, which will be described in subsection 2.9.1.

1.3 Contributions of this Thesis

In this thesis, we will describe how homomorphic cryptosystems, such as ElGamal [ElG85] and Pedersen-commitments, [Ped92] can be used in combination with zero-knowledge protocols to describe a confidential transaction protocol. We used self-developed and established Σ-protocols to define a confidential transaction protocol that satisfies all of the required constraints. For integrity protection, we defined a scheme that builds upon bulletproofs [Bün+18], and ensures that no malicious transaction will result in a valid verification.To verify the utility of our protocol, we also provide a library implementation in C/C++.

1.4 Outline

The upcoming parts of this thesis are structured in the following way: in chapter 2 we will discuss the cryptographic background; in this context also all necessary primitives will be presented. Chapter 3 defines the general system framework on which the protocol specification presented in chapter 4 will be based on.
In chapter 5 we present our implementation results, regarding the computational overhead compared to a *basic* transaction protocol. The last chapter of this thesis concludes our work and describes which directions future work could look into.

2 Preliminaries

In this section, we will discuss all necessary cryptographic primitives and their preliminaries.

2.1 Abstract Algebra

Some of the upcoming primitives will build upon concepts of abstract algebra. In the following we will describe the basic parts:

Definition 2.1.1 (Group)**.** A Group $\mathbb{G}$ is a set which is closed under an operation $\times$, so $\forall a, b \in \mathbb{G} : a \times b \in \mathbb{G}$ holds. Additionally a group satisfies the following properties:

Associative : $\forall a, b, c \in \mathbb{G} : a \times (b \times c) = (a \times b) \times c$
Identity: $\exists e \in \mathbb{G}, \forall a \in \mathbb{G} : a \times e = a.$
Inverse: $\forall a \in \mathbb{G}, \exists a^{-1} \in \mathbb{G} : a \times a^{-1} = e.$

The "power" g^n, for an element $g \in \mathbb{G}$, is defined by applying the group operation n times to g e.g. $g^n = \underbrace{g \times g ... \times g}_{\text{n times}}$.

Consider a practical example: Set $\mathbb{G} = (Z_{11}^*, \cdot)$, with $Z_{11}^* = \{1, ..., 10\}$ denoting the set of integers modulo 11 without 0, and $g = 2 \in \mathbb{G}$. By applying the group operation multiple times to g, we observe that every other element $x \in \mathbb{G}$ can be represented as such a power: $2 = 2^1, 3 = 2^8 \pmod{11}, ..., 10 = 2^5 \pmod{11}$. The order of a group $\mathbb{G}$, denoted as $|\mathbb{G}|$, describes the number of elements in the group. Following this a **cyclic** group is defined as:

Definition 2.1.2. A Group $\mathbb{G}$ is called cyclic if there exists an element $g \in \mathbb{G}$ s.t. every element $x \in \mathbb{G}$ can be represented as a power of g.

For our purposes we will need to operate on groups of large prime order p. Furthermore a group is called an abelian group if $a \times b = b \times a$ for every $a, b \in \mathbb{G}$ holds. Additionally consider the definition of a field $\mathbb{F}$:

Definition 2.1.3 (Field)**.** A field $\mathbb{F}$ is a set which is closed under two operations $\cdot, +$. Additionally $\mathbb{F}$ satisfies the following properties:

1: $\mathbb{F}$ is an abelian group under $+$
2: $\mathbb{F} \setminus \{0\}$ is an abelian group under $\cdot$

According to the field definition a **finite** field is defined as follows:

Definition 2.1.4 (Finite-Field)**.** A field $\mathbb{F}$ which contains finitely many elements.

The number of elements in a field is always a prime power p, denoted as $\mathbb{F}_{p^n}$. A **prime** field denoted as $\mathbb{F}_p$, then denotes the field of congruence classes of integers modulo p, with the set $\{0, ..., p-1\}$ describing the elements of $\mathbb{F}_p$. Finite fields serve as fundamental building blocks in the field of cryptology. For example, they are used to define elliptic curves but also in the context of other fundamental cryptographic primitives.

2.2 Elliptic Curve Cryptography

With the previous definitions in mind, we will now explain the very basic concepts of elliptic curves in a cryptographic setting.
An elliptic curve $\mathbb{E}$, defined over a prime field $\mathbb{F}_p$, is the set of solutions $(x, y) \in \mathbb{F}^2$ with $x, y \in \mathbb{F}_p$, of equations of the the Weierstrass-form: $y^2 = x^3 + ax + b$. A point on $\mathbb{E}$, is then defined by the tuple (x, y). According to the previous definition of a generator, a point P is called a *base* point of $\mathbb{E}$ if every other point $Q \in \mathbb{E}$ can be represented as a multiple of P, $Q = nP$. The neutral element, in terms of addition, is defined as $\mathcal{O} = (0, \infty)$, so $P + \mathcal{O} = P$ holds[1]. Analogously an *inverse* of a point is defined as follows: $P + P^{-1} = \mathcal{O}$.
Elliptic curves allow one to adapt a scheme defined in a group setting to an

[1]$\mathcal{O}$ is often referred to as the point at infinity.

elliptic curve variant while preserving the security guarantees implied by the scheme. Given a power of generator $g^n \in \mathbb{G}$, the elliptic curve adaption would be described by nP where P denotes the base point. Considering the element multiplication in a group setting with $q = g^a, r = g^b \in \mathbb{G}$ and $qr = g^{a+b}$, the elliptic curve variant is described as: $Q = aP$, $R = bP$, $Q + R = (a + b)P$. Additional details and evaluations can be found at [Kob87].

2.3 Diffie-Hellman Problem

Some of the upcoming security guarantees will be based on the **Decisional-Diffie-Hellman Assumption**, which is described in the following setting: let $\mathbb{G}$ be a cyclic group of order q with generator g. Further let x, y, $z \in \mathbb{Z}_q$ be exponents of g, yielding the triple (g^x, g^y, g^z). The DDH-Assumption (Decisional-Diffie-Hellman Assumption) is then defined as: given this triple it should be infeasible to decide if $g^z = g^{xy}$ holds. It is obvious that one would succeed with probability 1/2 by simple guessing. Hence the DDH-Assumption holds if there exists no p.p.t. (probabilistic polynomial time) algorithm which decides this problem with non-negligible probability better than 1/2.
The DDH-problem is related to the discrete logarithm problem, **DLP**, in cyclic groups which is defined as: given a cyclic group $\mathbb{G}$ with generator g and order q and an element $b = g^x \in \mathbb{G}$, a value y solving the equation $b = g^y$ is defined as the discrete logarithm of b. Finding such a value y becomes significantly harder with a growing group order, and further becomes infeasible if q is a large prime. In an elliptic curve setting this problem will be defined as **EC-DLP**: given a base point of $\mathbb{E}$ as P and another point $Q \in \mathbb{E}$ find the smallest integer n for which $Q = nP$ holds.
The DDH problem is also related to the **computational**-Diffie-Hellmann Assumption, CDH, which is defined as: given two values g^a and g^b compute g^{ab}. We observe that if the CDH problem can be efficiently solved, also the DDH assumption does not hold.
Both problems are also related to the DLP: if there exists a p.p.t. algorithm which calculates $x = \log_g g^x$ also the CDH can be solved by calculating $x = \log_g g^x$ and setting $(g^y)^x$, which further also breaks the DDH.
So we know that if the DLP can be solved also the CDH and DDH can be solved, and if the CDH can be efficiently decided also the DDH can be decided.

Consider the following notation: $A \leq_p B$, describing that if there exists a p.p.t. algorithm which solves the problem B also an algorithm solving A can be found. We say that the computational complexity of A can be reduced to B, so A is *not* harder then B.
With this notation an *order* of the above problems can be described: DLP $\geq_p$ CDH $\geq_p$ DDH.
When the DDH is considered to be hard, also the CDH and DLP are considered hard. The hardness of the DDH problem is thus the **strongest** security assumption in cyclic groups.

2.4 Cryptographic Hash Function

A hash function maps a message input of arbitrary length to a fixed length output space. The output will then serve as a *fingerprint* for the given message. More formally a hash function is defined as: $\mathcal{H} : \mathbb{F}_2^* \rightarrow \mathbb{F}_2^t$, $\mathcal{H}(M) = T$. For hash functions to be used in a cryptographic setting, additionally the following properties must hold:

1. **Preimage Resistance:** given $\mathcal{H}(M) = T$, it should be infeasible for an attacker to calculate M from T.
2. **Second Preimage Resistance:** given a message M, it should be infeasible for an attacker to find a second message $M' \neq M$, s.t. $\mathcal{H}(M) = \mathcal{H}(M')$ holds.
3. **Collision Resistance:** it should be infeasible for an attacker, to find two distinct messages $M \neq M'$, s.t. $\mathcal{H}(M) = \mathcal{H}(M')$ holds.

Hash functions that fulfill the above properties are called **cryptographic-hash-functions**. Only algorithms that follow the standard presented in [NIS15], such as the SHA-2 algorithm family, should be used for cryptographic purposes.
The field of application for this primitive is widely spread; it ranges from message authentication and integrity protections to serving as building blocks for data structures such as blockchains and Merkle-trees [Mer88].

2.5 Blockchain

As shortly described during the introduction, blockchains suite the requirements of a distributed ledger-based payment system very well. In this section, we will explain this structure in more detail and describe how it can be utilized.
A blockchain is a single linked list, where each entry or *block* holds a reference to his predecessor. Cryptographic hash-values of the entries implicitly define the links; a primitive such as SHA-256, which fulfills the fundamental properties defined in section 2.4, should be used for the hash-value calculation.
The design ensures that once a block has been inserted, it may not be removed without destroying the list. To replace an entry, one would need to update all links in the following blocks. Due to the second-preimage-resistance property, it is also infeasible to find a different entry that hashes to the same value. If such entries could be found, an attacker would be able to tamper with the list, because a block could be swapped with a malicious one without notice.
New blocks will be appended to the end of the list; hence the data-structure also keeps track of all occurred insertions. Once an entry has a predecessor and a successor, an insertion cannot be deleted anymore.

```
block{
    header,
    block_content
}
```

A block consists of meta information, here described as *header* and the actual content. In the most trivial case the field header only holds a hash-value of the previous block, which further describes the link to its predecessor. More sophisticated designs may also include additional information, like the solution to a hash puzzle as described in [Nak08].
In the following we will explain how such a data-structure could be used in the setting of an asset system: a payment system's state is defined by all users balances, a blockchain can hence be used to administer this information, by storing the necessary system information in blocks. By design such a blockchain then describes the state history of the system, with the last block defining the current state. In the setting of a payment system a block may hold a list of transactions which caused the state changes. A users balance is then defined by counting up all ever received and spent transactions. Due to the *immutability*

of a block-chain, users will not be able to deny a spent or received transaction. When all transactions are saved blocks, anyone with access to the chain may verify the current state and it's integrity, by replaying all occurred state transfers. Each user may hold a copy of this chain to perform the sanity checks. When a new state change occurs the new information will be broadcasted all users, which will check the validity of the new state. On success the respective copies are updated by appending the new information to the chains.
A blockchain may hence be a very useful data-structure in the setting of a payment system, because it supports administration of the world-state without a central authority. Users may check the validity of the current state at any time, by replaying the stored state-history.

2.6 Public-Key Encryption

Encryption is a very useful tool in the fields of cryptography; it allows communicating parties to keep their information secret by making sure that an eavesdropping party will not be able to deduce any sensitive information from the messages. In a symmetric setting the same secret information, called a key, will be used to transform the plain messages. Only the parties holding the key will then be able to decipher these messages again. As described previously, such a symmetric approach would be highly impractical in the setting of a distributed ledger-based payment system.
The asymmetric approach on the other hand, works as follows: each party holds a *key-pair*, which consists of a public and a private part. The public part will be announced, so that every other party of the system will be able to use this information. The secret part, as implied by the name, will be kept secret. Usually the public-key will be calculated by evaluating a function which depends on the secret information. Each party of the system can then encrypt messages for any other user's public-key. Only the party holding the corresponding private key will then be able to decipher these messages again.
[RSA78] defined the properties of a public-key cryptosystem:
Each party announces a public encryption procedure E, and keeps the corresponding public procedure D secret. Given these two procedures the following holds:

1. Applying D to $E(M)$, yields the plain message M. $D(E(M)) = M$
2. E and D are defined by *non* complex algorithms.
3. Announcing E will not yield any sensitive information about D.

E and D are general methods, $E(M)$ can be calculated by using the public encryption key. D, on the other hand, can only be used to obtain M form $E(M)$ by providing the secret key. So the security of D depends on keeping the private key secret.

2.6.1 ElGamal

The homomorphic-encryption scheme, ElGamal[ElG85] satisfies our previously required properties and is defined as follows:

Definition 2.6.1 (ElGamal)**.** ElGamal is an asymmetric key encryption scheme which is based on the Diffie-Hellman key exchange [DH76] and is defined over a cyclic group $\mathbb{G}$. The scheme consists of three algorithms: *KeyGen*, *Encrypt*, *Decrypt*.

$\mathsf{KeyGen}(1^k)$: let $\mathbb{G}$ be a cyclic group of order q with generator g. On input of security parameter k return a key-pair with $s_k \in \mathbb{Z}_q$ as private key and $h := g^{s_k} \in \mathbb{G}$ as public key. Publish the tuple $(\mathbb{G}, g, q, h)$ as public information and keep s_k secret. This algorithm is performed once by the receiving party.

$\mathsf{Encrypt}(\mathsf{m}, \mathsf{h})$: on input of a message $m \in \mathbb{G}$ and public encryption-key h compute the cipher-text as follows: sample a random element $y \xleftarrow{\$} \mathbb{Z}_q$ and set the shared secret to $s := h^y$. Compute $c_1 := g^y$ and $c_2 := ms$, the cipher-text is then described by the tuple (c_1, c_2).

$\mathsf{Decrypt}((\mathsf{c}_1, \mathsf{c}_2), \mathsf{s}_\mathsf{k})$: on input of cipher-text tuple and secret key s_k, decryption is performed as follows: given c_1 and s_k, the decrypting party is able to calculate the shared secret as: $s = c_1^{s_k} = g^{s_k y} = h^y$. The plain-text $m \in \mathbb{G}$ is then extracted by computing $m = c_2 s^{-1} = mss^{-1} = m$.

Remarks It is easy to see that an encryption under a public key $h = g^{s_k}$ can only be decrypted by a party holding the corresponding secret key s_k. Considering this property, one would be able to break this scheme by calculating

s_k from h. As described above this is equivalent to solving the discrete logarithm problem, which is considered to be infeasible.
In addition a fresh value $y \xleftarrow{\$} \mathbb{Z}_q$ should be used for every encryption under the same public-key, otherwise the cryptosystem would not be considered secure due to the following property: given a cipher-text $C = (c_1, c_2)$ and the corresponding plain-text $m \in \mathbb{G}$, an adversary would be able to obtain the shared secret, for this encryption, by computing $s = c_2 m^{-1} = smm^{-1}$. If y is reused, the shared secret s will be the same for more than one cipher-text. Following this an attacker would be able to retrieve the corresponding plain-text from any other cipher-text encrypted under the same y and h. Hence it is essential to sample a *fresh* value y for every encryption, in this context y is also called an *ephemeral-key*. Otherwise a known plain-text attack would allow an attacker to break every encryption performed under this public-key.

2.6.2 Elliptic Curve ElGamal

To obtain the elliptic curve variant of the scheme described in subsection 2.6.1, the following adaptions need to be performed:

- Replace cyclic group $\mathbb{G}$ by a suitable elliptic curve $\mathbb{E}$.
- All group operations need to be replaced by equivalent elliptic curve operations.

Taking this adaptions into account we obtain the following definition for an elliptic-curve-based ElGamal variant:

Definition 2.6.2 (EC-ElGamal)**.** As the cyclic group variant, also EC-ElGamal consists of three algorithms: *KeyGen*, *Encrypt*, *Decrypt*.

$\mathsf{KeyGen}(1^k)$: let $\mathbb{E}$ be an elliptic curve, defined over a prime field, with order $\mathcal{N}$ and base point P. On input of security parameter k return a key-pair with $s_k \in [1, \mathcal{N}-1]$ as the private-key and publish $Y := s_k P$ as the public key.

$\mathsf{Encrypt}(\mathsf{M}, \mathsf{Y})$: on input of a message $M \in \mathbb{E}$ and public encryption-key Y compute the cipher-text as follows: sample a random element $z \in [1, \mathcal{N}-1]$ and set the shared secret to $s := zY$. Compute $c_1 := zP$ and $c_2 := s + M$. Return the cipher-text as tuple (c_1, c_2).

$\mathsf{Decrypt}((\mathsf{c}_1, \mathsf{c}_2), \mathsf{s}_\mathsf{k})$: on input of cipher-text tuple (c_1, c_2) and secret key s_k, decryption is performed as follows: compute $s = c_1 s_k$. c_1 is calculated as zP and s is defined as $zY = zs_kP$, hence the decrypting party is able to calculate the shared secret by using the private key and c_1. The plain-text $M \in \mathbb{E}$ is retrieved by computing $M = c_2 - s = s + M - s = M$.

The elliptic curve variant of ElGamal behaves in the same way as the scheme defined in subsection 2.6.1. According to the cyclic group setting, this adaption would not be considered secure if there exists a p.p.t. algorithm that efficiently computes s_k from Y. Therefore the security of this scheme depends on the *elliptic-curve*-discrete-logarithm problem.

2.6.3 Homomorphic Property of Elliptic Curve ElGamal

We will now show how the elliptic curve adaption for this cryptosystem can be used to fulfill our previously defined requirements. Consider two cipher-texts encrypted with EC-ElGamal under the same public-key, yielding cipher-texts $C_1 = (z_1P, z_1Y + M_1)$ and $C_2 = (z_2P, z_2Y + M_2)$. Given this the following property holds:

$$\begin{aligned} C_1 + C_2 &= (z_1P, z_1Y + M_1) + (z_2P, z_2Y + M_2) \\ &= ((z_1 + z_2)P, (z1 + z_2)Y + (M1 + M_2)) \\ &= Encrypt(M_1 + M_2) \end{aligned} \tag{2.1}$$

Given two cipher-text, encrypted under the same public-key, it is possible to *calculate* a valid encryption of the sum of the corresponding plain-texts by performing the operation described above.
Considering the requirement that no plain information should be present during the protocol, this scheme can be utilized in the following way: given the encrypted balance of the receiver as b and the encrypted amount as a, both encrypted under the receiver's public key, the balance-update can be computed as $b \oplus a$, where $\oplus$ denotes the operation defined in Equation 2.1. The receiver will then be able to decrypt its balance with the corresponding private key[2].

[2]The same holds for the spender by using the inverse operation $\ominus$.

Remarks The encryption scheme ElGamal is only capable of encrypting values in the message space, which is defined by the underlying group or elliptic curve, so messages need to be represented accordingly.

2.7 Commitment Scheme

Commitments are cryptographic primitives which allow one to commit to a secret, by computing a value called a *commitment*, which depends on the secret but simultaneously does not leak anything about the plain value. Once committed it should not be possible to change the commitment without destroying it. The secret can then be revealed at a later point in time by providing a so called opening, which are essentially the parameters used to calculate the commitment in the first place. The verifier will then use this information to check if the commitment is correct, by recomputing it. Commitments need to fulfill two important cryptographic properties: Hiding and Binding.

Hiding means that no information, whether direct nor indirect, about the secret value can be deduced from the announced commitment. Binding on the other hand means that once committed it should not be possible to change the secret value after the announcement.

Seen in a more abstract way, one could interpret a commitment as a locked box where a sheet of paper holding the secret is placed in. The box itself will not leak any information about the secret and no-one will be able to change it without destroying the box. In this abstraction the corresponding key will be interpreted as the opening, which will be handed out to the verifier at a later point in time. The verifier is then be able to check the claimed statement of the committer after he has opened the box.

2.7.1 Pedersen Commitment

A Pedersen-Commitment provides the desired properties described above and was first presented in [Ped92].

Definition 2.7.1 (Pedersen Commitment)**.** Pedersen commitments allow one to commit to a secret value and reveal it at a later point in time, while preserving the secrecy and integrity of the committed value. This primitive is defined by three algorithms: *Setup*, *Commit*, *Open*:

$\mathsf{Setup}(1^k)$: given a cyclic group $\mathbb{G}$ of prime order p select $g, h \leftarrow \mathbb{G}$ as generators of group $\mathbb{G}$. Announce g, h, G_p as public parameters, such that no-one knows $log_g h$.

$\mathsf{Commit(m, g, h)}$: on input of secret m, set $\gamma \xleftarrow{\$} \mathbb{Z}_p$ and calculate the commitment as $C \leftarrow g^m h^\gamma$. Set the opening to $O \leftarrow (m, \gamma)$. Return O and C

$\mathsf{Open(C, O)}$: on input of commitment C and corresponding opening O, parse O as (m, γ). Calculate $C' \leftarrow g^m h^\gamma$ output m if $C = C'$ holds, otherwise return $\perp$.

Remarks Pedersen commitments are computational-binding due to the following property: suppose the committer is able to find two pairs (x, γ), (x', γ') s.t. $g^x h^\gamma = g^{x'} h^{\gamma'}$, with $x \neq x'$ holds, the committer is able to compute $log_g h$, because $a = (x - x')/(\gamma - \gamma')$ holds. So the binding property depends on the DLP which is considered to be infeasible.
This scheme is also information-theoretic hiding: given x, γ, x' there exists an γ' s.t. $g^x h^\gamma = g^{x'} h^{\gamma'}$ holds. Due to the properties of the underlying group the probability that either x or x' have generated C is identical.

2.7.2 Elliptic Curve Pedersen Commitment

As with ElGamal the plain version of this commitment scheme can be implemented in an elliptic curve setting by applying the same adaptions as described in Definition 2.6.2. Following this an EC-Pedersen-Commitment is described as follows:

Definition 2.7.2 (EC-Pedersen-Commitment)**.** Like the plain form, the elliptic curve variant defines three algorithms: *Setup*, *Commit*, *Open*:

$\mathsf{Setup}(1^k)$: given an elliptic curve $\mathbb{E}$ over a prime field of order q, sample two points $G \xleftarrow{\$} \mathbb{E}$ and $H \xleftarrow{\$} \mathbb{E}$ and announce them as public parameters, s.t. the no-one knows the EC-DLP of H.

$\mathsf{Commit}(\mathsf{x}, \mathsf{g}, \mathsf{h})$: on input of secret x, choose $\gamma \xleftarrow{\$} \mathbb{Z}_p$ and calculate the commitment as $C \leftarrow mG + xH$. Set the opening to $O \leftarrow (m, x)$. Return O and C

$\mathsf{Open}(\mathsf{C}, \mathsf{O})$: on input of commitment C and corresponding opening O, parse O as (m, x). Calculate $C' \leftarrow mG + xH$ output m if $C = C'$ holds, otherwise return $\perp$.

Remarks Pedersen commitments have a similar structure as Definition 2.6.2, hence this primitive also defines a homomorphic property. Given two Pedersen commitments, C_1 and C_2, calculated with the same generators G and H, yielding $C_1 = x_1G + \gamma_1H$ and $C_1 = x_2G + \gamma_2H$ the following holds:

$$\begin{aligned} C_1 + C_2 &= (x_1G + \gamma_1H) + (x_2G + \gamma_2H) \\ &= ((x_1 + x_2)G + (\gamma_1 + \gamma_2)H) \\ &= Commit(x_1 + x_2) \end{aligned}$$

The structure of a Pedersen-commitment allows one to homomorphically calculate a valid commitment to the sum of the plain values x_1, x_2 without knowing neither opening to C_1 nor C_2.

2.8 Discrete Logarithm Recovery

Numeric values can not be used as direct message input to the encryption scheme described in subsection 2.6.1. Hence the actual message input for a numeric value $a \in Z_p$ will be defined as $M = g^a \in \mathbb{G}$. The actual secret is then the value a. Calculating a from M and g is equivalent to solving the discrete logarithm problem, which provides a significant security guarantee of the described primitives. So the decrypting party needs to break this guaranty in order to recover the actual numeric value.

The discrete logarithm problem grows in difficulty, depending on the order of the group, which is usually a *huge* large prime, and the actual exponent x of g^x. Primitives like subsection 2.6.1 are only considered to be secure, regarding

the DLP if the *exponent* is significantly large because then the recovery of x becomes infeasible.
The calculation of the discrete logarithm will become feasible if the exponent is *small* enough. Note that an attacker will still not be able to recover the message from a corresponding cipher-text, under the condition that the primitive is used correctly. Solving the DLP is hence only reasonable if M is already present. For the general case, this means that only the deciphering party needs to solve the *small* DLP in order to recover the numeric secret. A malicious party will only be able to recover the secret if also the message can be recovered, which further means that the attack can solve the general DLP, and the encryption itself is no longer considered secure.
In the following section, we will present an algorithm that efficiently computes *small* discrete logarithms.

2.8.1 Babystep-Giantstep

In this section we will present an algorithm which is capable of solving the DLP: $x = \log_g a$, with $g^x = a$ for small x in reasonable time. The algorithm is defined by a space-time trade-off, and a meet in the middle approach.
First set $m := \lceil\sqrt{n}\rceil$, with n describing the maximum recoverable value. Second calculate a list of *baby-steps* as: $\forall j \in \{0, ..., m-1\}$ calculate g^j and save the tuple (j, g^j) in an appropriate data-structure.
The second step is defined as follows: $\forall i \in \{0, ..., m-1\}$ calculate $\sigma = ag^{-mi}$, as *giant-step*. For every intermediate σ search for a match $g^j = ag^{-mi} = g^x g^{m-i}$, if found return $x = i \cdot m + j$. The algorithm is dominated by the construction of the list holding the *baby-steps*, yielding a runtime of $\Theta(\sqrt{n})$ and a memory consumption of $\Theta(\sqrt{n})$ For further evaluations, and improvements we refer to [BL12]. Algorithm 1 depicts a listing of the described algorithm.

2.9 Zero-Knowledge

As described multiple times in this thesis, the transaction protocol will operate on hidden values. No information about the plain values will be present, hence a

Algorithm 1: Babystep-Giantstep

```
Data: $a \in \mathbb{G}$, $n \in \mathbb{N}$ upper bound in $|G|$,  $g \in \mathbb{G}$ as generator of the group
Result: return $\log_g a$ if $0 \leq a \leq n$, $\perp$ otherwise
$m := \lceil \sqrt{n} \rceil$;
baby_steps[m] = empty;
for $j$ in $\{1, ..., m-1\}$ do
    insert $g^j$ into baby_steps;
for $i$ in $\{1, ..., m-1\}$ do
    $\sigma = ag^{-mi}$;
    if found $\sigma$ in baby_steps[j] then
        return $i \cdot m + j$;
return $\perp$;
```

cryptographic protocol which proves knowledge of these values, while revealing nothing else besides the truth of the statement, needs to be used.

To provide a better understanding of the concept behind these schemes, a well known practical example called the Ali Baba Cave, first presented in [Qui+89], will be be presented in the following.
Imagine a round-shaped cave with only one entrance. At the inside there are two paths connected through a magic door in the middle, for convince these paths will be called A and B. The magic door only opens if the person standing in front of it knows a secret passphrase.
A Person called Peggy wants to prove to Victor that she knows this passphrase without actually telling it to Victor. For Victor to be convinced the following protocol will be initiated by Peggy: Peggy enters the cave and either takes path A or B, after that Victor enters the cave s.t. he has eyes on both paths. Victor then asks Peggy to come out at either path A or B, here two cases have to be distinguished:

- Peggy took a path, and Victor asks her to come out at the **same** one.
- Peggy took a path, and Victor asks her to come out at the **other** one.

Looking at case one we see that Peggy actually does not need to know the secret, because she can simply turn around and will appear at the requested path. In the other case Peggy actually needs to know the passphrase, otherwise

she will not be able to appear at the correct path. We realize that a cheating prover (one who does not know the passphrase) will succeed in this setting with probability $1/2$.
To actually convince Victor that Peggy really knows the secret passphrase, the protocol needs to be repeated k times, which reduces the success probability for a cheating prover to $(1/2)^k$. A significantly large choice of k will turn this probability negligible small.

Now that the general idea of these schemes has been presented, we want to concretize their conditions: proof-systems are defined over two actors, *prover* $\mathcal{P}$ and *verifier* $\mathcal{V}$, and have the following properties:

Completeness: if the provided statement is true and both parties follow the protocol accordingly, a honest verifier will always be convinced of this fact by a honest prover.

Soundness: if the provided statement is false, no cheating prover will be able to convince a honest verifier, except with negligible probability ϵ, that the given statement is true.

Zero-Knowledge: if the provided statement is true, a cheating verifier will not be able to learn anything more besides this fact. More generally: all information which can be observed by the *cheating* verifier could have been generated by its own, without interacting with the honest prover at all.

Proof schemes which provide these properties are called **Zero-Knowledge-Protocols**, which were invented by [GMR89]. In the following we will examin a well understood protocol, first presented in [Sch91], which provides the above properties. The protocol has been discussed in [Sch19] and consists of three messages: *announcement*, *challenge* and *response*, an illustration of the scheme is depicted in Figure 2.1.

First we want to discuss the soundness property of the protocol. A cheating prover will be able to convince a honest verifier by correctly guessing the challenge value c even though he does not know the secret value x. Therefore he needs to prepare the announcement in advance, such that response r will be accepted. Considering the challenge set $\{0, 1\}$, the prover prepares the announcement in the following way: for $c = 0$ he sets $a = g^u$ and $r = u$, for $c = 1$ the announcement is calculated as $a = g^u/h$ and the response is set to $r = u$. In both cases the announcement does **not** depend on the actual secret

$$\begin{array}{ccc}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
\hline
(x = \log_g h) & & \\
u \in_R \mathbb{Z}_n & & \\
a \leftarrow g^u & \xrightarrow{a} & \\
 & & c \in_R \{0,1\} \\
 & \xleftarrow{c} & \\
r \leftarrow_n \begin{cases} u, & \text{if } c = 0 \\ u + x, & \text{if } c = 1 \end{cases} & \xrightarrow{r} & g^r \overset{?}{=} \begin{cases} a, & \text{if } c = 0 \\ ah, & \text{if } c = 1 \end{cases}
\end{array}$$

Figure 2.1: Schnorr's protocol: Proof of discrete logarithm.

value x. We observe that a cheating prover will only be able to prepare for one of this cases at a time; hence the verifier will be convinced with probability 1/2.

Theorem 1 (Soundness)**.** If a prover is able to calculate a valid response r for both cases $c = 1$ and $c = 0$, after the announcement a has been revealed, then he must now the secret value x.

Proof. A prover is able to calculate two valid responses r_0, r_1 for challenge $c = 0$ and $c = 1$, respectively. These values simultaneously satisfy $g^r_0 = a$ and $g^{r_1} = ah$, leading to $h = g^{r_1 - r_0}$, which implies that the prover knows x, since $r_1 - r_0 = x \pmod{n}$ holds. □

Second we take a look at the zero-knowledge property of this scheme. In this context zero-knowledge means that a cheating verifier will not be able to obtain any information besides the truth of the statement.
This property can be examined by defining a p.p.t. algorithm that allows the verifier to *simulate* valid conversations. The simulator does not require the secret value, but the resulting conversations will follow the same probability distribution as *real* ones, which further means that an observer will not be able to distinguish them. A malicious verifier will thus not be able to extract anything about the proven value form a transcript, because all present information could have calculated by the verifier itself. In other words this means that no information about the secret is disclosed during the protocol.

We now define two p.p.t. algorithms, one for real conversations and one for simulations.

Real Conversations:
Input: secret x

1. $u \xleftarrow{\$} Z_n$
2. $a \leftarrow g^u$
3. $c \xleftarrow{\$} \{0,1\}$
4. $r \leftarrow u + cx$
5. return (a, c, r)

Simulated Conversations:
Input: public information h

1. $c \xleftarrow{\$} \{0,1\}$
2. $r \xleftarrow{\$} Z_n$
3. $a \leftarrow g^r h^{-c}$
4. return (a, c, r)

Theorem 2. *Honest-Verifier*-Zero-Knowledge An honest verifier (one that follows the protocol), would be able to produce valid conversations by executing the simulator-algorithm, which would then be indistinguishable from real conversations.

Proof. We observe that the *simulator* can produce valid conversations because the constraint g^r is satisfied. c and r are selected uniformly at random; therefore, the announcement a is random, and hence the conversation is random. c and u are selected uniformly at random during the real scenario; thus, r and a are also random. Both conversations are, therefore, indistinguishable. □

For our purpose, the *honest-verifier*-zero-knowledge property is sufficient enough for further details on the zero-knowledge property we refer to [Sch19] and [Sch91].
We observe that the scheme described in Figure 2.1 allows a cheating prover to convince a honest verifier with probability 1/2 of a false statement. Taking the explanation at the beginning of this section into account, the success probability for a cheating prover can be significantly reduced by engaging the protocol many times.
This approach clearly produces a large execution overhead. The basic scheme, illustrated in Figure 2.1, can be adapted in a way, s.t. the probability of convincing the verifier of a false statement is negligible small, while only one protocol execution is required. The following illustration describes the necessary adaptions:

$$
\begin{array}{ccc}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
\hline
(x = \log_g h) & & \\
u \xleftarrow{\$} \mathbb{Z}_n & & \\
a \leftarrow g^u & \xrightarrow{\quad a \quad} & \\
 & & c \xleftarrow{\$} \mathbb{Z}_n \\
 & \xleftarrow{\quad c \quad} & \\
r \leftarrow_n u + cx & \xrightarrow{\quad r \quad} & g^r \overset{?}{=} ah^c
\end{array}
$$

Figure 2.2: Schnorr's identification protocol.

The improved scheme enlarges the challenge space. The challenge c will then be selected at random. A cheating prover will thus only succeed with probability $1/n$. By choosing an appropriately ample challenge space, no additional executions are needed anymore, because the probability $1/n$ will become negligibly small.

Analysis Considering this adaption, we now want to evaluate if the desired properties still hold. Soundness can be shown in a similar way to Theorem 1. If a prover is able to produce valid conversations after the announcement of a the prover must know the secret, because the following property holds: given that $g^{r_0} = ah^{c_0}$ and $g^{r_1} = ah^{c_1}$ hold simultaneously, also $h = g^{(r_0-r_1)/(c_0-c_1)}$ must hold, therefore the prover must know the secret x otherwise he would have not been able to calculate r_0 and r_1 after the announcement of a.
Honest-verifier-zero-knowledge can be shown by adapting the simulator accordingly, the challenge as well as the response will be sampled at random from $\mathbb{Z}_n$. A valid conversation (a, c, r) will occur with the same probability in both cases, therefore simulated and real conversations are still indistinguishable.

2.9.1 Σ-Protocols

Sigma protocols concretize the scheme depicted in Figure 2.2. The simulator will be adapted such that it also takes the challenge c as additional input. A scheme/protocol is to be said a Σ-Protocol if the following properties hold:

Completeness: if the provided statement is true and both parties follow the protocol accordingly, a honest verifier will always be convinced of this fact by a honest prover.

Special Soundness: given two accepting conversations (a, c_0, r_0) and (a, c_1, r_1) with $c_0 \neq c_1$, there exists a p.p.t. algorithm $\mathcal{E}$ which allays computes a witness ω.[3].

Special honest-zero-knowledge : There exists a p.p.t. simulator algorithm, which on input of v and challenge $c \in C$ [4], produces accepting conversations with the same probability distribution as real conversations between prover and verifier.

In general Σ-protocols consist of three steps/messages: *announcement*, *challenge* and *response*. A protocol of this form obviously requires interaction between prover and verifier. As stated in previous sections of this thesis, a transaction protocol applicable to the desired setting should be non-interactive. To achieve this the scheme described in Figure 2.2 will be turned non-interactive by applying the following changes: instead of sampling $c \xleftarrow{\$} \mathbb{Z}_n$ at random, the prover will calculate the challenge by applying a cryptographic hash-function H to all public parameters. The idea behind this is that H produces pseudorandom outputs and is therefore a suitable replacement for selecting c at random. Following this the challenge will now be calculated as follows: given a cryptographic hash-function $H : \{0,1\}^* \rightarrow \mathbb{Z}_n$ set challenge $c \leftarrow H(g\|a\|h)$. With this adaption a prover can now calculate all necessary information on its own. The properties of the used hash-function ensure that, even though c is now a deterministic value, the verifier will still be convinced of the provided statement. This method is know as **Fiat-Shamir-heuristic** [FS87], which can be applied to every interactive Σ-protocol, to turn it into a non-interactive variant.

[3] A witness ω denotes the private input of the verifier to calculate a valid response r

[4] v denotes the common input to prover and verifier.

In a formal way, a non-interactive Σ-protocol, for Figure 2.2, is described as:

1. $u \xleftarrow{\$} \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. $c \leftarrow H(g\|a\|h)$
4. $r \leftarrow_n u + cx$
5. everyone is then able to verify that $g^r \overset{?}{=} ah^c$ holds, by calculating the challenge c from announcement a.

2.9.2 Range-Proofs

[GMW91] showed that all languages in NP have zero-knowledge proofs, so we know that it is possible to prove our desired statements in zero-knowledge. When considering obfuscated values it is necessary to ensure that operations on these values will yield valid results. Given a ring $\mathbb{Z}_p$ with $p := 11$, values greater than p will behave like negative numbers. To make this more clear, consider a really basic and practical example: $(1 + 1) \pmod{11} = (5 + 19) \pmod{11} = (5 - 3) \pmod{11}$. By taking adequate large numbers, the structure of Z_p hence allows one to perform *subtraction.*
For our case we need to ensure that such *large* values are not allowed in the protocol, because the presence of *negative* numbers would significantly endanger the integrity of the system e.g. a user would be able to *steal* coins from others. Range proofs are a useful tool which allows one to convince a verifier that a hidden value, like in a Pedersen commitment, lies in a provided range and will hence cause no *overflow*[5].
Considering our setting, the spender would to show that a committed value lies in a given range. This could be done by proving that an arithmetic circuit which implements this requirement is satisfied. Looking at the method of [Boo+16], this would mean that the circuit needs to implement the commitment scheme itself, which would lead to a significant increase in complexity. This fact has been stated by [Bün+18]. They invented a system allowing one to prove that a committed value lies in a given range, while keeping the proof size as short as possible. In the following the concept of [Bün+18] as well as the suitability for our requirements will be described.

[5]In this context an overflow means that a value is larger as the defined set and will hence be reduced mod p.

2.9.3 Bulletproofs

Bulletproofs [Bün+18] provide the possibility to prove that a value hidden in a Pedersen commitment lies in a provided range, while keeping the proof-size logarithmically short in the input-size. They used the inner-product argument discussed in [Boo+16] as a building block to define their system.

An inner-product argument allows one to prove the following: given two vectors $\mathbf{g}$, $\mathbf{h} \in \mathbb{G}^n$ defined as $(g_1 ... g_n)$, with $g_i \in \mathbb{G}$ denoting a generator of $\mathbb{G}$, and a scalar value $c \in \mathbb{Z}_p$. A prover $\mathcal{P}$ can show the knowledge of two vectors $\mathbf{a}$, $\mathbf{b}$ $\in \mathbb{Z}_p^n$, with $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=0}^{n} a_i b_i$ denoting the dot product, such that the following property holds:

$$P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle$$

So the inner-product argument provides a proof-system for the following relation:

$$\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, P \in \mathbb{G}, c \in \mathbb{Z}_p : \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle\} \tag{2.2}$$

The knowledge of vectors $\mathbf{a}$, $\mathbf{b}$ could simply be shown, by sending them to the verifier, which is essentially the opening O for P. The relation of Equation 2.2 can further be improved by adding c as a part of the commitment P:

$$\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} u^{\langle \mathbf{a}, \mathbf{b} \rangle}\} \tag{2.3}$$

Inner Product Argument. As stated previously, such a statement could be proven be sending both vectors. The verifier then recomputes c and checks if equality holds. In this scenario the proof-size and verification time grows *linearly*, depending on the input vectors. [Bün+18] showed how to reduce to complexity to $O(\log_2 n)$: Consider two vectors $\mathbf{a}, \mathbf{b} \in Z_p^n$ for which $c = \langle \mathbf{a}, \mathbf{b} \rangle$ holds :

$$a = \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} \quad b = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}$$

The prover splits these vectors into two parts of equal size $n' = n/2$, yielding:

$$\mathbf{a}_{lo} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n'-1} \end{pmatrix} \quad \mathbf{a}_{hi} = \begin{pmatrix} a_{n'} \\ \vdots \\ a_n \end{pmatrix} \quad \mathbf{b}_{lo} = \begin{pmatrix} b_0 \\ \vdots \\ b_{n'-1} \end{pmatrix} \quad \mathbf{b}_{hi} = \begin{pmatrix} b_{n'} \\ \vdots \\ b_n \end{pmatrix}$$

Further in the explanation we will use the following notation: $\mathbf{a}_{lo} = \mathbf{a}_{[:n']}, \mathbf{a}_{hi} = \mathbf{a}_{[n':]}$ and $\mathbf{b}_{lo} = \mathbf{b}_{[:n']}, \mathbf{b}_{hi} = \mathbf{b}_{[n':]}$. To now prove knowledge of vectors $\mathbf{a}, \mathbf{b}$, prover $\mathcal{P}$ and verifier $\mathcal{V}$ engage the following protocol: $\mathcal{P}$ receives a challenge value $x \xleftarrow{\$} \mathbb{Z}_P$ from $\mathcal{V}$ and calculates $\mathbf{a}' = \mathbf{a}_{[:n']}x + \mathbf{a}_{[n':]}x^{-1}$ and $\mathbf{b}' = \mathbf{b}_{[:n']}x^{-1} + \mathbf{b}_{[n':]}x$:

$$\mathbf{a}' = \begin{pmatrix} a_0 x + a_{n'-1}x^{-1} \\ \vdots \\ a_{n'}x + a_n x^{-1} \end{pmatrix} \quad \mathbf{b}' = \begin{pmatrix} b_0 x^{-1} + b_{n'}x \\ \vdots \\ b_{n'-1} + b_n x \end{pmatrix}$$

With $\mathbf{a}'$ and $\mathbf{b}'$, c' is calculated as: $c' = \langle \mathbf{a}', \mathbf{b}' \rangle$. The following structure can be observed for c':

$$\begin{aligned}
c' =& \langle \mathbf{a}', \mathbf{b}' \rangle \\
=& \langle \mathbf{a}_{[:n']}x + \mathbf{a}_{[n':]}x^{-1}, \mathbf{b}_{[:n']}x^{-1} + \mathbf{b}_{[n':]}x \rangle \\
=& \Sigma_{i=0}^{n'} (\mathbf{a}_{[n':]_i}x + \mathbf{a}_{[:n']_i}x^{-1}) \cdot (\mathbf{b}_{[:n']_i}x^{-1} + \mathbf{b}_{[n':]_i}x) \\
=& \Sigma_{i=0}^{n'} (\mathbf{a}_{[:n']_i}x \cdot \mathbf{b}_{[n':]_i}x^{-1}) + (\mathbf{a}_{[n':]_i}x^{-1} \cdot \mathbf{b}_{[:n']_i}x^{-1}) + \\
& (\mathbf{a}_{[:n']_i}x \cdot \mathbf{b}_{[n':]}x) + (\mathbf{a}_{[n':]_i}x^{-1} \cdot \mathbf{b}_{[n':]_i}x) \\
=& \Sigma_{i=0}^{n'} (\mathbf{a}_{[:n']_i} \cdot \mathbf{b}_{[n':]_i}) + (\mathbf{a}_{[n':]_i} \cdot \mathbf{b}_{[n':]_i}) + \\
& \Sigma_{i=0}^{n'} x^{-2} (\mathbf{a}_{[n':]_i} \cdot \mathbf{b}_{[n':]_i}) + \\
& \Sigma_{i=0}^{n'} x^{2} (\mathbf{a}_{[:n']} \cdot \mathbf{b}_{[n':]_i}) \\
=& \langle \mathbf{a}, \mathbf{b} \rangle + x^2 \langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle + x^{-2} \langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle \\
=& c + x^2 L + x^{-2} R
\end{aligned} \tag{2.4}$$

Looking at the last line of Equation 2.4, it is easy to see that the result depends on c, which is the claimed statement, as well as two values L and R, which depend on the actual representation of $\mathbf{a}$ and $\mathbf{b}$ and the challenge value x. Regarding subsection 2.9.1, the tuple (L, R), will be treated as the announcement and the tuple $(\mathbf{a}', \mathbf{b}')$ as the respective response.
We observe that this scheme can be repeated recursively by setting $c = c'$ and $\mathbf{a} = \mathbf{a}', \mathbf{b} = \mathbf{b}'$, until the vectors $\mathbf{a}', \mathbf{b}'$ both reach length one. Every step reduces the length of the current vectors by $1/2$, hence the complexity is defined by $O(\log_2 n)$. To get a better understanding of the *shrinking* of the proven vectors, an illustration can be found in figure Figure 2.3.

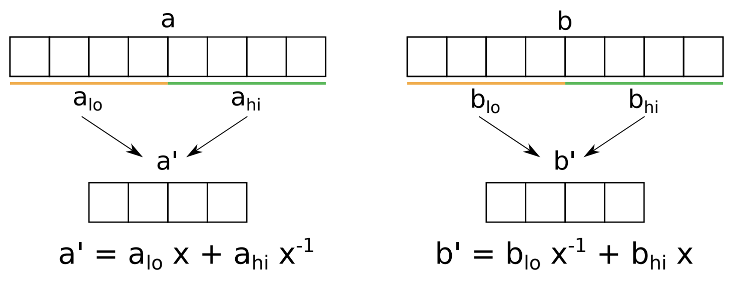


Figure 2.3: Vector shrinking for inner product arguments.

In the last step the verifier will be able to check that $c' = a' \cdot b'$ holds. Taking values c', L, R of previous steps, the verification procedure can be *rewinded* to then check c against c' and L, R of this step.
Taking this approach into account a proof-scheme for the relation defined in Equation 2.3 is defined as follows:

$$\begin{array}{lcr}
\textsf{Prover } \mathcal{P} & & \textsf{Verifier } \mathcal{V} \\
\hline
n' = n/2 & & \\
L \leftarrow \mathbf{g}_{[n':]}^{\mathbf{a}_{[:n']}} \cdot \mathbf{h}_{[:n']}^{\mathbf{b}_{[n':]}} \cdot u^{\langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle} & & \\
R \leftarrow \mathbf{g}_{[:n']}^{\mathbf{a}_{[n':]}} \cdot \mathbf{h}_{[n':]}^{\mathbf{b}_{[:n']}} \cdot u^{\langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle} & \xrightarrow{L,R} & \\
 & & x \overset{\$}{\leftarrow} \mathbb{Z}_p \\
 & \xleftarrow{x} & \\
\mathbf{a}' \leftarrow \mathbf{a}_{[:n']} x + \mathbf{a}_{[n':]} x^{-1} & & \\
\mathbf{b}' \leftarrow \mathbf{b}_{[:n']} x^{-1} + \mathbf{b}_{[n':]} x & \xrightarrow{\mathbf{a}', \mathbf{b}'} & \\
 & & P' = \mathbf{g}_{[:n']}^{x^{-1}\mathbf{a}'} \mathbf{g}_{[n':]}^{x\mathbf{b}'} \cdot \mathbf{h}_{[:n']}^{\mathbf{b}'} \mathbf{h}_{[n':]}^{x^{-1}\mathbf{b}'} \cdot u^{\langle \mathbf{a}', \mathbf{b}' \rangle} \\
 & & P' \overset{?}{=} L^{x^2} \cdot P \cdot R^{x^{-2}}
\end{array}$$

Figure 2.4: Inner product argument scheme.

The two parties reengage the scheme, as described before, $\log_2 n$ times. In the end, the verifier will be convinced that the prover indeed knows two vectors $\mathbf{a}, \mathbf{b}$ for which $\langle \mathbf{a}, \mathbf{b} \rangle$ holds.
Bulletproofs [Bün+18] build upon this approach to define a range proof-system: given a value $v \in \mathbb{Z}_p$ and let $V \in \mathbb{G}$ be the corresponding Pedersen commitment to v using blinding factor $\gamma \xleftarrow{\$} \mathbb{Z}_p$. A proof system can be defined to convince a verifier that $v \in [0, 2^n - 1]$ holds. Their developed scheme convinces a verifier of the following relation:

$$\{(g, h \in \mathbb{G}, V, n : v, \gamma \in \mathbb{Z}_p) : V = g^v h^\gamma \wedge v \in [0, 2^n - 1]\}$$

Bullet proofs Considering the conditions of an inner-product argument, the bulletproof range-proof system is defined over the following setup: represent the value v as bit vector $\mathbf{a}_L = (a_0, ..., a_{n-1}) \in \{0,1\}^n$ s.t $\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$ is satisfied; with $\mathbf{2}^n = \{2^0, 2^1, ..., 2^{n-1}\}$ being a vector of powers of two. Further trough the explanation the notation $\mathbf{y}^n = \{y^0, ..., y^{n-1}\}$ will be used to describe a vector of n powers of the base $\mathbf{y}$. The general idea of [Bün+18] is to show that the vector $\mathbf{a}_L$ is indeed a bit-representation of v as well as that every entry in $\mathbf{a}_L$ is either 0 or 1; otherwise arbitrary values could be used to compose $\mathbf{a}_L$ s.t $\langle \mathbf{a}_L, \mathbf{2}^n \rangle$ holds. So additionally the proof-system must ensure that the following constraints hold:

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v \wedge \mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^n \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}^n \tag{2.5}$$

So only valid vectors $\mathbf{a}_L \in \{0,1\}^n$ will result in a successful verification. In the end the verifier will be convinced that the prover knows v, which also has a bit representation in $\{0,1\}^n$ and therefore v **must** lie in the range $[0, .., 2^{n-1}]$. Bulletproofs furthermore allow one to proof that Equation 2.5 holds, by providing a single inner-product argument. This can be done due to the following properties:
To show that a provided vector λ only consist of zeros, namely $\lambda = \mathbf{0}^n$ is satisfied, the verifier sends a challenge value $x \xleftarrow{\$} \mathbb{Z}_p$ to the prover. The prover then shows that $\lambda \circ \mathbf{x}^n = \mathbf{0}^n$ holds[6]. This property can be utilized to rewrite the required constraints as an inner-product relation. Given a challenge value

[6] $\circ$ denotes the element-wise product of two vectors which is defined as $\mathbf{x} \circ \mathbf{y} \in \mathbb{Z}_p^n = (x_0 y_0, .., x_n y_n)$

$x \xleftarrow{\$} \mathbb{Z}_p$, to following inner-product constraints are equivalent to Equation 2.5:

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v \wedge \langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{x}^n \rangle = 0 \wedge \langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R, \mathbf{x}^n \rangle = 0 \tag{2.6}$$

We observe that $\langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{x}^n \rangle = 0$ and $\langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R, \mathbf{x}^n \rangle = 0$ only hold if $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^n$ is satisfied. This composition can further be merged into one equation by adding an additional challenge value $z \xleftarrow{\$} \mathbb{Z}_p$, yielding the following:

$$z^2 \cdot \langle \mathbf{a}_L, \mathbf{2}^n \rangle + z \cdot \langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{x}^n \rangle + \langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R, \mathbf{x}^n \rangle = z^2 v$$

It is easy to see that this equality will only be satisfied if Equation 2.6 is also satisfied. By doing the math this equation can be rewritten into a single inner-product argument:

$$\langle \mathbf{a}_L - z\mathbf{1}^n, \mathbf{x}^n \circ (\mathbf{a}_R + z\mathbf{1}^n) + z^2\mathbf{2}^n \rangle = z^2 v + \delta(x, z) \tag{2.7}$$

with $\delta(x, z) = (z - z^2)\langle \mathbf{1}^n, \mathbf{x}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{2}^n \rangle$. $\delta(x, z)$ depends only on the challenge values as well as constant vectors, hence the verification can be calculated easily by the verifier. The vectors in Equation 2.7, contain information about $\mathbf{a}_L$. To ensure that no information about v is disclosed, the prover needs to choose two additional vectors $\mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^n$ which will then be used to blind the vectors $\mathbf{a}_L - z\mathbf{1}^n$ and $\mathbf{x}^n \circ (\mathbf{a}_R + z\mathbf{1}^n) + z^2\mathbf{2}^n$ of.
Taking all of this into account the actual protocol defines three vector polynomials $l(X), r(x), t(X) \in \mathbb{Z}_p[X]$, of the following structure:

$$\begin{aligned} l(X) &= (\mathbf{a}_L - z\mathbf{1}^n) + \mathbf{s}_L \cdot X \\ r(X) &= \mathbf{x}^n \circ (\mathbf{a}_R + z\mathbf{1}^n + \mathbf{s}_R \cdot X) + z^2\mathbf{2}^n \\ t(X) &= \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2 \end{aligned} \tag{2.8}$$

The prover will hence be able to engage the protocol with $l(x)$ and $r(x)$ for $x \in \mathbb{Z}_p^*$, without revealing any information about v. Additionally the following structure can be observed for $t(X)$:

$$\begin{aligned} t(X) &= \langle l(X), r(X) \rangle \\ &= \langle (\mathbf{a}_L - z \cdot \mathbf{1}^n) + \mathbf{s}_L \cdot X, \mathbf{y}^n \circ (\mathbf{a}_R + z\mathbf{1}^n + \mathbf{s}_R \cdot X) + z^2\mathbf{2}^n \rangle \\ &= vz^2 + \underline{(z - z^2) \cdot \langle \mathbf{1}^n, \mathbf{y}^n \rangle - z^3 \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle} + t_1 \cdot X + t_2 \cdot X^2 \\ &= vz^2 + \delta(x, z) + t_1 \cdot X + t_2 \cdot X^2 \end{aligned}$$

If the prover can show that the constant term in $t(X)$ satisfies the desired property in Equation 2.7, the verifier will be convinced that the statement is true. To achieve this, the prover commits to the coefficients t_1 and t_2; $t(x)$ will then be evaluated at a randomly chosen challenge point.
Bulletproofs allow a verifier to prove that a committed value v can be represented as bit-vector $\mathbf{a}_L \in \{0,1\}^n$, which implies that the value v **must** lie in the range $[0, ..., 2^n - 1]$. This primitive suite the required constraints for a confidential transaction protocol very well. The protocol further fulfills all necessary properties described in subsection 2.9.1 and can thus be used as a tool to preserve the integrity of the ledger.
For more details on Bulletproofs and their possible other fields of application we refer to [Bün+18].

2.10 Digital Signature Scheme

Digital signature schemes allow one to verify the authenticity of signed information. The concept was was first discussed in [DH76].
Digital signatures can be considered as the digital counter part to hand written ones. Such schemes are usually, like public-key encryption schemes, defined in a "asymmetric" way. Different keys will be used for signing and verification; the private part will be used for signing and the public counterpart is then used for verification. A digital signature provides the possibility to calculate a signing-key depended value of a provided message, which can then be verified by anyone at a later point in time.
Furthermore a digital signature should not leak anything about the used signing key. It should also not be possible for an adversary to calculate valid signatures for arbitrary messages of his choice, by deducing information from valid signatures produced by a dedicated party. Every digital signature-scheme must fulfill the above mentioned requirement. [GMR95] further defined what is considered to be a *break* of a signature scheme:

- **Total break**: obtaining the private signing-key. A total break would allow the attacker to sign arbitrary messages on behalf of party A.
- **Universal Forgery**: obtaining a signing algorithm equivalent to A.
- **Selective Forgery**: forge a signature for a *a priori* selected message.

- **Existential Forgery**: forge a signature for a message m the attacker has not obtained a message/signature pair beforehand.

A digital signature scheme provides the following cryptographic properties:

1. **Authenticity:** a signature should ensure authenticity, which is the assurance that the source of the provided signature is the one its claims.
2. **Integrity:** it should not be possible to change the message a signature has been issued for, while the signature stays valid.
3. **Non-Repudiation:** it should be impossible to deny a previous action.

A digital signature can be used to ensure that transactions can only be issued by the actual owner of the assets. The signature will be provided as part of the transaction, and its validity is verified during the verification process.

2.10.1 Schnorr Signature

Schnorr's-identification protocol, as described in Figure 2.2 can be transformed into a digital signature scheme by applying the Fiat-Shamir heuristic: given message M, set the challenge to $c \leftarrow H(a \| M)$ as described in [Sch19].

Definition 2.10.1 (Schnorr Signature)**.** The Schnorr signature scheme consists of three algorithms: *KeyGen*, *Sign*, *Verify*

$\mathsf{KeyGen}(1^k)$: let $\mathbb{G}$ be a cyclic group of prime order p with generator $g \in \mathbb{G}$. Set private-key $x \xleftarrow{\$} \mathbb{Z}_p$ and public-key $h = g^x$.

$\mathsf{Sign}(\mathsf{x}, \mathsf{M})$: On input of private-key x and message M, choose $u \xleftarrow{\$} \mathbb{Z}_p$ and set $a \leftarrow g^u$. Set challenge value c to $c \leftarrow H(a \| M)$ where H denotes a cryptographic hash function. Set response $r \leftarrow_p u + xc$. Return the tuple (c, r) as signature of M.

$\mathsf{Verify}((\mathsf{c}, \mathsf{r}), \mathsf{h}, \mathsf{M})$: On input of signature tuple (c, r), public-key h and Message M, accept (c, r) as signature of M iff. $H(g^r h^{-c} \| M) = c$ holds, otherwise return $\perp$.

Remarks In section 2.9 we described that Figure 2.2 is special-honest-zero knowledge. So an honest verifier would be able to generate accepting conversations by executing the described simulator algorithm. But in the general case

of a cheating verifier, the success-probability will be $1/n$.
If it could be shown that Schnorr's-identification protocol is indeed **zero-knowledge**, this would mean that signatures could be forged. The general zero-knowledge property requires the possibility of simulating accepting conversations for an arbitrary verifier. For the challenge space $\mathbb{Z}_n$, there exist no such p.p.t. algorithm which fulfills this requirement, because the probability of guessing the correct challenge c is $1/n$. Hence on average n tries are needed to obtain an accepting transcript. If a simulator, producing accepting conversations with the same probability distribution as real conversations for an arbitrary verifier, exists a cheating prover would be able to compute valid signatures for a message M without knowing the signing key. So the honest-verifier-zero-knowledge property provides a significant security guaranty. For further details on this property we refer to [Sch19].
The signature scheme is obtained by applying the Fiat-Shamir heuristic as previously described. The message M will be added as additional input to the hash-function, due to the properties of a cryptographic hash function the distribution of c will not change, therefore the adapted scheme can be used as a signature scheme.

3 System Setting

In this chapter, we will define the surrounding conditions a distributed ledger-based payment system should be composed of, such that the upcoming protocol description will be directly applicable.

3.1 Cryptographic Setting

In this section, we will define the dedicated cryptographic setting needed to enable the actual definition of a confidential transaction protocol. In the upcoming descriptions we will use the following notation: the variable a denotes the transferred amount and b denotes the balance of the specified user. If not clear from the text a subscript may be added to the variable b to define the owner.

3.1.1 Homomorphic Cryptosystem

As explained before, simply hiding the amounts is not sufficient enough, because the protocol requires operations on the amounts. When considering a transaction from one account to another, the corresponding balances need to be updated after the transaction has been executed. The spender's balance will be decremented and the receiver's balance will be incremented by the provided amount. In the classical design, where both of these values are available in plain form, the update is trivial.

Considering a system, where all of these values are no longer available in plain-text, the trivial operations need to be adapted accordingly.
A primitive which supports this requirements could be a commitment scheme as

described in Definition 2.7.2. Amounts and balances would then be represented as Pedersen-commitments. Due to the described homomorphic-properties a commitment to the updated balance can be computed without knowing the actual plain values. However this primitive has a significant disadvantage: only the spender knows the transferred amount, so without the opening the receiver will not be able to recover its updated numeric balance. Taking this into account Pedersen-commitments will not be directly usable to hide the sensitive numeric values. The opening information needs to be made available to receiver, which could be done using an additional secure channel. However this may require both parties to be online at the same time, and obviously also breaks with the goal of non-interaction between spender and receiver. So this approach will not cover all desired requirements.

To omit the necessity of interaction, we will instead use ElGamal as described in Definition 2.6.2, as our cryptosystem. This scheme works similar to the described commitment scheme, but does not require any additional communication.
The properties described in subsection 2.6.3 allows the computation of a valid encryption of the updated balances. Only the party holding the corresponding secret key, will then be able to decrypt their updated balances again. So in general the advantage of this approach is that no interaction between the involved parties is needed in order to recover the new balance values, which fits our desired design goals.
Amounts of assets as well as balances will be present as ElGamal cipher-texts. The plain values are positive integers, hence an encoding for integers to elliptic-curve points must be defined. To encrypt numeric values with EC-ElGamal, the values will be represented as multiples of the base-point P. The message input will then be defined as: $M \leftarrow aP$. Considering two encryptions under the same public key, using the same message encoding, the following property holds: referring to equation Equation 2.1 with $M_1 \leftarrow a_1P$, $M_2 \leftarrow a_2P$ representing the messages and C_1, C_2 representing the encryptions of M_1 and M_2, it is easy to see that the homomorphic addition of C_1 and C_2 yields $Encrypt((a_1 + a_2)P)$. Hence a valid encryption for the encoding of $a_1 + a_2$ can be obtained by only using the cipher-texts.

3.1.2 Convincing

Because the amounts and balances are encrypted and thus no information about the plain values can be deduced, the spender needs to provide a proof showing that $a > 0$ holds and that the encrypted amounts are correctly computed as $enc\_amount_i \leftarrow Enc(a, pk_i)$ and $enc\_amount_j \leftarrow Enc(a, pk_j)$. To ensure that only valid amount can be transferred, additionally the inequality $a \leq b$ must hold.
To prevent tampering with the system, all of these properties need to be proven in a way s.t. no information about the plain values will be leaked.

Consider the following transaction proposal: User A currently holds 15 units and wants to transfer -10 units to user B. We omit a proof for $a > 0$ and just rely on the fact that user A knows a value a s.t. $enc\_amount_i \leftarrow Enc(a, pk_i)$ and $enc\_amount_j \leftarrow Enc(a, pk_j)$ holds, and that also his balance is correctly encrypted. Considering this case, the constraint $b \geq a$ will also hold. When this transaction is confirmed and a balance update is executed, user A actually steals 10 units from user B.
The inequalities $a > 0$ and $a \leq b$ thus ensure the integrity of the ledger. We observe that the second inequality can be rewritten to $b - a \geq 0$, so both inequalities can be proven by using bulletproofs. A more detailed explanation on how bulletproofs can be utilized in this context will be presented in chapter 4.

3.2 System Parts

Ledger The system's state is stored in a blockchain, as described in section 2.5. Considering consistency reasons, it will be necessary to fix the elliptic curve, namely its parameters, which will be used throughout the protocol so that every depending primitive operates on the same curve.
The payment system keeps track of all balances of every user while keeping them secret. Due to the properties of ElGamal, an observer won't be able to deduce any information about the numeric balances and amounts.

Block. One block is essentially one entry of the list. A block holds the necessary information to define the actual state. In our case, a block contains a list of valid transactions. Transactions move assets from the spender to the receiver, by defining from to relations. The balance of every user will be saved in the world state, similar to [But14], which essentially is a database relating to the meta-information stored in the blockchain. The nodes have access to this world state and will execute the balance updates. The stored transactions can then be used to validate the state history. The consistency and integrity of the system will be preserved by ensuring that only valid transactions will be added to the ledger.

Users Users are the actual owners of the assets. The system provides the possibility to transfer an arbitrary amount a, which satisfies $a \leq b$ to any other user of the system. Users will propose these asset-transfers as transactions to the nodes. Nodes will be able to collect multiple transaction requests to define a new block. For simplicity, we will consider one transaction per block so that a single valid transaction will cause a state change. Keep in mind that the state handling will not have an impact on the transaction protocol.
Users hold a key-pair as described in section 2.6. To carry out a transaction, the spending user needs to know the receiver's public-key. Hence the system needs to provide a possibility to query these keys; otherwise, transactions will not be possible. Besides, users will also hold another key-pair as describe in section 2.10, which will be used to sign and verify transactions. The according signatures will be verified by the nodes during the verification process.
Users are identified by a unique value called **address**, which will be used to identify the spender and receiver of a proposed transaction. Keep in mind that the users balances are only *protected* from malicious parties if the decryption and signing key are kept secret.

Nodes Nodes are designed as dedicated parts of the ledger. They are responsible for verifying transactions. Consistency and integrity of the system needs to be ensured, thus only well formed and valid transactions will cause a state change. Transactions contain values, which have been constructed using the according primitive. Malformed or invalid information will not lead to a successful verification with overwhelming probability.

Transactions contain multiple cryptographic proofs, and are only considered valid iff. the following properties hold:

- The senders and receivers addresses are valid. So they identify users of the system.
- The spenders signature is valid.
- All provided proofs are valid.

We observe that transactions can necessarily be seen as a combination of different proofs. Even though the amounts are encrypted, the verifier does not need to decrypt them in the verification process. Proof verifications will thus dominate the verification of transactions.
When the system should be capable of handling many transactions per second, proof-systems that support a fast verification while being reliable may be considered.

Transactions A transaction is essentially a container holding all information necessary to describe an asset transfer from one party to another. All contained proofs should be non-interactive to allow verification at a later point in time. A detailed description of the actual transaction structure will be presented in subsection 4.1.8

3.3 Additional Properties

Keep in mind that a payment system may differ from the described system setting. Some systems should be capable of managing multiple assets. To achieve this, the actual ledger will consist of multiple blockchains. Every single chain will keep track of one asset, resulting in every user holding different balances for different assets.
It could also be of interest to define a permission scheme, so only permissioned users will be allowed to issue transactions or inspect the blockchain itself.
Even though these options exist and may be desirable to achieve different goals, the verification process itself will not be different.

4 Confidential Transaction Protocol

In this section we will describe a confidential transaction protocol which will fulfill all required constraints. Every part of the protocol will be described in detail. Protocol parts including cryptographic primitives will be analyzed in terms of their security.

The confidential transaction protocol will describe three actors: *Spender*, *Receiver*, *Verifier/Node*. In the following we will describe every step for each actor in detail, which will then compose the whole protocol. The protocol will not require any interaction between spender and receiver as well as between spender and verifier. Hence the described transaction protocol is non interactive and can therefore be split into separate parts.

4.1 Spender

The protocol is initiated by the spender, by sending a composed transaction to the nodes.
The meta-information of the transactions is decided by the spender and is defined as:

- **Amount:** the desired numeric amount to transfer to the receiver.
- **Receiver:** the recipient of the transferred units.

This information will be used as starting point of the transaction composition.

To construct a valid transaction, the spender performs the following steps:

1. Represent the amount in a form applicable to the cryptosystem defined in subsection 2.6.1
2. Encrypt the amount w.r.t the spenders and receivers public key using ElGamal yielding values x_s and x_r.
3. Calculate a commitment C to the amount a, depending on encryption x_s and x_r.
4. Construct a proof describing that the amount encrypted in x_s and x_r is the same as in commitment C.
5. Construct a proof describing that the amount is strictly positive using the algorithm described in subsection 2.9.2.
6. Construct a proof describing that the amount is less than or equal to the spender's balance using the algorithm described in subsection 2.9.2.
7. Re-encrypt the currently stored balance in the world state.
8. Sign the transaction with the corresponding private key to prove ownership of the transferred units, using the primitive defined in subsection 2.10.1.
9. Propose transaction for verification.

In the following, each step of this workflow will be explained in detail.

4.1.1 Amount Representation

As previously described, ElGamal is only capable of operating with elements applicable to the system, so the numeric amount a must be represented accordingly. Further, the homomorphic property of ElGamal should allow a confidential balance update of the involved users. For this to work in the desired way, the amount a, as well as the balance b_i of every user, will be represented as powers of the decided generator $g \in \mathbb{G}$, $M = g^a$.

4.1.2 Encrypt Transaction Amount

The previous step yields a representation for the amount, which will then be used as message input during encryption.
Encrypt the representation of amount a two times w.r.t the spender's and receiver's public key denoted as y_s, y_r yielding: $x_s = (C_1, C_2) = (g^r, g^a y_s^r)$ and

$x_r = (C_1, C_2) = (g^{r'}, g^a y_r^{r'})$ with $r, r' \xleftarrow{\$} Z_p$. The spender then adds x_s and x_r to the transaction.

4.1.3 Commit to Amount

Calculate a Pedersen-commitment to amount a as $C = g^a h^\gamma$ with $\gamma \xleftarrow{\$} \mathbb{Z}_p$ and $h \in \mathbb{G}$ denoting a generator of the group s.t. no-one knows $\log_g(h)$. During the calculation of C, the same generator $g \in \mathbb{G}$ must be used as in subsection 4.1.2. The spender then adds the commitment C to the transaction.

4.1.4 Proof of Equality

To prevent a user from tampering with the system state, it must be ensured that only correctly computed values will lead to a successful verification. To prevent the encryption of arbitrary values, which could lead to a unrecoverable system state, the protocol must provide a possibility to check the correctness of these values.
Hence it must be ensured that the same amount has been encrypted in both cipher-texts, x_r and x_s. Otherwise the integrity of the ledger can not be guaranteed. In the following a Σ-protocol, which proves this relation will be described. s

Scheme. A proof showing the equality of a Pedersen-commitment to an ElGamal-cipher-text, in terms of the same hidden value x, is defined in the following setting: given a cyclic group $\mathbb{G}$ of order p, a private/public key-pair, as described in Definition 2.6.1, is defined as $s_k = \alpha$ and $y = g^{\alpha}$[1].
According to the description of Definition 2.6.1 an ElGamal-cipher-text is defined as: $(C_1, C_2) = (r, s) = (g^r, g^x y^r)$. A Pedersen-commitment in this context is then defined as: $C_3 = g^x h^\gamma$ with $\gamma \xleftarrow{\$} \mathbb{Z}_p$.
Given this setup the following depiction illustrates the corresponding Σ-protocol, which proves that an ElGamal cipher-text and a Pedersen-commitment hide the same secret value:

[1] s_k denotes the secret key and y denotes the corresponding public key.

$$\begin{array}{ccc}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
\hline
\mathcal{P}(x, r, \gamma, g, h, y, \mathbb{G}) & & \mathcal{V}(C_1, C_2, C_3, g, h, y, \mathbb{G}) \\
v_1, v_2, v_3 \xleftarrow{\$} \mathbb{Z}_p & & \\
q_1 \leftarrow g^{v_1} y^{v_2} & & \\
q_2 \leftarrow g^{v_1} h^{v_3} & & \\
q_3 \leftarrow g^{v_2} & & \\
& \xrightarrow{q_1, q_2, q_3} & \\
& & c \xleftarrow{\$} Z_p \\
& \xleftarrow{c} & \\
z_1 = v_1 + cx & & \\
z_2 = v_2 + cr & & \\
z_3 = v_3 + c\gamma & & \\
& \xrightarrow{z_1, z_2, z_3} & \\
& & g^{z_1} y^{z_2} \stackrel{?}{=} q_1 C_2^c \\
& & g^{z_1} h^{z_3} \stackrel{?}{=} q_2 C_3^c \\
& & g^{z_2} \stackrel{?}{=} q_3 C_1^c
\end{array}$$

Figure 4.1: Proof of equality of ElGamal cipher-text and Pedersen commitment.

Analysis. We now want to analyse this scheme in terms of **correctness**, **special-soundness** and **special honest-verifier-zero-knowledge**.

Correctness: To show the correctness property of the scheme provided in Figure 4.1, the equations need to be evaluated:

$$\begin{aligned}
g^{z_1} y^{z_2} &\stackrel{?}{=} q_1 C_2^c \\
g^{v_1 + cx} y^{v_2 + cr} &\stackrel{!}{=} g^{v_1} y^{v_2} g^{cx} y^{cr} \\
g^{v_1} y^{v_2} g^{cx} g^{rc} &= g^{v_1} y^{v_2} g^{cx} g^{rc}
\end{aligned} \tag{4.1}$$

$$\begin{aligned}
g^{z_1} h^{z_3} &\stackrel{?}{=} q_2 C_3^c \\
g^{v_1 + cx} h^{v_3 + c\gamma} &\stackrel{!}{=} g^{v_1} h^{v_3} g^{cx} h^{c\gamma} \\
g^{v_1} h^{v_3} g^{cx} g^{c\gamma} &= g^{v_1} h^{v_3} g^{cx} h^{c\gamma}
\end{aligned} \tag{4.2}$$

$$\begin{aligned} g^{z_2}h^{z_3} &\overset{?}{=} q_3C_1^c \\ g^{v_2+cr} &\overset{!}{=} g^{v_2}g^{cr} \\ g^{v_2}g^{cr} &= g^{v_2}g^{cr} \end{aligned} \tag{4.3}$$

Special Honest-Verifier-Zero-Knowledge: A scheme provides special honest-verifier-zero-knowledge, as described in subsection 2.9.1, if there exists a p.p.t. algorithm $\mathcal{S}$ which, on input of public information and challenge value c, produces simulated conversations $((q_1, q_2, q_3); c; (z_1, z_2, z_3))$, which are indistinguishable from real conversations produced by $\mathcal{R}$[2]. A simulator algorithm $\mathcal{S}$ for the scheme provided in Figure 4.1 is defined as:

Simulated Conversations:
Input: C_1, C_2, C_3, c, g, h, $\mathbb{G}$

1. $z_1, z_2, z_3 \xleftarrow{\$} \mathbb{Z}_p$
2. $q_1 = g^{z_1}y^{z_2}C_2^{-c}$
3. $q_2 = g^{z_1}h^{z_3}C_3^{-c}$
4. $q_3 = g^{z_2}C_1^{-c}$
5. return $((q_1, q_2, q_3); c; (z_1, z_2, z_3))$

Proof. Considering a communication transcript produced by $\mathcal{S}$, we observe that the response tuple (z_1, z_2, z_3) is generated uniformly at random. Given this, the calculated announcement tuple (q_1, q_2, q_3) will also be random. The constructed conversation will be accepted because the required constraints are satisfied, even though the secret values x, γ, r are unknown to the verifier. The response tuple (z_1, z_2, z_3), as well as the announcement tuple (q_1, q_2, q_3) calculated during $\mathcal{R}$ will depend on uniformly selected random values v_1, v_2, v_3 as well as $c \xleftarrow{\$} \mathbb{Z}_p$ and will therefore follow the same distribution as in $\mathcal{S}$. Hence constructed conversations of $\mathcal{S}$ and $\mathcal{R}$ will occur with the same probability distribution and are therefore indistinguishable. More formally $\mathcal{S} \cong \mathcal{R}$ holds[3]. $\square$

[2] $\mathcal{R}$, denotes the real conversation protocol.

[3] $\cong$ the output of left operator p.p.t. algorithm occurs with the same probability distribution as the output of the right operator algorithm.

Special Soundness: The special soundness property, as described in subsection 2.9.1, can be shown in the following way: *after* the announcement of tuple $a = (q_1, q_2, q_3)$ a prover is able to calculate two valid responses for distinct challenges $c \neq c'$, which yields **accepting** conversations $\varphi_1 = (a; c; (z_1, z_2, z_3))$ and $\varphi_2 = (a; c'; (z'_1, z'_2, z'_3))$.
As an analogy, scheme Figure 4.1 can be seen as a virtual machine $\mathcal{Z}$ which implements the protocol. After the announcement a has been published, a snapshot of the current state will be taken. $\mathcal{Z}$ will be resumed and another virtual machine $\mathcal{Z}'$ will be initialized with the previously taken snapshot. So both executions start with the same announcement, hence also values (v_1, v_2, v_3) will be the same, but the challenge values will be different in both executions. Accepting conversations $\mathcal{Z} \rightarrow \varphi_1$, $\mathcal{Z}' \rightarrow \varphi_2$, can only be calculated iff. the prover knows the secret values x, γ, r. A cheating prover, as previously stated, will only succeed with negligible probability.
To show this property a p.p.t. extractor algorithm $\mathcal{E}$ can be defined, which on input of the tuple $(a; c, c', (z_1, z_2, z_3), (z'_1, z'_2, z'_3))$ always calculates a witness. The according algorithm $\mathcal{E}$, for scheme Figure 4.1 is defined as follows:

Extractor $\mathcal{E}$:
Input: $a, c, c', (z_1; z_2; z_3), (z'_1; z'_2; z'_3)$
S1: Extract secret value r from z_2, z'_2:

$$\begin{aligned} z'_2 - z_2 &= (v_2 + c'r) - (v_2 + cr) \\ &= c'r - cr \\ &= r(c' - c) \\ r &= \frac{z'_2 - z_2}{c' - c} \end{aligned}$$

S2: Extract secret value γ from z_3, z'_3:

$$\begin{aligned} z'_3 - z_3 &= (v_3 + c'\gamma) - (v_3 + c\gamma) \\ &= c'\gamma - c\gamma \\ &= \gamma(c' - c) \\ \gamma &= \frac{z'_3 - z_3}{c' - c} \end{aligned}$$

S3: Extract secret value x from z_1, z_1':

$$\begin{aligned}
z_1' - z_1 &= (v_1 + c'x) - (v_1 + cx) \\
&= c'x - cx \\
&= x(c' - c) \\
x &= \frac{z_1' - z_1}{c' - c}
\end{aligned}$$

Output witness $\omega = (x, r, \gamma)$.

Given the extractor algorithm we now show that the output of $\mathcal{E}$ is indeed valid: given the returned witness $r = (z_2' - z_2)(c' - c)$ we can rewrite the claimed statement g^r as:

$$\begin{aligned}
g^r &= g^{(z_2' - z_2)/(c' - c)} \\
g^{r(c'-c)} &= g^{z_2' - z_2} \\
C_1^{c'-c} &= g^{z_2' - z_2} \\
\frac{C_1^{c'}}{C_1^c} &= \frac{g^{z_2'}}{g^{z_2}} \\
g^{z_2} &= \frac{g^{z_2'} C_1^c}{C_1^{c'}}
\end{aligned} \tag{4.4}$$

We know that both conversations are accepted, so $g^{z_2} = q_3 C_1^c$ and $g^{z_2'} = q_3 C_1^{c'}$ must hold simultaneously. By inserting each value individually in the above equation we obtain:

$$\begin{aligned}
q_3 C_1^c &= \frac{g^{z_2'} C_1^c}{C_1^{c'}} & \qquad g^{z_2} &= \frac{q_3 C_1^{c'} C_1^c}{C_1^{c'}} \\
g^{z_2'} &= q_3 C_1^{c'} & \qquad g^{z_2} &= q_3 C_1^c
\end{aligned}$$

Hence $(z_2' - z_2)(c' - c)$ is a correct witness for both conversations, so the prover must indeed know the secret value r, otherwise he would have not been able to correctly compute z_2 and z_2' in the first place.

Knowing that $r = (z'_2 - z_2)(c' - c)$ is a valid witness, we now want to show that $x = (z'_1 - z_1)/(c' - c)$ is also correct. The statement $g^x y^r$ can be rewritten to:

$$\begin{aligned}
g^x y^r &= g^{(z'_1 - z_1)/(c'-c)} y^{(z'_2 - z_2)/(c'-c)} \\
g^{x(c'-c)} y^{r(c'-c)} &= g^{(z'_1 - z_1)} y^{(z'_2 - z_2)} \\
C_2^{(c'-c)} &= g^{(z'_1 - z_1)} y^{(z'_2 - z_2)} \\
\frac{C_2^{c'}}{C_2^{c}} &= \frac{g^{z'_1} y^{z'_2}}{g^{z_1} y^{z_2}} \\
\frac{g^{z'_1} y^{z'_2} C_2^{c}}{C_2^{c'}} &= g^{z_1} y^{z_2}
\end{aligned} \tag{4.5}$$

As in Equation 4.4, we know that $g^{z_1} y^{z_2} = q_1 C_2^c$ and $g^{z'_1} y^{z'_2} = q_1 C_2^{c'}$ must hold:

$$\begin{aligned}
q_1 C_2^c &= \frac{g^{z'_1} y^{z'_2} C_2^c}{C_2^{c'}} & \qquad g^{z_1} y^{z_2} &= \frac{q_1 C_2^{c'} C_2^c}{C_2^{c'}} \\
g^{z'_1} y^{z'_2} &= q_1 C_2^{c'} & \qquad g^{z_1} y^{z_2} &= q_1 C_2^c
\end{aligned}$$

So $x = (z'_1 - z_1)/(c' - c)$ is also a correct witness for both conversations. As shown in Equation 4.4, the prover is already bound to the secret value r. z_2 and z'_2 will be used to check both claimed statements g^r and $g^x h^r$. Hence a prover will only be able to calculate two valid responses z_1, z'_1, considering the fact that he is already bound to r, by also knowing the secret value x. Finally we evaluate the correctness of the extracted witness $\gamma = (z'_3 - z_3)(c' - c)$, with already proven correct witness $x = (z'_1 - z_1)/(c' - c)$:

$$\begin{aligned}
g^x h^\gamma &= g^{(z'_1 - z_1)/(c'-c')} h^{(z'_3 - z_3)/(c'-c')} \\
g^{x(c'-c)} h^{\gamma(c'-c)} &= g^{z'_1 - z_1} h^{z'_3 - z_3} \\
C_3^{(c'-c)} &= g^{(z'_1 - z_1)} h^{(z'_3 - z_3)} \\
\frac{C_3^{c'}}{C_3^{c}} &= \frac{g^{z'_1} h^{z'_3}}{g^{z_1} h^{z_3}} \\
\frac{g^{z'_1} h^{z'_3} C_3^{c}}{C_3^{c'}} &= g^{z_1} h^{z_3}
\end{aligned} \tag{4.6}$$

As in the previous steps we know that both statements $g^{z_1}h^{z_3} = q_2C_3^c$ and $g^{z'_1}h^{z'_3} = q_2C_3^{c'}$ are satisfied:

$$\begin{aligned} q_2C_3^c &= \frac{g^{z'_1}h^{z'_3}C_3^c}{C_3^{c'}} & \qquad g^{z_1}h^{z_3} &= \frac{q_2C_3^{c'}C_3^c}{C_3^{c'}} \\ g^{z'_1}h^{z'_3} &= q_2C_3^{c'} & \qquad g^{z_1}h^{z_3} &= q_2C_3^c \end{aligned}$$

Equation 4.6 shows that also the extracted value $\gamma = (z'_3 - z_3)(c' - c)$ is a correct witness for both conversations. As in the previous step z_1, z'_1 will be used to verify both statements g^xh^r and g^xh^γ. z_1, z'_1 already satisfy g^xh^r and we know that g^xh^γ must also be true. So the prover is bound to the values x and r; otherwise the evaluation would have failed. Considering this and the fact that $\gamma = (z'_3 - z_3)(c' - c)$ is a valid witness for both conversations. We know that the prover must also know the secret value γ. The desired special soundness property thus holds for scheme Figure 4.1.
The provided scheme describes a proof-system for the following relation:

$$\{(g, h, y \in \mathbb{G}, C, X : x, r, \gamma \in \mathbb{Z}_p) : C = g^xh^\gamma \wedge X = (g^r, g^xy^r)\} \tag{4.7}$$

To convince a verifier that both cipher-texts and the commitment encapsulate the same value a, a prover needs to engage the scheme two times with the pairs (V, x_s) and (V, x_r).
So essentially the spender shows that $x_s \Leftrightarrow C \Leftrightarrow x_r$ holds[4].
In order to carry out non-interactive-transactions, so no interaction between spender and receiver should be necessary, the described scheme will be turned non-interactive by applying the Fiat-Shamir heuristic: calculate challenge c as $c \leftarrow_p H(q_1\|q_2\|q_3)$.
The spender will then add the statements $\pi_1 \leftarrow x_s \Leftrightarrow C$ and $\pi_2 \leftarrow x_r \Leftrightarrow C$ to the transaction.

4.1.5 Positive Amount Proof

The integrity of the ledger must be preserved. To prevent possible integrity violations as described in subsection 2.9.2, the spender needs to provide a

[4] $X \Leftrightarrow C$, with X as ElGamal cipher-text and C as Pedersen commitment, denotes that the relation in Equation 4.7 is satisfied.

proof showing that the amount hidden in C, x_s and x_r is “positive”, so no “overflow” will be possible. The scheme described in subsection 4.1.4 ensures that $x_s \Leftrightarrow C \Leftrightarrow x_r$ holds, which serves as a precondition for $a \geq 0$. To show this property, the following needs to be done by the spender: construct a bulletproof as described in subsection 2.9.3, with $V = C$ and $v = a$, to prove the relation:

$$\{(g, h \in \mathbb{G}, C, n : a, \gamma \in \mathbb{Z}_p) : C = g^a h^\gamma \wedge a \in [0, 2^n - 1]\}$$

By showing this relation, the verifier will be convinced that $a \in [0, 2^n - 1]$ holds and hence the hidden amount must be positive. The spender will then add a bulletproof, showing this relation, as ϕ to the transaction.
We will set n to 32 in our case, therefore the amount must be representable a 32-bit unsigned integer. The scheme described in [Bün+18] needs to be turned non-interactive by applying the Fiat-Shamir heuristic, to omit the necessity of interaction and to allow verification at a later point in time.

4.1.6 Quantity Proof

A spender may not transfer more assets than he actually possesses, therefore the spender must convince the verifier of the following inequality: $a \leq b$. As previously stated, this inequality can be rewritten to: $b - a \geq 0$, which is equal to the statement that the difference between the current balance and the transferred amount satisfies $(a - b) \in [0, 2^n - 1]$.
The proof system described in subsection 2.9.3 only convinces a verifier that a prover knows a value v, hidden in a Pedersen-commitment $V = g^v h^\gamma$, s.t v lies in a desired range. With no further restrictions to the choice of v, a prover would be able to commit to an arbitrary value v and then prove that $v \in [0, 2^n - 1]$ holds. Hence it must be ensured that the proven value is of the form $v = b - a$. To achieve this we define the following workflow: Construct a Pedersen commitment to the current balance b of the spender as $P = g^b h^\gamma$, by using the same generators $g, h \in \mathbb{G}$ as in $C = g^a h^\gamma$. Given an ElGamal cipher-text of the spenders current balance b as $x_b = (g^r, g^b y^r)$, construct a proof μ showing the relation $x_b \Leftrightarrow P$ as described in subsection 4.1.4. If μ is valid, then the verifier will be convinced that P is a Pedersen commitment to b. Next construct a bullet-proof ρ, as described in subsection 4.1.5, showing that $b \in [0, 2^{32} - 1]$ holds. Given ρ and proof ϕ from the step described in

subsection 4.1.5, the verifier will forthermore be convinced that a and b are both positive; so $a, b \geq 0$ holds. With these preconditions, the spender is then able to construct a proof for the following relation:

$$\{(g, h \in \mathbb{G}, C, P, n : (b - a), \gamma \in \mathbb{Z}_p) : V = g^{b-a} h^{\gamma - \sigma} \land v \in [0, 2^n - 1]\} \quad (4.8)$$

The prover is the only who knows the plain values a and b, as well as used blinding factors γ and σ in C and P. Hence a proof satisfying the relation shown in Equation 4.8 can be constructed.
During the verification process the homomorphic-property described in section 2.7.2 will be used to achieve the following: given commitments C and P compute $V' = P - C = g^{b-a} h^{\gamma-\sigma}$. We observe that the prover must use the blinding factor $\gamma - \sigma$ when construct V, otherwise the proof will be considered invalid. The verifier will then check the provided statement against the computed value V'.
On success the verifier will be convinced that $a \leq b$ holds, because both values are positive and due to the homomorphic property of Pedersen-commitments the verifier will be able to compute a valid commitment to $a - b$, which will then be used to check the claimed statement. The check will only success if the prover indeed used $v := (b - a)$ and $(\gamma - \sigma)$ as the blinding factor to construct the proof in the first place. A false-positive evaluation will only occur with negligible probability. The spender will then add two bulletproofs to the transaction: one which proves that the balance is positive further defined as ρ and a second one showing that the difference between the balance and amount is "positive" defined as τ.
To make the workflow more clear consider the illustration depicted in Figure 4.2.

Paramters:

C	: Commitment to amount a, with $C = g^a h^\gamma$.
b	: Current balance of the spender.
a	: Amount of units to transfer, $a \in [0, 2^{32})$.
h	: Generator $h \in \mathbb{G}$, must be the same as in C.
g	: Generator $g \in \mathbb{G}$, must be the same as in C.

Quantity Proof (Spender):

S1. Calculate commitment $P = g^b h^\sigma$, with $\sigma \xleftarrow{\$} \mathbb{Z}_p$
S2. Construct proof $\rho \leftarrow \{P = g^b h^\sigma \wedge b \in [0, 2^{32})\}$, as described in subsection 2.9.3
S3. Commit to $(b - a)$, $V = g^{b-a} h^{\sigma-\gamma}$
S4. Construct proof $\tau \leftarrow \{V = g^{b-a} h^{\sigma-\gamma} \wedge (b - a) \in [0, 2^{32})\}$, as in subsection 2.9.3
S5. return (P, ρ, τ)

Figure 4.2: Quantity proof scheme.

4.1.7 Re-Encrypt Balance

The scheme described in subsection 4.1.6 allows one to proof that the inequality $a \leq b$ holds. The presented workflow requires the calculation of value μ, which proves the equality of the users current balance and the provided balance commitment. All balances are stored in the form of ElGamal cipher-texts $(g^r, g^b y^r)$, a valid transaction will lead to a balance update by utilizing the linear homomorphic property of the cryptosystem. The involved parties will then be able to decrypt these updated values by using their secret keys.
Recap the property of homomorphic addition, as described in subsection 2.6.3, of two cipher-texts under the same public key: $C_1 = (g^r, g^a y^r)$, $C_2 = (g^{r'}, g^b y^{r'})$, $C_1 \oplus C_2$ then yields $C_3 = (g^{r+r'}, g^{a+b} y^{r+r'})$. The value C_3 can then be decrypted to recover $a+b$ [5]. We observe that the two ephemeral keys r, r' will be combined to a new value $\gamma = (r + r')$.
Given this the following problem arises: as previously described the spender

[5]To actually recover $(a + b)$, the algorithm described in algorithm 1 needs to be applied to g^{a+b}.

encrypts the amount two times, by selecting the ephemeral keys at random. By updating the balances of the involved parties, the current and new ephemeral key will be combined. The spender obviously knows these values used during the encryption and hence also knows the updated value for his encrypted balance. The receiver on the other hand does not know this temporary value and will therefore **not** be able to construct proofs which require the knowledge of this discrete value. Due to the DLP the receiver will also not be able to calculate $log_g g^{r+r'}$, otherwise the security guarantees of the system will not hold and the whole protocol will not be considered secure.
An obvious solution to this problem would be that the spender sends the used ephemeral key to the receiver, using an additional secure channel. This aproach clearly requires interaction and therefore breaks with the design goals of the protocol. If a user wants to spend his assets again he will need to perform all previously described steps. The workflow described in subsection 4.1.6 requires a cipher-text holding the spenders current balance, as well as a Pedersen commit to the balance. The current balance is stored in the world state, but a user may not know the according discrete ephemeral key, which is necessary to calculate the necessary proof.
We observe that not the actually stored encrypted balance needs to be used to verify the proofs but only **a** cipher-text which holds the balance. By providing a self computed encryption of the current balance, the proof could be calculated because the ephemeral key will be known. The spender needs to convince the verifier, in this case the node, that the provided cipher-text encrypts the same value as stored on the system. On success the verifier knows that the provided encryption holds the same value and can therefore be used during the verification process. We will now describe a scheme which proves the following relation:

$$\{(g \in \mathbb{G}, B, C, y : (b = x) : B = (g^r, g^b y^r) \wedge C = (g^{r'}, g^x y^{r'})\} \tag{4.9}$$

The prover convinces the verifier that the same value is encrypted in both cipher-texts B and C. This is equal to the statement that the difference of B and C yields an encryption of zero. By *subtracting* C from B, we obtain the following: $C \ominus B = (g^{r-r'}, g^0 y^{r-r'}) = (g^p, (g^p)^x)$. We observe that this structure is equivalent to a Schnorr-proof as described in Figure 2.2, for which the required properties such as soundness and zero-knowledge have already been shown. With $C \ominus B = (C_1, C_2) = (g^p, (g^p)^x)$, the plain scheme will be utilized in the following way:

$$
\begin{array}{ccc}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
\hline
\mathcal{P}(x, C_1, \mathbb{G}) & & \mathcal{V}(C_1, C_2, \mathbb{G}) \\
u \overset{\$}{\leftarrow} \mathbb{Z}_p & & \\
a \leftarrow {C_1}^u & \xrightarrow{a} & \\
 & & c \overset{\$}{\leftarrow} \mathbb{Z}_n \\
 & \xleftarrow{c} & \\
r \leftarrow_n u + cx & \xrightarrow{r} & {C_1}^r \overset{?}{=} a{C_2}^c
\end{array}
$$

Figure 4.3: Encryption equivalence scheme.

As before correctness can be shown by evaluating the required equation:

$$
\begin{aligned}
{C_1}^r &\overset{?}{=} a{C_2}^c \\
g^{p(u+cx)} &= g^{up}(g^{xp})^c \\
g^{up}g^{cxp} &= g^{up}g^{cxp}
\end{aligned}
$$

Special-Soundness as well as honest-verifier-zero-knowledge holds because the scheme is exactly the same as the one described in Figure 2.2 with $g = g^{(r-r')}$ and $h = g^x = g^{x(r-r')} = y^{(r-r')}$.
With this scheme the spender will be able to attach a self computed encryption of the balance, for which the discrete ephemeral key is known, and hence allows the spender to compute all other necessary proofs. The spender will add the self computed cipher-text as value y alongside with the described proof as value α to the transaction.

Remarks With this system at hand a spender is able to proof that a self computed encryption holds the same plain value as the currently stored one. A distribution ledger-based payment system which is built upon this protocol may hence allow a balance replacement. More precisely a user may request a replacement of the currently stored value in the world state with his self computed one. Essentialy a user would thus be able to re-*randomize* its stored balance in the world state.

4.1.8 Sign Transaction

As the last step of the transaction construction, the spender will sign the obtained transaction with its dedicated signing-key, which furhter serves as a prove of account ownership. For this purpouse any digitial-signature scheme which fulfills the required properties described in section 2.10 can be used. In our case we will use the scheme presented in subsection 2.10.1. The provided signature then proves the ownership of an account, because only the holder of the secret signing-key will be able to construct a valid signature.
Considering all previously described steps and the according values which need to be calculated during each of them, the following transaction structure can be obtained:

```
transaction {
    from,
    to,
    C,
    P,
    y,
    x_s,
    x_r,
    alpha,
    pi_1,
    pi_2,
    mu,
    phi,
    rho,
    tau,
    signature
}
```

- **from**: The senders address.
- **to**: The receivers address.
- **C**: A Pedersen commitment to amount a.
- **P**: A Pedersen commitment to the current balance b.
- **y**: Self computed encryption of the spenders current numeric balance.
- $\mathbf{x}_s$: Encryption of amount a under the spenders public key.

- $\mathbf{x}_r$: Encryption of amount a under the receivers public key.
- $\boldsymbol{\alpha}$: Encryption proof showing that the currently stored balance and provided cipher-text in $\mathbf{y}$ hold the same plain value.
- $\boldsymbol{\pi_1}$: Equivalence proof showing that $x_s \Leftrightarrow C$ holds.
- $\boldsymbol{\pi_2}$: Equivalence proof showing that $x_r \Leftrightarrow C$ holds.
- $\boldsymbol{\mu}$: Equivalence proof showing that $y \Leftrightarrow P$ holds.
- $\boldsymbol{\phi}$: A range-proof showing that the numeric amount satisfies $a \in [0, ..., 2^{32})$.
- $\boldsymbol{\rho}$: A range-proof showing that the numeric balance satisfies $b \in [0, ..., 2^{32})$.
- $\boldsymbol{\tau}$: A range-proof showing that the numeric balance of the sender is greater or equal to the actual amount, namely $b \geq a$ holds.
- **signature**: A digital signature, which proofs that the spender is the actual owner of the transferred assets, and the transaction has not been changed.

The serialization of this representation, as well as the private-key x, will then be used as inputs to the *Sign* algorithm of the selected signature scheme. The spender will then add the obtained signature as the last entry to the transaction. This value proves knowledge of the private-key x, which convinces the verifier that the user is indeed the owner of the spending account. The signature also provides integrity protection against tampering, because only unmodified transactions will result in a successful signature verification.

4.1.9 Send Transaction

The successful execution of the steps presented in subsection 4.1.1 - 4.1.8, will result in a transaction object as defined in subsection 4.1.8. As the last step, the spender proposes the constructed transaction for verification to the nodes.

4.2 Verifier

Nodes of the system will take care of the verification process, by checking proposed transactions for their validity. They have access to the current state, namely the encrypted balances of the users. On successful verification the nodes will perform a state transfer by applying an update to the involved users balances. These updates will be performed by utilizing the homomorphic property of ElGamal. The verified transaction will then be added to a block, which will then itself be added to the ledger.
The ledger contains the transaction history of every user up to the current point in time. The design of the blockchain ensures that no tampering of this history is possible. The current state of the system can then be verified by replaying every occurred transaction. In practice the verifying node will need to parse the serialized transaction-package to reconstruct the structure defined before. During the explanation the variable T will be used to describe the currently verified transaction. The notation $T[\lambda]$ will be used to describe a *field* access of transaction T. The system state will be defined by variable Θ, which defines a list of accounts. $\Theta[x]$ then represents the current balance of a user identified by address x. Additionaly the world state holds every users encryption and signature verification key.
The verification-process is defined by the following steps:

1. Check that spender and receiver are members of the system.
2. Verify the provided signature.
3. Verify all provided proofs.
4. Update the world state.

As for the spender these steps will now be concretized:

4.2.1 Address Verification

Verify that the values $T[from]$ and $T[to]$ are valid addresses, so these values describe valid identifiers for users of the system. On successful verification, proceed with the next step, otherwise reject the transaction.

4.2.2 Signature Verification

Verify the validity of the provided signature by using the stored public-key as well as the serialized transaction as message as input to the *Verify* algorithm described in subsection 2.10.1. On success, the verifier knows that the transaction has not been changed and was issued by the claimed spender. If the verification fails, the transaction will be rejected.

4.2.3 Proof Verifications

As stated multiple times throughout this thesis, the central part of the verification process will be dominated by proof verifications. These proofs are necessary in order to ensure that only correct values have been used to construct the transaction in the first place. A *malicious* transaction will only be considered valid with negligible probability, due to the properties of the used crypto and proof-systems. The process will be executed in a hierarchic manner, which means that once a check fails, the whole verification fails and hence results in a rejection of T.

The procedure depicted in Figure 4.4 illustrates the mentioned hierarchic verification process, which will be executed by the node step by step.
First it needs to be checked that the provided balance-encryption hides the same value as the one which is currently stored in the world state. Therefore the verifier will engage the subroutine, illustrated in Figure 4.5, with the calculate homomorphic difference between the given and saved encrypted balance, as described in subsection 4.1.7, as parameter. This statement is the main precondition of the verification process, because if this statement does not hold, there is no point in verifying the rest of the proofs.
Second the process will continue with the verification that the provided Pedersen-Commitment in C hides the same value as the encryptions in x_s and x_r. To do so the verifier engages the subroutine depicted in Figure 4.7.
Third the verification of the range-proof provided in the field ϕ, stating that $a \in [0, 2^{32})$ holds, will be performed. When both statements are valid the verifier will be convinced that x_s, x_r indeed hold the same value, which additionally lies in the required range. A balance update by homomorphic addition and subtraction will hence cause no integrity violations.

Paramters

```
$\mathbb{G}$: Group $\mathbb{G}$ of order $p$.
$g$: Public generator $g \in \mathbb{G}$
$h$: Public generator $h \in \mathbb{G}$
$p_k$: Public key of spender.
$T$ : Transaction
```

Proof Verifications:

```
S1. **abort** if: **not** VerifyEncryption($T[y]$, $\Theta[T[from]]$)
S2. **abort** if: **not** VerifyEquivalence($T[C], T[x_s], T[\pi_1], \mathbb{G}, g, h, p_k$)
S3. **abort** if: **not** VerifyEquivalence($T[C], T[x_r], T[\pi_2], \mathbb{G}, g, h, p_k$)
S4. **abort** if: **not** VerifyRange($T[C], T[\phi]$)
S5. **abort** if: **not** VerifyEquivalence($T[P], T[y], T[\mu], \mathbb{G}, g, h, p_k$)
S6. **abort** if: **not** VerifyRange($T[P], T[\rho]$)
S7. Set $V' \leftarrow T[P] \ominus T[C]$
S8. **abort** if: **not** VerifyRange($V', T[\tau]$)
S9. return $\top$
```

Figure 4.4: Verification process.

Furthermore, the verifier checks that P is indeed a commitment to the current balance. Due to the first step, the verifier knows that the spender provided a valid self-computed encryption of the balance, therefore this value can be used to check that $P \Leftrightarrow b$ holds. So as the next step the subroutine depicted in Figure 4.7 will be executed with P and y as parameters. If P is valid the actual sanity-check besides the existence of the spender and receiver can be performed: as stated multiple times a user may not transfer more assets, than he currently holds. If all previous checks are successful the verifier knows that the spender committed to a value a which is also present in the encryptions x_s, x_r, he further knows that P holds a commitment to the current balance. Pedersen-commitments, as shown in Definition 2.7.2, are linearly homomorphic, hence a commitment to the difference between b and a can be computed without knowing the actual plain values. The computed commitment can then be used to verify the necessary statement $a \leq b$.

For this to work the spender is required to construct a proof for commitment $V = g^{b-a}h^{\sigma-\gamma}$, where γ, σ are the used blinding factors in C and P, which

Input:

X: ElGamal cipher-text to verify, $X = (C_1, C_2)$.
α: Tuple (a, r) as described in Figure 4.3.

Verify Encryption:

S1. Verify that ${C_1}^r \overset{?}{=} a{C_2}^c$ holds, return $\top$ on success $\perp$ otherwise.

Figure 4.5: VerifyEncryption subroutine.

Input:

V: Commitment to verify, $V = g^a h^\gamma$.
β: Proof to verify against Commitment V.

Verify Range:

S1. Verify β return $\top$ if $V = g^a h^\gamma \land a \in [0, ..., 2^{32})$ holds, else return $\perp$.

Figure 4.6: VerifyRange subroutine.

are only know to the spender. If any other commitment has been used to construct the proof, this statement will only be considered valid with negligible probability.
The verification of the proof provided in the field τ is then the last step of the whole process and finally completes all integrity checks.
After all check have passed the node can safely proceed with the balance update, because the transaction transfers a valid amount.
The described checks will be performed in this hierarchic order, meaning that as soon as any of them fails the transaction will be rejected immediately.

4.2.4 Update World State

Update the system state by applying the actual balance update. This can be done by utilizing the linear homomorphic property of ElGamal. The spenders balance will be updated by *subtracting* the transferred amount from it. The receivers balance on the other hand will be increased by performing a homo-

Input:

C_3:	Pedersen commitment to verify.
X:	ElGamal cipher-text to verify, $X = (C_1, C_2)$.
β:	Tuple $((q_1, q_2, q_3); (z_1, z_2, z_3))$.
$\mathbb{G}$:	Group $\mathbb{G}$ of order p.
g:	Generator $g \in \mathbb{G}$.
h:	Generator $h \in \mathbb{G}$.
y:	Public key of spender.

Verify Range:

```
S1: set $c \leftarrow_p H(q_1, q_2, q_3)$
S2: verify $g^{z_1} y^{z_2} \overset{?}{=} q_1 C_2^c$, on fail return $\perp$
S3: verify $g^{z_1} h^{z_3} \overset{?}{=} q_2 C_3^c$, on fail return $\perp$
S4: verify $g^{z_2} \overset{?}{=} q_3 C_1^c$, on fail return $\perp$
S5: return $\top$
```

Figure 4.7: VerifyEquivalence subroutine.

morphic addition.
The current balance of spender and receiver will be denoted as $B_s = \Theta[from] = (g^u, g^{b_s} y_s^u)$ and $B_r = \Theta[to] = (g^v, g^{b_r} y_r^v)$. The respective encryption of a is further given as $x_s = (g^w, g^a y_s^w)$ and $x_r = (g^x, g^a y_r^x)$. Subscripts s, r indicate the corresponding values for spender and receiver and b_s, b_r denote the respective numeric balance.
By utilizing the property described in Equation 2.1 we obtain the following: $B_s \ominus x_s = (g^{u-w}, g^{b_s-a} y_s^{u-w})$ and $B_r \oplus x_r = (g^{v+x}, g^{b_r+a} y_r^{v+x})$, which are valid encryptions of the updated balances. These values g^{b_s-a}, g^{b_r+a} will then only be recoverable by decrypting the new cipher-texts using the corresponding private key. To obtain the updated numeric values $(b_s - a)$ and $(b_r + a)$ the algorithm described in algorithm 1 will be used by each user.
Additionally the node will formulate a new block including the valid transaction T. To complete the state transfer, this block will then be added to the ledger. Then newly added block will then define the new state.

5 Implementation Results

In this section, the actual implementation and its results will be presented. Also, some general evaluations in terms of necessary elliptic curve operations, as well as dedicated benchmarks, will be presented.

5.1 General Setting

The whole protocol presented in chapter 4 has been implemented as a C/C++ library, using the 2011 standard for C and the 2017 standard for C++. The main goal of the implementation was to evaluate the transaction composition and verification in terms of performance and implementability, so no actual payment system has been implemented. Note that the provided library may be used to implement an actual distributed ledger-based payment system.
To keep track of all users and their balances, the class *bc_ system* can be used. This class takes serialized transactions as input and verifies them, by executing the process described in section 4.2; on success, the involved users balances will be updated by utilizing the homomorphic property of ElGamal encryptions.

Remarks Our implementation is not capable of instantiating a fully operational payment system. The goal of this thesis is to demonstrate how the presented protocol could be used. However, our work could be extended to represent a fully operational distributed ledger-based payment system.

5.2 Implementation Framework

Many of the used primitives, such as ElGamal or Pedersen, as well as the presented proof-schemes, require a cyclic-group setting. The implementation of these schemes has been done using the elliptic curve *secp-k1* as defined in [Bro10].
As stated the protocol will use range-proofs to preserve the integrity of the ledger. For this purpose, we used the reference C-implementation of the bulletproof protocol, licensed under MIT [Ele], which also operates on the curve *secp-k1*. This library already provides the construction and verification of bulletproofs, as well as an implementation of Pedersen-commitments, and Schnorr-signatures. The library itself also implements elliptic curve operations, which are not exposed by the API. In order to implement the necessary schemes used in chapter 4, the library has been adapted in a way such that a dedicated interface for elliptic curve operations is provided.
For serialization and storage purposes, the C++ JSON-library [Loh] licensed under MIT is used. The provided benchmark results of our implementation have been generated by using the library [Goo], licensed under Apache-2.0.

5.3 Evaluation Results

In this section the actual implementation results will be presented. We will focus on the abstract performance of this protocol in terms of elliptic curve operations, such as point addition and scalar point multiplication per routine. The presented results also include a dedicated performance evaluation in terms of runtime in milliseconds, which have been performed on a i7-4790 CPU at 3.60GHz.
The transaction construction and verification process is split into several subroutines. Each subroutine will have a different impact on the whole protocol performance. We will therefore take a look at the significant parts, to provide a good overview on how these routines influence the overall performance.

First we want to evaluate the range-proof operations. Table 5.1 and Table 5.2 show the number of EC-operations needed during construction and verification, depending on the bit-width of the proven value.

bit-width	point-add	scal-mult	ms
8	4	28	1.19
16	8	56	2.36
32	16	112	4.71
48	24	168	7.12
64	32	224	9.42

Table 5.1: Bulletproof: Construct EC Operations.

bit-width	point-add	scal-mult	ms
8	0	16	1.03
16	0	32	2.06
32	0	64	4.13
48	0	96	6.21
64	0	128	8.27

Table 5.2: Bulletproof: Verify EC Operations.

We observe that the number of required operations grows linearly in the bit-width of the proven values.
The transaction build process also requires the construction of equivalence proofs, as defined in section Figure 4.3. The evaluation of the respective subroutine implementation is depicted in Table 5.3.

	point-add	scal-mult	ms
prove	2	5	0.301
verify	5	8	0.415

Table 5.3: Equivalence Proof: EC Operations.

By definition these subroutine is static in the number of operations. Further the spender needs to include an encryption proof as defined in subsection 4.1.7. The results for this routine can be found in Table 5.4.

	point-add	scal-mult	ms
prove	0	1	0.091
verify	1	2	0.125

Table 5.4: Encryption Proof: EC Operations.

Like the previous routine, this part is also static in the number of operations by definition. From the presented data we observe that the transaction construction and verification will be dominated by bulletproof operations. Due to this fact one may consider to restrict the allowed bit-widths, to maximize the number of processable transaction.
The illustrations in Table 5.5 and Table 5.6, conclude the overall results regarding the complete process. These tables show how many elliptic curve operations are necessary to construct and verify transactions with different balance and amount bit-widths as inputs.

balance-bw	amount-bw	point-add	scal-mult	ms
16	16	35	196	10.7
32	16	51	308	13.4
32	32	59	364	16.2
48	16	75	403	24.2
48	32	83	459	26.5

Table 5.5: Construct Transaction: EC Operations.

balance-bw	amount-bw	point-add	scal-mult	ms
16	16	19	123	7.77
32	16	19	183	12.8
32	32	19	215	15.0
48	16	19	251	16.3
48	32	19	283	18.7

Table 5.6: Verify Transaction: EC Operations.

To get an idea of how these numbers are obtained, consider the concrete number of subroutine-executions as described in section chapter 4. The spender needs to

execute the equivalence-proof subroutine three times and the encryption-proof subroutine once. Further, the spender computes two Pedersen-commitments and three ElGamal encryptions as well as one Schnorr-signature.
The verifier, on the other hand, needs to compute two homomorphic commitment subtractions as well as one point addition during the signature verification process.
These executions result in a static number of elliptic curve operations, namely 28 scalar multiplications and 11 point additions for construction and 27 scalar multiplications and 19 additions for verification. The remaining number operations will be dominated by bulletproof constructions and verifications, which depend on the bit-width of the proven values.
On our test machine, we would achieve a throughput of about 128 transactions per second, for balances and amounts representable with 16 bits. Considering 48-bit balances and 32-bit amounts, the throughput would be 53 transactions per second.
Keep in mind that the verification process is implemented on a single thread, and could hence be improved by splitting the work onto multiple threads.

5.4 Transaction Size

In this section, we will provide an overview of how different input sizes impact the overall transaction size. While investigating this fact, we observed that the size of the corresponding range-proofs depends on the bit-with of the input values. The transaction structure defined in subsection 4.1.8,can hence be split, regarding the influence on the overall transaction size, into a static and dynamic part. The static sizes of a transaction, presented in bytes, can be found in Table 5.7.

	size	count	total
sha-256-hash	32	2	64
Pedersen-cm	33	2	66
Schnorr-sig	64	1	64
enc-proof	96	1	96
ElGamal-ct	128	3	384
eq-proof	288	3	864

Table 5.7: Static sizes of transaction parts.

The values presented above, result in a static size of **1538** bytes for confidential transactions of the defined structure, presented in subsection 4.1.8.
The *dynamic* size-part, on the other hand, will be made up by the range-proofs. Table 5.8 shows how different input bit-widths influence the corresponding range-proof size.

bit-width	size
8	643
16	1283
32	2564
48	3845
64	5126

Table 5.8: Range Proof sizes depending on input bit-width.

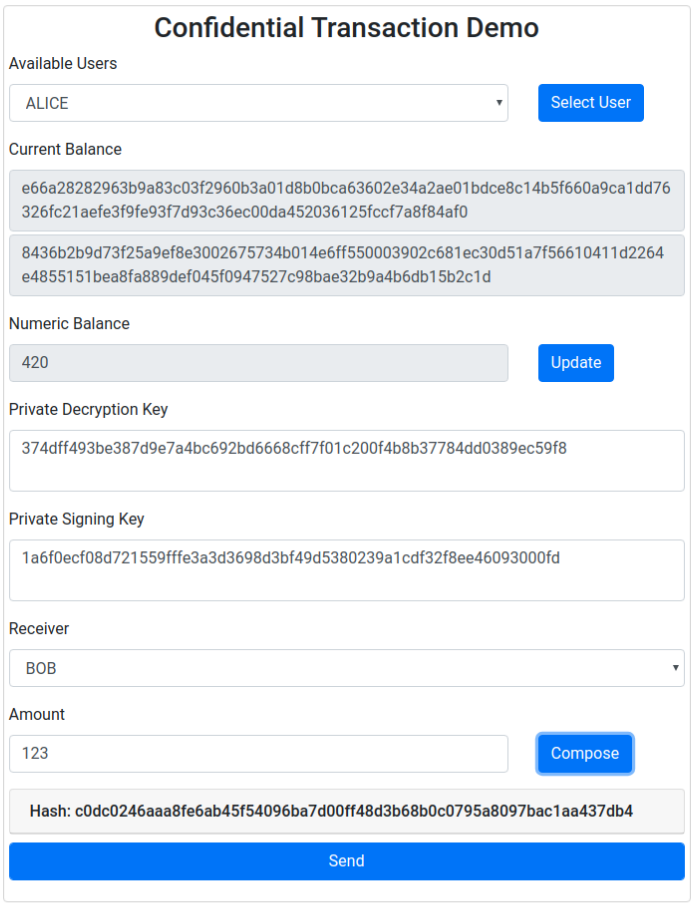
From the data presented above, we observe that the range-proof size grows linearly in the input bit-width. Depending on the actual proven values, a transaction's total size thus ranges from **3,467 KB** to **16,916 KB**.
Keep in mind that not all involved values must be of the same bit-width. Hence the representability, of balances and amounts in terms of bit-widths, heavenly impacts the overall transaction size. Table 5.9 illustrates how different bit-widths impact the overall transaction size.
In a setting, where transactions need to be stored on the ledger, one may consider only to allow transaction amounts which are representable in a certain bit-width, to keep the storage overhead as low as possible.

balance-bw	amount-bw	proof-size	total
16	16	3849	5227
32	16	6411	7949
32	32	7692	9239
48	16	8973	10511
48	32	10254	11792
48	48	11535	13073

Table 5.9: Transaction sizes depending on balance and amount bit-width.

5.5 Graphical Demo

To get a better feeling for the practical usage of this protocol, also a graphical version building upon the developed library has been implemented. An illustration of the implemented interface is depicted in Figure 5.1.


Confidential Transaction Demo

Available Users

ALICE

Current Balance

e66a28282963b9a83c03f2960b3a01d8b0bca63602e34a2ae01bdce8c14b5f660a9ca1dd76
326fc21aefe3f9fe93f7d93c36ec00da452036125fccf7a8f84af0

8436b2b9d73f25a9ef8e3002675734b014e6ff550003902c681ec30d51a7f56610411d2264
e4855151bea8fa889def045f0947527c98bae32b9a4b6db15b2c1d

Numeric Balance

420

Private Decryption Key

374dff493be387d9e7a4bc692bd6668cff7f01c200f4b8b37784dd0389ec59f8

Private Signing Key

1a6f0ecf08d721559fffe3a3d3698d3bf49d5380239a1cdf32f8ee46093000fd

Receiver

BOB

Amount

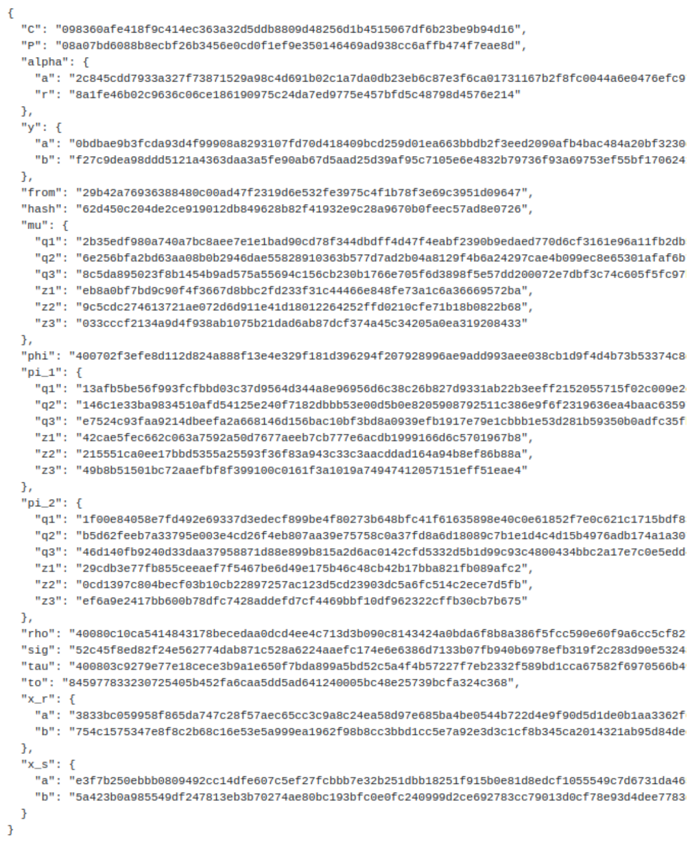
123

Hash: c0dc0246aaa8fe6ab45f54096ba7d00ff48d3b68b0c0795a8097bac1aa437db4

Figure 5.1: Interface to construct and send transactions.

One may select a provided user of the system by choosing from the dropdown. The field current balance will then be filled with the currently stored ElGamal cipher-text of the selected user. By providing the secret decryption-

and signing-key, a click on update will decrypt the current balance, and the according numeric value will be recovered by executing the algorithm described in algorithm 1.
The receiver of the transaction can be selected from the provided dropdown, which shows all other users of the system. As the last step, the numeric amount to be transferred needs to be provided.
With a click on compose, the steps described in subsection 4.1.1-4.1.8 will be executed. The constructed transaction can then be examined by clicking on the hash value. An example of a transaction representation can be found in Figure 5.2[1]. With a click on send, the transaction will be proposed for verification.



```
{
  "C": "098360afe418f9c414ec363a32d5ddb8809d48256d1b4515067df6b23be9b94d16",
  "P": "08a07bd6088b8ecbf26b3456e0cd0f1ef9e350146469ad938cc6affb474f7eae8d",
  "alpha": {
    "a": "2c845cdd7933a327f73871529a98c4d691b02c1a7da0db23eb6c87e3f6ca01731167b2f8fc0044a6e0476efc9
    "r": "8a1fe46b02c9636c06ce186190975c24da7ed9775e457bfd5c48798d4576e214"
  },
  "y": {
    "a": "0bdbae9b3fcda93d4f99908a8293107fd70d418409bcd259d01ea663bbdb2f3eed2090afb4bac484a20bf3230
    "b": "f27c9dea98ddd5121a4363daa3a5fe90ab67d5aad25d39af95c7105e6e4832b79736f93a69753ef55bf170624
  },
  "from": "29b42a76936388480c00ad47f2319d6e532fe3975c4f1b78f3e69c3951d09647",
  "hash": "62d450c204de2ce919012db849628b82f41932e9c28a9670b0feec57ad8e0726",
  "mu": {
    "q1": "2b35edf980a740a7bc8aee7e1e1bad90cd78f344dbdff4d47f4eabf2390b9edaed770d6cf3161e96a11fb2db
    "q2": "6e256bfa2bd63aa08b0b2946dae55828910363b577d7ad2b04a8129f4b6a24297cae4b099ec8e65301afaf6b
    "q3": "8c5da895023f8b1454b9ad575a55694c156cb230b1766e705f6d3898f5e57dd200072e7dbf3c74c605f5fc97
    "z1": "eb8a0bf7bd9c90f4f3667d8bbc2fd233f31c44466e848fe73a1c6a36669572ba",
    "z2": "9c5cdc274613721ae072d6d911e41d18012264252ffd0210cfe71b18b0822b68",
    "z3": "033cccf2134a9d4f938ab1075b21dad6ab87dcf374a45c34205a0ea319208433"
  },
  "phi": "400702f3efe8d112d824a888f13e4e329f181d396294f207928996ae9add993aee038cb1d9f4d4b73b53374c8
  "pi_1": {
    "q1": "13afb5be56f993fcfbbd03c37d9564d344a8e96956d6c38c26b827d9331ab22b3eeff2152055715f02c009e2
    "q2": "146c1e33ba9834510afd54125e240f7182dbbb53e00d5b0e8205908792511c386e9f6f2319636ea4baac6359
    "q3": "e7524c93faa9214dbeefa2a668146d156bac10bf3bd8a0939efb1917e79e1cbbb1e53d281b59350b0adfc35f
    "z1": "42cae5fec662c063a7592a50d7677aeeb7cb777e6acdb1999166d6c5701967b8",
    "z2": "215551ca0ee17bbd5355a25593f36f83a943c33c3aacddad164a94b8ef86b88a",
    "z3": "49b8b51501bc72aaefbf8f399100c0161f3a1019a74947412057151eff51eae4"
  },
  "pi_2": {
    "q1": "1f00e84058e7fd492e69337d3edecf899be4f80273b648bfc41f61635898e40c0e61852f7e0c621c1715bdf8
    "q2": "b5d62feeb7a33795e003e4cd26f4eb807aa39e75758c0a37fd8a6d18089c7b1e1d4c4d15b4976adb174a1a30
    "q3": "46d140fb9240d33daa37958871d88e899b815a2d6ac0142cfd5332d5b1d99c93c4800434bbc2a17e7c0e5edd
    "z1": "29cdb3e77fb855ceeaef7f5467be6d49e175b46c48cb42b17bba821fb089afc2",
    "z2": "0cd1397c804becf03b10cb22897257ac123d5cd23903dc5a6fc514c2ece7d5fb",
    "z3": "ef6a9e2417bb600b78dfc7428addefd7cf4469bbf10df962322cffb30cb7b675"
  },
  "rho": "40080c10ca5414843178becedaa0dcd4ee4c713d3b090c8143424a0bda6f8b8a386f5fcc590e60f9a6cc5cf82
  "sig": "52c45f8ed82f24e562774dab871c528a6224aaefc174e6e6386d7133b07fb940b6978efb319f2c283d90e5324
  "tau": "400803c9279e77e18cece3b9a1e650f7bda899a5bd52c5a4f4b57227f7eb2332f589bd1cca67582f6970566b4
  "to": "845977833230725405b452fa6caa5dd5ad641240005bc48e25739bcfa324c368",
  "x_r": {
    "a": "3833bc059958f865da747c28f57aec65cc3c9a8c24ea58d97e685ba4be0544b722d4e9f90d5d1de0b1aa3362f
    "b": "754c1575347e8f8c2b68c16e53e5a999ea1962f98b8cc3bbd1cc5e7a92e3d3c1cf8b345ca2014321ab95d84de
  },
  "x_s": {
    "a": "e3f7b250ebbb0809492cc14dfe607c5ef27fcbbb7e32b251dbb18251f915b0e81d8edcf1055549c7d6731da46
    "b": "5a423b0a985549df247813eb3b70274ae80bc193bfc0e0fc240999d2ce692783cc79013d0cf78e93d4dee7783
  }
}
```

Figure 5.2: Illustration of a constructed confidential transaction.

[1]Note that the range proofs value is cut for better readability

The system will then verify the validity of the transaction as described in Procedure Figure 4.4. On success a message will be shown indicating that the transaction is valid and a state transfer has been performed.
If the transaction was rejected, an according error message indicating why the transaction has been considered invalid will be presented. The state transfer can be observed by clicking on the update button, which again retrieves the current balance from the system and recovers the according numeric value by executing the algorithm described in algorithm 1.

6 Conclusion and Further Work

The goal of this thesis was to develop a confidential transaction protocol for a distributed ledger-based asset system. In the general setting, a user needs to provide sensitive information such as the transaction amount and his current balance in plain form to allow public verification at a later point in time. We showed how this necessity could be eliminated by utilizing homomorphic cryptosystems as well as zero-knowledge proofs.

Every sensitive information will now be present in encrypted form, meaning that only the user itself will be able to decipher this information again. To still preserve the ability of public verification, we showed how zero-knowledge protocols could be used to convince the verifier that the necessary computations have been performed correctly. We also showed how homomorphic cryptosystems could be used in this context to update persistent values such as a user's balance, without decrypting the values in the process.
Users may transfer numeric assets secretly to any other user, by following the steps described in chapter 4. The transaction will not contain any plain information about the secret values, but it can still be verified at a later point in time. Further, a verifier will not learn anything besides the validity of the proposed transaction.
To demonstrate the performance and applicability of the described protocol, we also provided an implementation in the form of a C++ library. This implementation has been analyzed in terms of computational overhead; the according results can be found in section chapter 5.

Further work may take a look into how the described protocol may be improved. For this purpose, one may consider other homomorphic encryption schemes, which do not require a discrete-logarithm recovery. Hence ElGamal could be replaced by Paillier [Pai99] or another suitable cryptosystem. The flat performance may also be improved by evaluating the influence of different underlying elliptic curves.

The described protocol enhances the confidential regarding numeric assets transactions, but systems may hold assets that are not representable in numeric form. Due to [GMW91] we know that every statement in NP can be proven in zero-knowledge. So the protocol could be extended by also considering proof-schemes that apply to other types of assets.
Additionally the protocol could be extended such that it allows users to prove that a specific transaction has not exceeded the tax-free amount. This may be necessary when thinking about asset transfers between trade areas with different regulations. An extension may also include the possibility for a user to prove that its current balance independent from a transaction, which could especially be useful when a user does its taxes. For this purpose, a certificate could be designed, which can then be verified by the regulation authority at a later point in time.
In any way we showed how to establish *confidentiality* in the setting of a distributed ledger-based asset system by hiding any sensitive information. The validity of these hidden values is proven in zero-knowledge, which allows verification at a later point in time without the presence of any sensitive information.

Limitations The presented protocol only enhances the confidentiality of the users, so no curious party will be able to deduce any information about the transferred amounts. However, even though no direct information about the users is present, the transaction habits can still be tracked, and at a later point in time, a user's identity may still be compromised.

Related Work and Directions There exist other approaches like Mimblewimble [Poe16], ZeroCash, [Ben+14] and [Max], which improve the security of users in terms of privacy and confidentiality. Anyhow these approaches are only defined in the setting of a UTXO based transaction model, while the presented protocol in chapter 4 is applicability to both approaches.
To improve the protocol regarding the problem of traceability and likability, an approach similar to the cryptocurrency Monero [Sab13] could be considered. Here untraceability is achieved by utilizing ring-signatures, as described in [RST01]. The idea is to sign a transaction on behalf of a *group* of users. A

verifier will then be convinced that one user of the group signed the transaction while not knowing who exactly constructed it. This approach makes it nearly impossible to trace a user's transaction habit because, with increasing transactions, the number of possible user interactions grows exponentially. The unlinkability, on the other hand, is solved by describing one-time receiving address; for more details on how this property is achieved see [Sab13].

In the last years many protocols and new approaches on how security can be enhanced in distributed ledger-based payment systems have emerged. The growing interest in these systems and the accompanying increase in users may also make it more attractive for parties with malicious intent to gain profit out of vulnerable designs. The development and improvement of new and already established solutions, regarding confidentiality and privacy in a distributed ledger-based asset system, is thus an essential field of research.

Bibliography

[Ben+14] Eli Ben-Sasson et al. “Zerocash: Decentralized anonymous payments from Bitcoin”. In: *Security and Privacy (SP), 2014 IEEE Symposium on. IEEE*. 2014 (cit. on p. 75).

[BL12] Daniel J. Bernstein and Tanja Lange. “Computing Small Discrete Logarithms Faster”. In: *Progress in Cryptology - INDOCRYPT 2012*. Ed. by Steven Galbraith and Mridul Nandi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 317–338. ISBN: 978-3-642-34931-7 (cit. on p. 22).

[Boo+16] Jonathan Bootle et al. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 327–357. ISBN: 978-3-662-49896-5 (cit. on pp. 29, 30).

[Bro10] Daniel R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. `https://www.secg.org/sec2-v2.pdf`. Version 2. 2010 (cit. on p. 65).

[Bün+18] B. Bünz et al. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. May 2018, pp. 315–334. DOI: `10.1109/SP.2018.00020` (cit. on pp. 8, 29, 30, 33, 35, 52).

[But14] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014. URL: `https://github.com/ethereum/wiki/wiki/White-Paper` (cit. on pp. 1–3, 41).

[DH76] W. Diffie and M. Hellman. “New directions in cryptography”. In: *Information Theory, IEEE Transactions on* 22.6 (1976), pp. 644–654 (cit. on pp. 16, 35).

[Ele] Andrew Poelstra ElementsProject. *Optimized C library for EC operations on curve secp256k1*. `https://github.com/ElementsProject/secp256k1-zkp` (cit. on p. 65).

[ElG85] Taher ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *Advances in Cryptology*. 1985, pp. 10–18 (cit. on pp. 8, 16).

[FS87] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Advances in Cryptology—CRYPTO'86*. 1987, pp. 186–194 (cit. on p. 28).

[GMR89] Shafi. Goldwasser, Silvio. Micali, and Charles. Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: `10.1137/0218012`. eprint: `https://doi.org/10.1137/0218012`. URL: `https://doi.org/10.1137/0218012` (cit. on p. 24).

[GMR95] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*. 1995 (cit. on p. 35).

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems". In: *Journal of the ACM (JACM)* 38.3 (1991), pp. 690–728 (cit. on pp. 29, 75).

[Goo] Google. *A microbenchmark support library*. `https://github.com/google/benchmark` (cit. on p. 65).

[Kob87] Neal Koblitz. "Elliptic Curve Cryptosystems". In: *Math. Comp.* 48 (1987), pp. 203–209 (cit. on p. 12).

[Loh] Niels Lohmann. *JSON for modern C++*. `https://github.com/nlohmann/json` (cit. on p. 65).

[Max] Greg Maxwell. *Confidential Transactions*. `https://people.xiph.org/~greg/confidential_values.txt` (cit. on p. 75).

[Mer88] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology — CRYPTO '87*. Ed. by Carl Pomerance. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378. ISBN: 978-3-540-48184-3 (cit. on p. 13).

[Nak08] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: *www.bitcoin.org* (2008) (cit. on pp. 1, 14).

[NIS15] NIST. “Secure Hash Standard (SHS)”. Version 4. In: *FIPS PUB* 180 (2015) (cit. on p. 13).

[Pai99] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *Advances in cryptology—EUROCRYPT’99*. 1999, pp. 223–238 (cit. on p. 74).

[Ped92] Torben Pryds Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing”. In: *Advances in Cryptology—CRYPTO’91*. 1992, pp. 129–140 (cit. on pp. 8, 20).

[Poe16] Andrew Poelstra. *Mimblewimble*. `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf`. 2016 (cit. on p. 75).

[Qui+89] Jean-Jacques Quisquater et al. “How to Explain Zero-knowledge Protocols to Your Children”. In: *Proceedings on Advances in Cryptology*. CRYPTO ’89. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1989, pp. 628–631. ISBN: 0-387-97317-6. URL: `http://dl.acm.org/citation.cfm?id=118209.118269` (cit. on p. 23).

[RSA78] Ronald L Rivest, Adi Shamir, and Len Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on p. 15).

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. “How to Leak a Secret”. In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565. ISBN: 978-3-540-45682-7 (cit. on p. 75).

[Sab13] Nicolas van Saberhagen. *CryptoNote v2.0*. `https://cryptonote.org/whitepaper.pdf`. 2013 (cit. on pp. 75, 76).

[Sch19] Berry Schoenmakers. *Lecture notes cryptographic protocols*. `https://www.win.tue.nl/~berry/CryptographicProtocols/`. 2019 (cit. on pp. 24, 26, 36, 37).

[Sch91] C.P. Schnorr. “Efficient signature generation by smart cards”. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174 (cit. on pp. 24, 26).

[Woo17] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION (759dccd - 2017-08-07)*. 2017. URL: `https://ethereum.github.io/yellowpaper/paper.pdf` (cit. on p. 2).