A Author’s Publications

Patent Applications

N. Liebau, V. Darlagiannis, and A. Mauthe: *Ein dezentrales, token-basiertes Accountingsystem für verteilte, autonome Systeme*. European patent application, No. 04 101 386.3

Journal Articles


Conference Contributions


**Book Chapters**


**Technical Reports/Miscellaneous**


# B Curriculum Vitae

## Personal Details

<table>
<thead>
<tr>
<th>Name</th>
<th>Nicolas Christopher Liebau</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/place of birth</td>
<td>October 24th, 1973/Fulda, Germany</td>
</tr>
<tr>
<td>Nationality</td>
<td>German</td>
</tr>
</tbody>
</table>

## Education

<table>
<thead>
<tr>
<th>Since 07/2002</th>
<th>To be obtained: Doktor-Ingenieur (Dr.-Ing.) in Electrical Engineering and Information Technology.</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/2006 – 09/2006</td>
<td>Visiting researcher at Lancaster University, UK with Dr. Andreas Mauthe.</td>
</tr>
<tr>
<td>08/1998 – 05/1999</td>
<td>Study abroad in the MBA-Program of the State University of New York at Buffalo, USA.</td>
</tr>
<tr>
<td></td>
<td>Degree: Diplom-Wirtschaftsingenieur (Dipl.-Wirts.-Ing.) in Electrical Engineering and Information Technology.</td>
</tr>
<tr>
<td></td>
<td>Degree: Allgemeine Hochschulreife.</td>
</tr>
<tr>
<td>Date</td>
<td>Position</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Since 10/2006</td>
<td>Head of the P2P Networking research group of the Multimedia Communications Lab (KOM), Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Germany</td>
</tr>
<tr>
<td>Since 07/2005</td>
<td>Researcher at the Multimedia Communications Lab (KOM), Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Germany</td>
</tr>
<tr>
<td>10/2002 – 06/2006</td>
<td>Head of computer administration group at the Multimedia Communications Lab (KOM), Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Germany</td>
</tr>
<tr>
<td>07/2002 – 06/2005</td>
<td>Ph.D student with scholarship in the DFG Research Training Group „System Integration for Ubiquitous Calculations in the Information Technology“ at the Multimedia Communications Lab (KOM), Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Germany</td>
</tr>
<tr>
<td>06/2001 – 08/2001</td>
<td>Internship at Accenture GmbH, Sulzbach/Taunus, Germany. Member of Operational Readiness Test Team on a project with the goal to build the functional foundation for the market presence of a new banking institute.</td>
</tr>
<tr>
<td>05/1999 – 08/1999</td>
<td>Internship at HR Foundations, Inc., Buffalo, NY, USA. Team member for designing and conducting Compensation &amp; Salary Surveys.</td>
</tr>
</tbody>
</table>
Supervision of Diploma-, Master-, Bachelor-, and Student-Theses


C Supplementary Details to Chapter 2

C.1 Definitions in Context

**Autonomous Distributed Systems**

Autonomous Distributed systems are distributed systems (CDK02) where the participating entities are highly autonomous. Autonomous distributed systems include, e.g., p2p systems. Another example is a system of servers, where the servers are controlled by different parties and mutual trust does not exist.

**Security**

Security of information systems is a well defined term (see, e.g., (Eck04, Buc03, Sch96)). Eckert (Eck04) distinguishes functional security, information security, data protection, and privacy, where functional security is the basis of the other security terms. Typically, in information systems the term security refers to information security.

Functional security is the attribute of a system that the realised as-is functionality of the system components is consistent with the specified to-be functionality.

Information security (security) is the attribute of a functional secure system to take only those states which do not lead to any unauthorised modification of information or access to information.

Data protection is the attribute of a functional secure system to take only those states which do not lead to any unauthorised access to system resources and especially to data. Therefore, data protection also includes measures against loss of data like data backup.

Privacy is the ability of a physical person to control the propagation of data pertaining to that user.

For information security and data protection, there are six protection objectives defined. These are authentication, integrity, confidentiality, improper flow of information, availability, non-repudiation, anonymity, and pseudonymity.

**Trustworthiness**

In (Sta02) trustworthiness is described as “... an information system’s accountability and its ability to produce reliable and authentic information and records. ... [Trustworthiness] denotes integrity, ability, faith, and confidence. We use trustworthiness to describe information system accountability.” Accordingly, trustworthiness expresses the trust a user has in an information system. This includes security. In this definition accountability is an important term. Thus, it is not well suited for defining a trustworthy accounting system.

In (BHS02) Belanger states the three main elements of trust: ability, benevolence, and integrity. These elements correspond to characteristics of a merchant. Transferred to information systems, ability means the ability of an information system to fulfil the functionality it was designed for. Ability implies that the information system was designed with expertise in the relevant functional areas. Integrity refers to information security as defined above. Benevolence is defined as the extent to which the users believe that the application provider wants to do good things rather than just profit.
In information systems the term security refers typically to information security. However, for a trustworthy system this is incomplete, because aspects of functional security are missing. Therefore, in information systems the term trusted entities is used. It comprises also that the as-is functionality of the system components is consistent with system specifications. Belanger further adds benevolence as an element of a trusted information system.

This is an important addition for p2p systems, since the good intentions of all participating peers cannot be assumed. Thus, the challenge for p2p systems is to design a system that offers benevolence, although its components (peers) might not act honestly. Hence, potential cheating and collusion of peers must be considered in the design. In consequence, to design trusted p2p systems the application of state-of-the-art security mechanism will probably not be sufficient.

In sum, a trusted p2p system is designed with good intentions, the as-is functionality of the system is consistent with its specifications, and is secure without relying on (a) trusted entity(ies).

C.1.1 Definitions Related to Token-Based Accounting

Accounting

Accounting throughout this dissertation is defined as the process of tracing relevant IS activities to a responsible source. Relevance is defined by the application applying accounting (see Chapter 2, Section 2.1.2).

Account Holder

An account holder is a peer storing one or more aggregation accounts. See Aggregation Account as well as Account Holder Set.

Account Holder Set (AHS)

The account holder set (AHS) is a set of peers storing a specific aggregation account. A peer can be member of several account holder sets. Due to churn in the p2p system the membership in account holder sets is dynamic and changes over time as well as the exact location of an account holder set in the p2p system. Account holder sets are maintained using a set of maintenance mechanisms. The account holder set size (AHSS) is given the variable $k$ throughout the dissertation. For details see Chapter 4, Section 4.2.

Aggregation Account

Each peer that is using the token-based accounting scheme requires an aggregation account. In the aggregation account the tokens issued to the peer are administered, that is the actual status of each token is stated. Aggregation accounts are located at remote peers, the so called account holder set. For details see Chapter 3, Section 3.6 as well as Chapter 4, Section 4.2.

Aggregation Function

See Token Aggregation.
Quorum

The quorum of trusted peers is a randomly selected group of trusted peers that signs new tokens with their share of the token-based accounting scheme’s private key during the token aggregation process in order to issue tokens. Using threshold cryptography an unanimous decision making of the quorum is implemented. The quorum size is given the variable \( t \) throughout the dissertation. Further details are described in Chapter 3, Section 3.2.

Token

Tokens are data objects that serve as a combination of transaction receipts and permission objects within the token-based accounting scheme. Tokens are issued to peers using the token aggregation process. During transactions tokens are spent in return for receiving an accounted service or accounted resources. For details see Chapter 3, Section 3.2.

Token Aggregation

For providing accounted services and accounted resources peers receive tokens belonging to other peers, so-called foreign tokens. Using the token aggregation process peers swap these foreign tokens against new own tokens they can use in order to receive accounted services and resources. The token aggregation process computes the number of new tokens a peer should receive using the aggregation function. New tokens are signed by a quorum of trusted peers with the token-based accounting scheme’s private key using threshold cryptography. For details see Chapter 3, Section 3.5.

Trusted Peer

A trusted peer is in possession of one share of the token-based accounting scheme’s private key. This key is shared among all trusted peers and signatures are created by a quorum of trusted peers using threshold cryptography. Trusted peers are selected based on their reputation value. Details about trusted peers are described in Chapter 4, Section 4.3.

C.1.2 Definitions Related to Threshold Cryptography

Secret \((s)\)

A secret denotes a secret key. A secret key is an encryption key that is kept concealed. Its discovery voids the security of the encryption session. A secret key generally refers to the key in a secret key cryptography system, in which both sides use the same key. It may also refer to the private key in a public key cryptography system, because the private key must also kept secret. (From (THe06), Secret Key).

Share \((s_i)\)

The share \(s_i\) is a mathematical derived value of the secret \(s\). A predefined number of the shares are needed to reconstruct the secret. These shares have to be stored secretly. An important property of these
shares is that no information about the secret is revealed, if less than the predefined number of shares is known by an attacker. (cf. (HJKY95).)

**Dealer (D)**

A dealer (of a \((k, n)\)-threshold scheme) owning a secret \(s\) wishes to distribute knowledge of \(s\) (shares) among a group of shareholders such that two conditions hold:
- **Correctness:** Any subset of \(k\) shareholders can together recover \(s\)
- **Security:** Any subset of less than \(k\) shareholders cannot recover \(s\) The dealer is only used to share the secret \(s\) and destroys its information if the shares are distributed correctly. (cf. (PSW04).)

**Shareholder \((P_i)\)**

A shareholder \(P_i\) receive a unique share from the dealer \(D\) and stores it securely. To reconstruct the secret \(s\) the shareholders are consulted, which deliver the shares for reconstruction (cf. (WWW02)).

**\((k, n)\)-Threshold Scheme**

A \((k, n)\)-threshold scheme is a way of distributing partial information called shares to \(n\) participants in order to allow any \(k\) of them to make an action (e.g., to find a secret \(s\) or to open a vault in a bank), but also to ensure that the action cannot be made by any subset of fewer than \(k\) participants. (From (TMO005).)

**Secret Sharing**

A trusted dealer \(TP\) gives each player \(P_i\) a secret share \(ss_i\) of the secret \(s\) in such a way, that any group of \(t\) or more players can together reconstruct the secret, but no group fewer then \(t\) players can do this. Here, \(t\) is denoted the threshold and \(n\) is the total number of players.

**Verifiable Secret Sharing**

A secret sharing scheme is verifiable secret sharing scheme (VSS) if additional information is made available to the players that enables the players to verify that their shares are consistent. I.e. even when the dealer is malicious there is a well defined secret that the players can reconstruct.

**Proactive Secret Sharing**

A **Proactive Secret Sharing** scheme enhances threshold schemes by periodic refreshment of the shared function in such a way that the secret kept unchanged. Breaking of the system requires the attacker to break into several shareholders in a short period of time. (HJKK97)

**Initialisation Phase**

In *initialisation phase* the secret \(s\) is encoded into \(n\) shares using a \((k, n)\)-threshold scheme. These shares are then distributed securely to the shareholders. At the end of this phase each good shareholder has a
valid and verified share. With this share the shareholder is able to participate the reconstruction phase. (cf. (HJKY95, NNPV02b).)

### Reconstruction Phase

In *reconstruction phase* a defined coalition of shareholders cooperate to reconstruct the secret or to generate a signature. Therefore, the corresponding reconstruction protocol is performed. (NNPV02a)

### Update Phase

Within the *update phase* the shares of the shareholders are updated without changing the secret $s$. This phase is performed periodically. At the end of the update phase the updated shares correspond to the secret $s$. An attacker that knows no more than $k$ shares from different update periods learns nothing about the secret. (HJKY95)

### Recovery Phase

The *recovery phase* is used to recover malicious shareholders. First all corrupted shareholders are identified and then recovered. For recovery a coalition of good shareholders has to cooperate to generate a new share. (HJKY95)

### Passive Adversary

A passive adversary obtains the complete information held by the attacked shareholder, but the attacked shareholder executes the protocol correctly. (NN04)

### Active Adversary

An active adversary takes full control of the attacked shareholder and is able to change the behaviour of the shareholder. (NN04)

### Mobile Adversary

A mobile (active) adversary can corrupt a shareholder during any time period. Corrupting a shareholder means any combination of learning the secret information, modifying its data, changing its intended behaviour, disconnection it from communication channel, etc. The adversary is connected to the broadcast channel, which means that he can hear all broadcasted messages or broadcast his own (DoS-attack possible). He is not able to modify messages sent between shareholders that he does not control, nor can he pretend non-crupted shareholders from receiving a broadcasted message. (Additionally, the adversary is computationally bounded, so that it cannot break the underlying cryptographic primitives (e.g., solving the discrete logarithm problem)) (HJKK97).

### C.1.3 Definitions of Related Forms of Attacks in Context

This section summarises the forms of attack that are especially relevant the context of the thesis.
Cheating

Cheating is an attack where a peer improves its own situation in the p2p system, e.g., by spreading false information or by using an alternative protocol.

Within the context of the token-based accounting scheme cheating is every form behaviour where a peer aims at increasing it amount of tokens in a wrongful way.

Collusion Attack

In an collusion attack a number of peers collaborate with the goal to defraud the system.

An example is a p2p file sharing application where upload is higher valued than download. Here, two peers could collude and claim they shared many files in order to manipulate the measured contribution of these two peers to the system.

Hitchhiking

Hitchhiking is an attack known from reputation systems. Here, a peer is making use of another peer's high reputation with or without this peer's knowledge.

Within the context of the token-based accounting scheme, a peer that makes use of another peer's account status in order to receive accounted resources or services would hitchhike.

Sybil Attack

The Sybil attack was first presented in (Dou02). It is an attack in p2p systems, where a single faulty peer presents multiple identities and by this means attempts to control a substantial fraction of the system, in order to defraud the system.

The Sybil attack is to be considered where trusted data is replicated in a p2p systems. With a successful Sybil attack a peer could control the majority of replicas of the data.

Whitewashing Attack

Whitewashing is an attack known from reputation and incentive systems. Typically, in such systems new peers receive a specific initial rating. A peer that gained over time a rating that is below that initial rating, can “white wash” itself by discarding its identity and taking a new one.

Double Spending

In digital currency systems and similar systems double spending is attack, when a peer spends a single currency unit several times, in order to avoid payment. In the token-based accounting scheme double spending refers to using a token more than once in order to pay for accounted resources or services.

Forgery

Forgery is the process where a peer forges a digital document issued by some entity. Forgery often includes forging a digital signature. In the token-based accounting scheme, forgery refers to forging a signature of the scheme’s private key.
C.1.4 Definitions Related to Commercial Transaction Processes

C.1.4.1 Resources

In (Car08) resources are defined as CPU, Memory, I/O, Peripheral, Network. Within p2p systems often bandwidth, storage capacity, and processing power are named as the relevant resources (Shi01, SW05a).

Resources are used in order to provide services in p2p systems. As in p2p systems services are provided between peers, the resource bandwidth is always required and therefore it is the most important resource.

C.1.4.2 Services And Transactions

In accordance with (Mer08) a transaction is “something transacted; especially: an exchange or transfer of goods, services or funds”. A good can be for example a specific digital content like an mp3-song. A service can be can vary from resource usage (e.g., calculations performed on a remote peer like in SETI@Home (Uni08), the storage of a file for a specific period of time like in OceanStore (UC 08)) to complex services (e.g., a radiologist does a diagnosis of an x-ray).

In p2p systems all transactions involve a data transmission service; either file transfer, a streaming service, or simply a message transfer. Therefore, for simplicity, in this dissertation all transaction objects (goods, services, funds from the above definition) accounted for are from now on summarised under the term services.

C.1.4.3 Price And Tariff

A price is “... an amount of money associated with a unit of service; this is used to compute the charge. The tariff refers to the general structure of prices and charges. An example of a tariff is \( a + pT \), where \( a \) is a price for setting up, \( p \) is a price per second for using the service, and \( T \) is the duration of the connection in seconds. A tariff is that part of the contract between two parties that specifies the way the charge will be computed for the service.” (CW03)

C.1.4.4 Charging

A charge is the amount that is billed for a service (CW03). Accordingly, charging is the process of computing the charge according the tariff agreed by the transaction partners for the service. Charging requires as input accounting information about the service.

C.1.4.5 Billing

In this dissertation billing is the process of creating a document, the bill, stating the amount a service consumer has to pay to the service provider. Billing requires as input from charging the relevant information that must be included in the bill: customer, provider, transaction object, charge, date, and potentially further information.
C.1.4.6 Payment

Payment is “the partial or complete discharge of an obligation by its settlement in the form of the transfer of funds, assets, or services equal to the monetary value of part or all of the debtor’s obligation.” (Web08)

Payment as functionality can be provided by an application or service external to the IT-system used to deliver the primary service of a transaction from provider to customer.

C.2 Details on Security Mechanisms for Distributed Systems

C.2.1 Secret Sharing

This section gives further details to different secret sharing techniques.

C.2.1.1 XOR Secret Sharing

A secret $s$ has to be shared between $n$ shareholders using XOR Operation (see (DW04)). The different shares are constructed and distributed in the initialisation phase and reconstructed in the reconstruction phase.

Initialisation Phase

To distribute the secret a trusted dealer $D$ chooses $n - 1$ random values ($s_1, s_2, \ldots, s_{n-1}$) that are used as shares and the last share is constructed using Equation C.1. Then, these shares are sent secretly to the shareholders ($P_1, \ldots, P_n$) that store the share.

$$s_n = s \oplus s_1 \oplus s_2 \oplus \ldots \oplus s_{n-1}$$ (C.1)

Reconstruction Phase

To reconstruct the secret $s$ all shares of the shareholders have to be combined using the XOR-Operation. Thus, all shares are sent to one shareholder that performs Equation C.2 for reconstruction.

$$s = s_1 \oplus s_2 \oplus \ldots \oplus s_n$$ (C.2)

C.2.1.2 Additive Secret Sharing

Similar to XOR secret sharing, additive secret sharing can be used. Instead of using the XOR operation, an addition is used. Additive secret sharing is, e.g., used and presented in (Rab98) and (JS05).

Initialisation Phase

Equivalent to the XOR secret sharing scheme, a trusted dealer $D$ generates $n - 1$ random numbers ($s_1, s_2, \ldots, s_{n-1}$). The missing last share is computed using Equation C.3. Then, these shares $s_1, s_2, \ldots, s_n$ are distributed secretly to the corresponding shareholders ($P_1, P_2, \ldots, P_n$).

$$s_n = s - \sum_{i=1}^{n-1} s_i$$ (C.3)
Reconstruction Phase

The common approach to reconstruct the secret $s$ is to send all shares to a shareholder that reconstructs the secret using Equation C.4.

\[ s = \sum_{i=1}^{n} s_i \]  

(C.4)

A disadvantage of the approach presented above, is that the secret is revealed. This reduces the security of the secret sharing scheme, since an attacker has the possibility to obtain the secret. However, if the secret $s$ is used for cryptographic tasks using RSA, $s$ does not need to be revealed.

The functions to decrypt, encrypt and sign messages using RSA are presented below. Equation C.5 is used to encrypt a message $m$ and Equation C.6 is used to decrypt the encrypted message or to sign a message. In this context, $d$ and $e$ are the private and the public key. Thus it is very probably that the secret $s$ will be equal to the private key $d$.

\[ c = m^e \mod N \]  

(C.5)

\[ c^d \mod N = m^{ed} \mod N = m \]  

(C.6)

For information on RSA: to construct $d, e$ one has to choose two primes $(p, q)$ and compute the product $N = pq$ of them. Then the public key $e$ is chosen using Equation C.7 and the private key $d$ using Equation C.8. For more information about RSA see e.g. (Buc03).

\[ 1 < e < \varphi(N) = (p-1)(q-1) \cap \gcd(e, (p-1)(q-1)) = 1 \]  

(C.7)

\[ 1 < d < \varphi(N) = (p-1)(q-1) \cap de = 1 \mod (p-1)(q-1) \]  

(C.8)

If the private key $d$ is shared between $n$ shareholders and a message $m$ has to be signed, the shareholders have to perform the following tasks:

Each shareholder $P_i$ $(i \in \{1, \ldots, n\})$ computes a partial signature $\text{sig}_{s_i}(m)$ (see Equation C.9) and sends this partial signature to the requester of the signature. Then the requester has to multiply each partial signature to get the final signature $\text{sig}_{s}(m)$ (Equation C.10).

\[ \text{sig}_{s_i}(m) = m^{s_i} \mod N \]  

(C.9)

\[ \text{sig}_{s}(m) = \prod_{i=1}^{n} \text{sig}_{s_i}(m) \mod N = \prod_{i=1}^{n} m^{s_i} \mod N = m^{\sum_{i=1}^{n} s_i} \mod N = m^{s} \mod N \]  

(C.10)

This is possible, because of the mathematical rules for exponentiation (Equation C.11 and C.12).

\[ a^b a^c = a^{b+c} \]  

(C.11)

\[ (a^b)^c = a^{bc} \]  

(C.12)

It is important to notice that it is mandatory to use integers as values for the shares $(s_1, \ldots, s_n)$. If non natural numbers are used it is not possible to compute the partial signature (because one has to compute the root modulo $N$). Additionally, it will be good if all shares are positive integer. Using negative integers, one has to compute the inverse modulo of the message $m \mod N$. This is possible as long as $m$ is relative prime to $N$. Note that, if one finds a message $m$ that is not relative prime to $N$, $N = pq$ is factorised and the RSA scheme is broken (or can be broken very easily).
C.2.1.3 Blakely’s Secret Sharing

Blakely developed another historical secret sharing scheme presented in (Bla79). It is not often mentioned in literature and thus, only the principle and not the details are presented here.

Blakely uses the intersection of hyperplanes to represent the secret. Therefore each share is a hyperplane in a m-dimensional space. To reconstruct the secret, at least m shares are needed and the intersection of these planes is used. Figure C.1 shows the secret sharing scheme within a 3-dimensional space. The secret can either be expressed as a coordinate of the axes or as the absolute value of the vector from the point of origin to the point of intersection (see (RSA04)).

One disadvantage of Blakely’s scheme is, that the shareholder knows, that the secret is on its hyperplane. This can help to find the secret faster in comparison to Shamir’s scheme, where the shareholder has no information about the secret. Additionally, one has to mention, that the secret has to be completely reconstructed, e.g., to decrypt, sign and encrypt messages. Shamir’s scheme offers, e.g., the possibility to create partial signatures that can be combined to create the final signature (see Section 2.4.7.1).

C.2.2 Verifiable Secret Sharing

This section presents some well-known secret sharing schemes and evaluates them, including Pedersen’s Verifiable Secret Sharing, which is presented in Chapter 2.4.5.1.

C.2.2.1 Feldman’s Verifiable Secret Sharing Scheme

Feldman developed a popular verifiable secret sharing scheme. It is based on Shamir’s scheme (polynomial interpolation) and presented in (Fel87). It increases the security by allowing the shareholders to check the correctness of the shares provided by the dealer. Therefore, the dealer broadcasts some verification values. These values do not reveal any information about the secret, because they are based on hardness of computing discrete logarithms. The detailed steps for verification in initialisation and reconstruction phase are presented below.

Initialisation Phase

The initial steps of Feldman’s scheme are similar to the Shamir’s scheme. The dealer chooses two prime numbers p and q, such that \( p = mq + 1 \) (m is a small integer) and creates a random polynomial function \( f(x) \mod q \) of degree \( k - 1 \) (see Equation 2.1). Then the dealer transmits the shares \( (s_1 \ldots s_n) \) constructed with Equation 2.2 to the shareholders.

Additionally the dealer computes the public verification values \( (y_i = g^{\alpha_i}) \) of the polynomial function. Whereas \( \alpha_i \ (i = \{0, \ldots , k - 1\}) \) are the coefficients of the polynomial from Equation 2.1 and \( g \in \mathbb{Z}_p^* \).
of order \( q \). These public values \( (g^s, g^{a_1}, \ldots, g^{a_{k-1}}) \) are then broadcasted to all shareholders, which can now check the correctness of the received shares by using Equation C.13.

\[
g^{s_i} \equiv (g^{s})(g^{a_1})^i (g^{a_2})^{i^2} \cdots (g^{a_{k-1}})^{i^{k-1}} \mod p \\
\equiv g^{s+a_1i+a_2i^2+\cdots+a_{k-1}i^{k-1}} \mod p \tag{C.13}
\]

Whereas \( i \) is the value of the variable used to compute the share \( s_i \) for shareholder \( P_i \) \((s_i = f(i))\).

If Equation C.13 holds, each shareholder broadcasts a message that it accepts the received share (and trusts the dealer). If all shareholders accept its share, the shares are distributed successfully and the initialisation phase ends. But, if a shareholder identifies an incorrect share, it publishes an accusation against the dealer. Then the other shareholders have to decide whether to trust the shareholder or the dealer. Therefore, the dealer publishes additional information, that each shareholder can verify the accused share.

Reconstruction Phase

The verification methods can also be applied in reconstruction phase. Therefore, each shareholder has to store the public verification values broadcasted by the dealer in initialisation phase. Now he can check the correctness of the shares, received to reconstruct the secret \( s \). If one of the received shares is incorrect, an accusation is broadcasted against the corresponding shareholder and another share is requested from a different shareholder. The disadvantage of the verification methods is that it cannot be applied for partial signatures (e.g. \( ms_i \mod N \)).

C.2.2.2 Verifiable Secret Sharing with A Symmetric Polynomial

Beside Shamir’s secret sharing scheme with one variable, secret sharing schemes with two variables exist. In this section a secret sharing scheme based on a symmetric polynomial is presented. It is unconditional secure, this means that each shareholder checks the correctness of its shares with each other shareholder in the secret sharing scheme. It is presented among others in (SW99).

Initialisation Phase

For the verifiable secret sharing scheme with a random symmetric polynomial function, the dealer \( D \) constructs a function following Equation C.14 with \( a_{00} = s \) and \( a_{ij} = a_{ji} \).

\[
f(x, y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} a_{ij} x^i y^j \tag{C.14}
\]

Each shareholder \( P_i \) \((i = \{1, \ldots, n\})\) receives its share (a function) \( s_i(x) = f(x, \omega^i) \) from the dealer, whereas \( \omega \) is an element of the finite field generated with the prime number \( q \). And \( \omega \) is known by all other shareholders participating the verifiable secret sharing scheme. To verify the received share, each \( P_m \) sends \( s_i(\omega^j) = f(\omega^j, \omega^i) \) to shareholder \( P_j \) \( j \in \{1..n \mid j \neq i\} \). Then shareholder \( P_j \) performs the verification using Equation C.15.

\[
s_i(\omega^j) = s_j(\omega^i) \Leftrightarrow f(\omega^j, \omega^i) = f(\omega^i, \omega^j) \tag{C.15}
\]

Equation C.15 holds, since \( f(x, y) \) is a symmetric polynomial function. If the check fails, the corresponding shareholder \( P_j \) broadcasts \((i, j)\) the index of the shareholders for that the equation does not hold. At the end of the initialisation phase each shareholder has the same set of valid shareholders. If the number of valid shareholders is greater than a certain threshold (e.g. \( n-k \)) the initialisation phase
was successful. Otherwise the dealer is malicious, since per definition only $k-1$ shareholder may be malicious.

Another approach to verify the shares distributed by the dealer is presented in (NN04). If shareholder $P_j$ decides that Equation C.15 does not hold, an accusation is broadcasted. If more than $k-1$ accusations were broadcasted, the dealer is malicious. Otherwise the share $s_i(x)$ of shareholder $P_i$ has to be revealed and all participating shareholders can check whether the dealer $D$, shareholder $P_i$ or shareholder $P_j$ is malicious.

**Reconstruction Phase**

To reconstruct the secret Lagrange interpolation is used (Equation 2.3). Therefore, at least $k$ shareholder $P_i$, $i \in \{1, \ldots, n\}$ have to send their share $s_i(x)$, e.g., to a shareholder that checks the correctness of the shares and reconstructs the secret $s$ using Equation 2.3 and $(i, s_i(0))$ as input values.

The shareholder (e.g. $P_j$) that reconstructs the secret, checks if $s_j(\omega^i) = s_i(\omega^j)$ for the shares $s_i(x)$ received from shareholder $P_i$, $i \in \{1, \ldots, n\}$. If the equation holds, the secret can be computed. Otherwise, the wrong share is sorted out and the shareholder is accused. Since $k-1$ shareholder may be malicious it will be mandatory to have at least $2k-1$ shares for verification. So it will be guaranteed that at least $k$ correct shares are available.

Additionally, it is possible to generate partial signatures that can be used for cryptographic tasks (using RSA). Therefore, each shareholder creates the partial signature $\text{sig}_{s_i}(m)$ using Equation C.16. The verification of these messages is not possible.

$$\text{sig}_{s_i}(m) = m^{s_i(0)} \mod n \quad \text{(C.16)}$$

**C.2.2.3 Verifiable Secret Sharing with An Asymmetric Polynomial**

Instead of a symmetric polynomial it is possible to use an asymmetric one. This involves a bit more overhead, like presented in the following section. The scheme is presented among others in (NN04).

**Initialisation Phase**

A dealer $D$ creates an asymmetric polynomial $f(x,y)$ using Equation C.17 with $a_{ij} \neq a_{ji}$ and $a_{00} = s$. Then $D$ computes the shares $s_i(x) = f(x, \omega^i)$ and $t_i(y) = f(\omega^i, y)$ for each shareholder $P_i \ (i \in \{1, \ldots, n\})$. Whereas, similar to Section C.2.2.2, $\omega$ is an element of the finite field generated with a prime number $q$.

$$f(x, y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} a_{ij} x^i y^j \quad \text{(C.17)}$$

To verify the received shares, shareholder $P_i \ (i \in \{1, \ldots, n\})$ first checks if $s_i(\omega^i) = t_i(\omega^i)$. If this equation does not hold, $P_i$ broadcasts an accusation against the dealer $D$. Otherwise $P_i$ computes for each shareholder $P_j \ (\{j \in \{1, \ldots, n\} | j \neq i\})$ the verification values $s_j(\omega^i)$ and $t_i(\omega^j)$ and distributes them secretly. Upon receive of these verification values $P_j$ compares its shares with the all received values $(s_i, t_i)$ using Equation C.18.

$$s_i(\omega^j) = t_j(\omega^i) \land t_i(\omega^j) = s_j(\omega^i) \quad \text{(C.18)}$$

If this equation does not hold, shareholder $P_j$ broadcasts $(i, j)$ and the accusation can be resolved using the methods presented in Section C.2.2.2.
Reconstruction Phase

The reconstruction phase of the secret is equal to the reconstruction of a secret sharing scheme with symmetric polynomial. At least \( k \) shareholder \( P_i, i \in \{1, \ldots, n\} \) send their shares \( s_i(x), t_i(x) \) to a shareholder (e.g. \( P_J \)) that reconstructs the secret \( s \) using Equation 2.3 (either with \((i, s_i(0))\) or \((i, t_i(0))\) as input values). Before reconstruction, the received shares can be verified, by checking \( s_i(\omega_j) = t_j(\omega_i) \land t_i(\omega_j) = s_j(\omega_i) \). If the checks are correct the values can be used to reconstruct the secret. Otherwise an accusation is broadcasted and another share has to be used.

Again it is mandatory to use at least \( 2k-1 \) shares for reconstruction, because at most \( k \) shares may be malicious. And it is also possible to generate partial signatures using \( s_i(0) \) or \( t_i(0) \).

C.2.2.4 Summary

Verifiable secret sharing is an important feature for applying a secret sharing scheme in autonomous distributed systems.

The schemes of Feldman et al. and Pedersen are based on the problem to solve the discrete logarithm. In difference to Feldman et al., Pedersen scheme reveals no Shannon information (see (Sha01) for further information). This means, if the value \( s \) is verified using Feldman, the corresponding verification value \( g^s \) is always the same. Pedersen randomises the verification value by choosing a random parameter \( t \). So the verification value of \( s \) is different \( (g^sh^t_1 \neq g^sh^t_2) \). This avoids that an attacker that knows a share \( s_i \) and the corresponding verification value does not know when the same share is reused by a fluke.

Stinson and Wei use polynomials with two variables and thus do not have to generate and broadcast extra verification values. That is the advantage of this approach. Each share enables the verification of all other shares in the scheme. However, if a high threshold is assumed (e.g. a \((20,100)\)-threshold scheme), this approach results in very large shares.

C.2.2.5 Frankel’s RSA-based Threshold Cryptography Scheme

The scheme of (FGMY97a) can be applied for both, polynomial secret sharing (Section 2.4.4.1) as well as additive secret sharing (Section C.2.1.2). The principle is to avoid non-natural numbers of the Lagrange coefficients by extending the exponent. In the following the focus is on polynomial secret sharing. The additive extension is not presented in detail.

Initialisation Phase

The dealer \( D \) chooses two primes \( p, q \) and generates the public key \( e \) and private key \( d \). The private key \( d \) is shared among the shareholders. In the following is \( L = n! \), whereas \( n \) is number of shareholders. Additionally, it is assumed that the shareholders are indexed from \( 1, \ldots, n \). The dealer performs the following mathematical operations to generate a proper polynomial and to share the private key \( d \).

- \( D \) computes the greatest common divisor \( H = \gcd(e, L^2) \)
- Two natural numbers \( P \) and \( s' \) are computed such that \( 1 = eP + \frac{L^2}{P} s' \) (using extended Euclidean algorithm)
- \( k' \) is computed such that \( d \equiv P + L^2k' \mod (p-1)(q-1) \) and \( k' \equiv ds'H^{-1} \mod (p-1)(q-1) \). Whereas \( H^{-1} \) is the inverse of \( H \) modulo \((p-1)(q-1)\).

Then the dealer chooses similar to Shamir (Section 2.4.4.1) a random polynomial \( f(x) = \sum_{i=0}^{k-1} f_i x^i \) with \( f(0) = L^2k' \) and \( f_i \in \{0, L, \ldots, 2L^3n^2+1\} \) (\( \epsilon \) is an optional security factor). Then the shares \( s_i = \)
\( f(i) \ (i = \{1, \ldots, n\}) \) are computed and secretly distributed to the corresponding shareholder \( P_i \). The parameter \( P \) is either distributed to each shareholder or published. Additionally to the shares, \( D \) chooses and distributes two generators \( g, g_1 \) for verification and publishes \( g^{s_i L^2} \mod n \) for all \( i \in \{1, \ldots, n\} \).

**Reconstruction Phase**

For reconstruction Equation C.19 is used. A predefined group \( \mathcal{A} \) of \( k \) shareholders compute the partial keys \( k_i \) (Equation 2.6). The private key \( d \) is reconstructed using Equation C.19.

\[
d = P + \sum_{i \in \mathcal{A}} k_i \quad \text{(C.19)}
\]

The advantage of this scheme is that each share is a multiple of \( L = n! \). Therefore, the sum of Equation C.19 is done over integers. To generate a partial signature \( \text{sig}(m) = m^d \mod N \) each shareholder in \( \mathcal{A} \) computes \( \text{sig}_i(m) = m^{k_i} \mod N \) and sends this partial signature to the signature requester. Then the requester combines the received signatures using Equation C.20.

\[
m^d = m^P \prod_{i \in \mathcal{A}} m^{k_i} \quad \text{(C.20)}
\]

The parameter \( P \) can either be added by a defined shareholder or send to the requester that can compute the signature by himself.

Additionally it is possible that more then \( k \) shareholders send a partial signed message \( m^{s_i} \) to the requester. Then the requester has to pick \( k \) partial signatures, generates \( m^{k_i} \) and performs Equation C.21. So, the shareholders have not to communicate to create a signature.

\[
m^d = m^P \prod_{i \in \mathcal{A}} (m^{s_i})^{\sum_{j \in \mathcal{A} \setminus i} \frac{\mathcal{A}}{\mathcal{A} - j}} \quad \text{(C.21)}
\]

**Summary**

Although this scheme circumvents the problem of negative numbers or non-natural numbers in the exponent, it has a limitation that prohibits its application large distributed anonymous systems. In order to avoid non natural numbers in the Lagrange coefficients, one has to compute the factorial of the number of shareholders. This will be applicable for threshold groups with around up to 100 shareholders, but not for threshold groups with thousands of shareholders. Accordingly, this scheme can be applied in server-based environments like COCA (ZSvR02). However, it is not suited for large p2p systems.
D Supplementary Details to Chapter 4

D.1 Account Holder Set

D.1.1 Peer Assignment, Information Storage and Retrieval

D.1.1.1 Aggregation Account Creation

Checking for Account Existence

Algorithm 4.2  Check for Aggregation Account Existance

This algorithm describes the procedure how a peer $P_i$ can determine, if the aggregation account for a peer $P_C$ exists.

1. Peer $P_i$ sends a $check\_aggregation\_account\_req(queryID, AA_C^{ID}, hc, peerID(P_i))$ to the peer responsible for $AA_C^{ID}$, $P_C$, with the hop counter $hc$ set to $-1$.

2. Whenever a peer $P$ receives a $check\_aggregation\_account\_req(queryID, AA_C^{ID}, hc, peerID(P_i))$ message,
   1. $P$ checks if it hosts the aggregation account for $AA_C^{ID}$. If true, $P$ creates an $aggregation\_account\_status(queryID, AA_C^{ID})$ message and sends it to $peerID(P_i)$.
   2. Else, $P$ creates a $check\_aggregation\_account(queryID, AA_C^{ID}, hc, peerID(P_i))$ message and forwards it to its successor and to its predecessor on the DHT.

3. Whenever a peer $P$ receives a $check\_aggregation\_account(queryID, AA_C^{ID}, hc, peerID(P_i))$ message,
   1. $P$ checks if it hosts the aggregation account for $AA_C^{ID}$. If true, $P$ creates an $aggregation\_account\_status(queryID, AA_C^{ID})$ message and sends it to $peerID(P_i)$.
   2. Else, $P$ increases the hopcounter $hc$ by one.
   3. If $hc < X_{max}$, $P$ forwards the message to its successor $succ(P)$ to its predecessor $pred(P)$ on the DHT, $succ(P)$, depending on the direction $P$ received the message from.
   4. Else $P$ creates an $aggregation\_account\_failure(queryID, AA_C^{ID})$ message and sends it to $P_i$.

Aggregation Account Creation

Algorithm 4.3  Aggregation Account Creation

This algorithm is used to create a new aggregation account for a peer $P_{new}$.

1. A new peer $P_{new}$ contacts trusted peer $TP_1$ in order to request new tokens (token aggregation or start-up number of tokens).

2. $TP_1$ checks if aggregation account for $P_{new}$ exists using Algorithm 4.2.
3. If aggregation account for $P_{new}$ does not exist,
   1. $TP_1$ calculates the aggregation account ID $AA_{new}^{ID}$ for $P_{new}$.
   2. $TP_1$ looks up the peer responsible for $AA_{new}^{ID}$, denoted $P_C$.
   3. $TP_1$ determines $x$, the number of hops $AA_{new}$ should be shifted on the DHT.
   4. $TP_1$ randomly picks $r$ values $y_i$, where $1 + 2r + \sum_{i=1}^{r} y_i = x$.
   5. Set $P_k = P_C$.
   6. For $i = 1$ to $r$ do:
      1. Set $P_i = P_k$.
      2. $TP_1$ looks up the $P_i$’s successor on the DHT, $P_j$.
      3. $TP_1$ sends request_peer message containing $(y_i - 1)$ to $P_j$.
      4. $P_j$ forwards the request_peer message to its $(y_i - 1)$-th successor, $P_m$.
      5. $P_m$ responds to $TP_1$ with its successor’s peerID, $P_k$.
   7. $P_k$ is the peer with $x-1$ hops distance to $P_C$. $TP_1$ sends an init_aggregation_account($AA_{new}^{ID}$) message to $P_k$’s successor on the DHT, $P_{AH1}$.
   8. $P_{AH1}$ initiates the aggregation account $AA_{new}$ using Algorithm 4.4. When done, $P_{AH1}$ sends an aggregation_account_confirmed message to $TP_1$.

4. Upon receiving the aggregation_account_confirmed message, $TP_1$ creates the $l$ new token and stores their token IDs in $AA_{new}$ by sending an update_account($t_{new}^1$, ..., $t_{new}^l$) message to $P_{new}$’s account holder set $AHS_{new}$ using Algorithm 4.1.

---

**Account Initialisation**

**Algorithm 4.4  Aggregation Account Initialization**

This algorithm is used as sub-part by Algorithm 4.3 in order to initialise the aggregation account at all peers of the account holder set.

1. $P_{AH1}$ receives an init_aggregation_account($AA_{new}^{ID}$) from a trusted peer $TP$.
2. $P_{AH1}$ creates a create_aggregation_account message($AA_{new}^{ID}$), adds its peerID to the message, sets its hop counter to 1, and forwards the message to its successor on the DHT, $succ(P_{AH1})$.
3. $P_{AH1}$ remembers the hop counter as its own position within the account holder set.
4. Whenever a peer $P_s$ receives a create_aggregation_account message,
   1. $P_s$ checks if it received this message before. If so it sets $AS = h$ and proceeds to step 4.e.i.
   2. It instantiates a new aggregation account for $AA_{new}^{ID}$.
   3. It adds its peerID to the message and increases the message’s hop counter $h$ by one.
   4. It remembers the hop counter as its own position within the account holder set.
   5. Let $AS$ be the preferred account holder set size in the p2p system.
      1. If $h = AS$ then
         - $P_s$ creates an aggregation_account_complete message containing the list of peerIDs from the received create_aggregation_account message.
• It remembers the list of peerIDs in the aggregation account.
• It sends this message to the peerID preceding its own peerID in the list of peerIDs.

2. else, $P_s$ forwards the create_aggregation_account to its successor on the DHT, $\text{succ}(P_s)$.

5. Whenever a peer $P_p$ receives an aggregation_account_complete message it
   • remembers the list of peerIDs contained in the message in the aggregation account.
   • it send the aggregation_account_complete message to the peerID preceding its own peerID in
     the list of peerIDs.

6. When $P_{AH1}$ receives the aggregation_account_complete message it remembers the list of peerIDs
   contained in the message in the aggregation account. Then, it sends an account_confirmed message
to $TP$.

D.1.1.2 Detection of Replica Number

In order to check if the account holder set is still complete, Algorithm 4.5 is applied. The algorithm can
be initiated by any account holder. In order to keep track of an aggregation account the algorithm needs
to be executed periodically. Thus, any account holder that did not receive a detect_AHSS-message for a
specific time span will initiate Algorithm 4.5. The algorithm uses reliable authenticated connections, as
all account holders must know each other and anonymity is not desired within an account holder set.

As explained in Algorithm 4.4, each account holder holds a list of all current account holders in the
set. The first step of $P_{AHi}$, the initialising peer of Algorithm 4.5, is to announce its know last status of
the account holder set to all set members. In order to do that peer $P_{AHi}$ send a detect_AHSS-message
containing the relevant status information (table with account holders and their last time active) to all
known account holders (step 1 and step 2). There are three options for the result to these messages (step
3). If the account holder is offline the message is lost. If the receiver is online but not an account holder
any more, it deletes itself from the account holder table contained in the message. Further, it checks
if it has more recent information that can be added to the table. However this is unlikely, as the peer
should have received less current information about the account holder set than the initialiser. In the
third option the message receiver is still an account holder. It will add any further up-to-date information
to the account holder table.

The updated account holder table is sent back to $P_{AHi}$ using a detected_AHSS-message. The message
contains a time stamp when it was sent. Further, each peer having an additional account holder in the
table will forward the detect_AHSS-message to it. These account holders will also respond to $P_{AHi}$, with
a detected_AHSS-message.

$P_{AHi}$ collects the detected_AHSS-messages (step 4). As reliable authenticated connections are used
for communication within the account holder set, it is assumed that an account holder is offline if it
does not respond to the detect_AHSS-message. $P_{AHi}$, extracts the account holder tables from the received
detected_AHSS-messages and creates an updated account holder table (step 5). In order to do that $P_{AHi}$,
has first to calculate potential differences between its local clock and the responsedees’ local clocks, as
these influence the time stamps in the account holder tables. The differences are calculated using the
message time stamps created in step 3. (This is used for potential large differences in the account holders’
clocks. Changes in the account holder table happen due to churn. Therefore, small clock variations can
be tolerated). $P_{AHi}$, modifies the received account holder tables by adding the difference in the local time
to it. Then, $P_{AHi}$, modifies its own account holder table by adding any new account holders received and
updating the “last time active”-entries according to the received responses.

$P_{AHi}$, now sends the updated account holder table to all actual account holders using confirm_AHS-
messages (step 6). If the account holder table shows that the account holder set currently has less
members than the preferred account holder set size, $P_{AH}$, also initialises the execution of Algorithm 4.6 with a subsequent 4.8. This will lead to an optimal account holder size and the correct position of it on the DHT.

**Algorithm 4.5 Detection of Current Account Holder Set Size**

This algorithm is used by any account holder in order to detect the current size of the account holder set it belongs to.

1. An account holder $P_{AH}$ creates a detect_AHSS($AA^{ID}$) message. It adds a table to the message that contains the account holders $P_{AH}$ believes are responsible for the aggregation account $AA^{ID}$ and the time the account holders have been active the last time.

2. $P_{AH}$ sends the detect_AHSS($AA^{ID}$) to all account holders in the table (apart from itself).

3. Each peer $P_j$ receiving a detect_AHSS($AA^{ID}$) message checks weather it is account holder for aggregation account $AA^{ID}$.
   - If $P_j$ is account holder for this aggregation account, then,
     - $P_j$ extracts the account holder table for aggregation account $AA^{ID}$ from the detect_AHSS($AA^{ID}$) message.
     - $P_j$ adds all peers missing in the account holder table to it. It remembers the peers it added to the table.
     - $P_j$ marks all account holders in the table that are not account holders for this account any more. The mark is the time when the peer left the account holder set.
     - $P_j$ forwards the updated detect_AHSS($AA^{ID}$) message to the peers it added to the account holder table.
     - $P_j$ responds to $P_{AH}$ with an detected_AHSS($AA^{ID}$) message, containing the updated account holder table and a time stamp.
   - Else,
     - $P_j$ extracts the account holder table for aggregation account $AA^{ID}$ from the detect_AHSS($AA^{ID}$) message.
     - $P_j$ removes itself from the account holder table.
     - $P_j$ responds to $P_{AH}$ with an detected_AHSS($AA^{ID}$) message, containing the updated account holder table and a time stamp.

4. $P_{AH}$ collects all detected_AHSS($AA^{ID}$) messages.

5. $P_{AH}$ extracts all account holders from the received detected_AHSS($AA^{ID}$) messages and creates an updated account holder table:
   1. for each received detected_AHSS($AA^{ID}$) messages send by $P_s$ do
      1. $\Delta t_s = t_r - t_s$, where $t_r$ is the local time when the message was received, $t_s$ is the local time when the message was sent by $P_s$.
   2. for each entry $e$ in the received account holder table do
      A. $e.t = e.t + \Delta t_s$, where $e.t$ is the last-time-active of the entry.
   2. $P_{AH}$ creates a new account holder table from the received responses.

6. $P_{AH}$ creates a confirm_AHS($AA^{ID}$) message containing the updated account holder table and sends it to all peers in the table.
7. If the account holder table contains less entries than the preferred account holder set size, then
   • $P_{AH_i}$ executes Algorithm 4.6 with a subsequent Algorithm 4.8.

### D.1.1.3 Detection of Correct Account Holder Set Position

The algorithm determines if the account holder set should be moved without compromising the account holder set's position on the DHT. As described in Section 4.2.1.5 each account holder knows its position in an account holder set. The detection of the correct account holder position must be led by the first account holder in the set. Therefore, any other account holder $P_{AH_i}$ initialising the algorithm must request the check of the account holder position from the first peer using a check_account_position_request-message (step 1). When the first account holder receives the message it replies with a checking_account_position-message, in order to inform the initialiser that the algorithm is executed (step 2). If the first account holder cannot be reached, $P_{AH_i}$ tries to send the check_account_position_request-message to the other account holders it knows. This way, the first account holder will be found (step 3).

The next steps of the algorithm serve the purpose of counting the hops on the DHT between the first account holder and the peer responsible for the aggregation account ID $AA^{ID}$, $P_{AA^{ID}}$. However, it must be concealed which peer is collecting this information, as this would reveal the aggregation account's position on the DHT. That is, the message must be sender anonymous. In order to do that the first account holder $P_{AH_1}$ creates a request_hop_count-message. This message contains a hopcounter that initially takes a random value. Due to this random value a peer receiving the request_hop_count-message cannot determine how many hops the message has been forwarded already on the DHT. Further, the message contains a unique query ID, in order to identify it when the message reaches the initial sender again. $P_{AH_1}$ will send the request_hop_count-message to $P_{AA^{ID}}$. In order to conceal the sender it will use a UDP-based message. This peer will increase the hopcounter by one and forward the message to its successor on the DHT. The successor will do the same. This is repeated until the message reaches again $P_{AH_1}$. Using the message's query ID $P_{AH_1}$ identifies the message as its own. By subtracting the hop counter's current value from its initial value $P_{AH_1}$ can determine how many hops it is away from the peer responsible in the DHT for the aggregation account ID. If this value differs from the account shift value $x$, then $P_{AH_1}$ initialises Algorithm 4.8 for moving the aggregation account.

There is the possibility that a peer is not increasing the hop counter by one but is modifying it in a different way. Then there are two options. Either $P_{AH_1}$ recognises the deviation of protocol because the result does not make sense. The other option is that the result deviates just by a few hops from the actual value, which cannot be recognised. This is without effects as the account moving algorithm, Algorithm 4.8, is independent of this result.

A potential problem of sender anonymous messages is Denial of Service attacks. In order to avoid that identified messages can be used, where the first message is not sent directly to $P_{AA^{ID}}$, but indirectly over a trusted peer. $P_{AA^{ID}}$ will not be able to determine the sender because the trusted peer appears as sender in the message it receives.

#### Algorithm 4.6 Detection of correct account holder set position

This algorithm is used by any account holder peer in order to detect if the first account holder is still located at $x$ hops distance from the peer that is responsible for the aggregation account ID in the DHT.

• Let $P_{AH_i}$ be an account holder for aggregation account $AA^{ID}$, where $i$ represents the account holders position in the account holder set. $P_{AH_i}$ sets variable $j = 1$. 

Appendix D. Supplementary Details to Chapter 4 259
1. If $P_{AH_i}$ is not the first account holder in the account holder set ($i \neq 1$), it creates a check_account_position_request($AA^{ID}$) message and will send it to the first account holder in the account holder set $P_{AH_i}$.

2. Any peer receiving a check_account_position_request($AA^{ID}$) message will
   1. forward the message to the first account holder in the account holder set, if it is not the first account holder in the set.
   2. will create as response to $P_{AH_i}$ a checking_account_position($AA^{ID}$) message $P_{AH_i}$, if it is the first account holder in the account holder set $P_{AH_i}$.

3. If $P_{AH_i}$ does not receive a checking_account_position($AA^{ID}$) message within a specific time period,
   1. it will assume that this peer went offline.
   2. it set variable $j = j + 1$.
   3. If $j < i$ it will send the check_account_position_request($AA^{ID}$) message to account holder $P_{AH_j}$.
   4. Else it will continue will consider itself to be the first account holder of the account holder set and continue with step 4.

4. $P_{AH_1}$ will
   1. create a request_hop_count message containing a hop counter set to a random value and a query ID.
   2. $P_{AH_1}$ will remember the query ID with the corresponding hop counter's value.
   3. $P_{AH_1}$ will send the message to $P_{AA^{ID}}$ that is responsible for $AA^{ID}$ in the DHT.

5. Any peer $P_a$ receiving a request_hop_count message will increase the message's hop counter by one.

6. $P_a$ checks, if it sent the request_hop_count message using the contained query ID.
   - If it did not sent the message, $P_a$ will increase the message's hop counter by one and forward it to its successor in the DHT.
   - If it sent the message, it will subtract the initial hop counter value from the received hop counter value. The result $HC$ is the number hops the peer is away from $P_{AA^{ID}}$.

7. If $HC \neq x$ then $P_{AH_1}$ initialises Algorithm 4.8 for moving the aggregation account.

---

**D.1.1.4 Account Holder Set Locking**

Before an account can actually be moved, it has to be ensured that the account holder set is not processing any $aa_write_requests$ in parallel. Otherwise, inconsistencies within the account holder set cannot be avoided. An $aa_write_request$ is used to add new accounting information to an aggregation account (see Algorithm 4.11). The challenge to solve is that even in the presence of competing lock requests from different peers an unanimous result is achieved. In order to achieve this, a variation of the Strawman Protocol presented in (LLZ04) is applied.

First, the peer requesting a lock $P_{AH_X}$ creates a request_account_lock-message, which it forwards to all account holders of aggregation account $AA^{ID}$ (step 1). The account holders receiving the message have
to check, if the lock is possible or not. A lock is not possible, if it is already locked by another account
holder or if another account holder has requested a lock earlier, however the lock is not active yet (step
2). If a lock is possible the account holder responds to all other account holders with an account_locked
message. To this message it attaches a list of the open write requests ($L_{WR}$). This is required, as the
account holders might receive write request in a different order and it is important that a locked account
has a consistent view on the processed write requests.

Each account holder (including $P_{AH_x}$) receiving account_locked($AA^{ID}$, locked, $L_{WR}$)-messages from
more than half of the account holders before this process times out, considers the account as locked
for $P_{AH_x}$ (step 3). It executes the write requests necessary for locking a consistent account holder set.
When finished they inform $P_{AH_x}$ that the lock is active by sending an account_lock_confirmed-message
to $P_{AH_x}$ containing the list of processed and still open write requests $L'_{WR_y}$ (step 4). $P_{AH_x}$ combines
the received lists $L'_{WR_y}$ and sends it to all account holders that have still an inconsistent view on the list.
These peers will again answer with a account_lock_confirmed-message.

When all account holders finally have a consistent view on which write request should have been
processed and which not, $P_{AH_x}$ finally will execute the operations for which the aggregation account
was locked, e.g., an account movement (step 5).

If $P_{AH_x}$ does not receive the required number of account_lock_confirmed-messages, it cannot acquire
the lock for the aggregation account and is sending an account_lock_cancelled-message in order to inform
the account holders that the lock request was not successful and is not maintained. If required, it will
try a random time later again to request a lock (step 6). A lock ends if the peer owning the lock sends
an account_lock_release-message.

**Algorithm 4.7 Account Holder Set Locking**

This algorithm is used by any account holder peer in order to lock an account for writing new data in it while
account management operations take place. Account holder $P_{AH_x}$ wants to request a lock on the account holder
set for aggregation account $AA^{ID}$.

1. $P_{AH_x}$ creates a request_account_lock($AA^{ID}$) message. It forwards the
request_account_lock($AA^{ID}$) message to all account holders for account $AA^{ID}$.

2. Whenever an account holder $P_{AH_y}$ receives a request_account_lock($AA^{ID}$) message,
   1. $P_{AH_y}$ checks if the account $AA^{ID}$ is already locked. If so, $P_{AH_y}$ responds to $P_{AH_x}$
      with an account_locked($AA^{ID}$, already_locked) message. Break.
   2. $P_{AH_y}$ checks if it responded already to another unfinished lock request. If so, $P_{AH_y}$ responds
to $P_{AH_x}$ with an account_locked($AA^{ID}$, lock_already_requested) message. Break.
   3. $P_{AH_y}$ creates a list of outstanding aa_write_requests, denoted $L_{WR}$. It responds to all account
      holders with an account_locked($AA^{ID}$, locked, $L_{WR}$) message.

3. Each account holder $P_{AH_x}$ collects the account_locked messages. If it receives more than $AHSS/2$
   account_locked($AA^{ID}$, locked, $L_{WR}$) messages,
   1. $P_{AH_x}$ executes all necessary aa_write_requests in order to achieve an consistent account
      holder set status. $P_{AH_x}$ marks the processed aa_write_requests in $L_{WR}$. The resulting list
      is denoted $L'_{WR}$.
   2. $P_{AH_x}$ sends an account_lock_confirmed($AA^{ID}$, $P_{AH_x}$, $L'_{WR_y}$) to $P_{AH_x}$.

4. If $P_{AH_x}$ received more than $AHSS/2$ account_lock_confirmed($AA^{ID}$, $P_{AH_x}$, $L'_{WR_y}$) messages
   before the process times out,
   1. it checks of the received $L'_{WR_y}$ are consistent. If not,
      1. it chooses $L''_{WR} = \bigcup L'_{WR_y}$
2. it sends a account_locked($AA^{ID}$, locked, $L''_{WR}$) message to the inconsistent $P_{AH_y}$.

5. $P_{AH_x}$ executes all necessary operations that required the lock.

6. If $P_{AH_x}$ received less than $AHSS/2$ account_lock_confirmed($AA^{ID}$, $P_{AH_x}$, $L''_{WR}$) messages, $P_{AH_x}$ sends a account_lock_cancelled($AA^{ID}$) message to all account holders and waits for a random time period and restarts this algorithm.

7. In order to release a lock, $P_{AH_x}$ sends an account_lock_release($AA^{ID}$) message to all account holders.

---

**D.1.1.5 Account Movement**

The account movement algorithm is used for two purposes. First, it moves an aggregation account to the correct location on the DHT. Second, it creates a subsequent account holder set with the correct size. In order to achieve this the algorithm is split into two parts. The first part verifies the location of the first account holder. The second part actually moves the aggregation account to the new position. The algorithms are similar to the account creation described in Section 4.2.1.5.

The first part, Algorithm 4.8, is executed by the first account holder of the set, $P_{AH_1}$. First, the consistency of all account holders must be ensured (step 1). Otherwise, during the account movement existing inconsistencies could increase.

The correct location of the account holder set is determined by a trusted peer, as this part of the algorithm reveals some part of the DHT. This knowledge should not be revealed to normal peers as here a higher chance for misuse is given. Accordingly, $P_{AH_1}$ sends a request_account_movement-message to a trusted peer $TP$. The message contains the account's account shift value $x$, as only account holders know this value (step 2). $TP$ will lookup the peer responsible for the aggregation account ID in the DHT, $P_C$ (step 4). Then, $TP$ finds the first peer of the correct account holder set using the same multiple step processes as described in Algorithm 4.3 (steps 5 - 9). Finally, $TP$ sends $P_{AH_i}$ a peer ID ($P_j$) that lies within the new first account holders ID range contained in a account_movement_result-message.

![Algorithm 4.8 Detect New Account Position](#)

This algorithm determines the correct position for an aggregation account.

1. The current first account holder $P_{AH_1}$ executes the consistency algorithm 4.12.

2. $P_{AH_1}$ creates a request_account_movement($AA^{ID}$, $x$) message and sends it to a trusted peer $TP$.

3. $TP$ receives the request_account_movement($AA^{ID}$, $x$) message.

4. $TP$ looks up the peer responsible for $AA^{ID}$, denoted $P_C$.

5. $TP$ randomly picks $r$ values $y$, where $1 + 2r + \sum_{i=1}^{r} y_r = x$.

6. Set $P_k = P_C$.

7. For $i = 1$ to $r$ do:
   1. Set $P_i = P_k$.
   2. $TP$ looks up the $P_i$’s successor on the DHT, $P_j$.
   3. $TP$ sends request_peer message containing $(y_i - 1)$ to $P_j$. 

---

D.1. Account Holder Set
4. \( P_j \) forwards the request_peer message to its \((y_i - 1)\)-th successor, \( P_m \).

5. \( P_m \) responds to \( TP \) with its successor’s peer ID, \( P_k \).

8. \( P_k \) is the peer with \( x - 1 \) hops distance to \( P_C \).

9. \( TP \) adds a small value to \( P_k \)’s peer ID to create a value that lies within the \( P_k \)'s successor's ID range, denoted \( P_x \).

10. \( TP \) creates an account_movement_result\((P_x)\) message and sends it to \( P_{AH_1} \).

Now, the current first account holder, \( P_{AH_1} \), will administer the moving of the account holder set to its new position. It receives the peer ID \( P_x \) from a trusted peer (step 1). If this peer ID lies within its own ID range, \( P_{AH_1} \) is also the first account holder in the new account holder set. Else, \( P_{AH_1} \) has to lookup the peer responsible for \( P_x \). This peer is denoted \( P_n \). \( P_{AH_1} \) will send a new_first_account_holder-message to \( P_n \), containing the aggregation account ID \( AA_{IB} \) as well as the list of the current account holders. (step 3). \( P_n \) has now to determine the new account holder set.

It sends a shift_aggregation_account-message, containing a hop counter set to 1 and its own peer ID, to its successor in the DHT (step 4). The peer ID is the first entry of a list that will be used to collect the peer ID of all new account holders.

The successor \( P_s \) receiving the shift_aggregation_account-message will increase the hop counter by one, add its peer ID to the list, and remember the hop counter’s value as its own position in the account holder set. Then it forwards the message to its successor on the DHT (step 6). This stops if either the hop counter has reached the preferred account holder set size or if a peer receives the message again, that has received the message before. The latter case means that the p2p system has only very few online peers and the message has traversed the complete DHT ring. Then, \( P_s \) sets the account holder set size to the current hop counter’s value. The new account holder set is complete now. All the new account holders must now be informed about the composition of the new account holder set. Therefore, the last peer in the new account holder set will create a shifting_aggregation_account-message that contains the list of the new account holders. It will send this message to all new account holders (step 6f).

A peers \( P_p \) receiving the shifting_aggregation_account-message will remember the list of new account holders (step 7). If it is also an account holder in the current set, then no further actions are required. Only if \( P_p \) is not an account holder in the current set it needs to retrieve now the account information from such a peer. For that it will choose the peer in the current set that is at the same account position as \( P_p \) in the new set. If this position was not occupied, \( P_p \) chooses a random account holder from the current set. This will distribute the workload among the current account holders. Further, it ensures that a potential incorrect account replica will not be multiplied in the new set.

Now the account information transfer needs to be performed. However, it must be ensured that only peers permitted as account holders will receive the information. Therefore, the new account holders will forward the shifting_aggregation_account-message to the current account holders that were selected for the information transfer (step 8). The current account holders will transfer their account information using account_transfer-messages. When the transfer was successful, the new account holders will respond with a transfer_complete-message. Then, the current account holders can delete the account.

In order to ensure that only the selected peers are taking place in the account movement, all communication uses reliable authenticated connections.

Algorithm 4.9  Move Account to New Position

This algorithm moves an aggregation account to the correct position on the DHT.

1. Account holder \( P_{AH_1} \) receives an account_movement_result\((P_x)\) message.

2. \( P_{AH_1} \) waits until Algorithm 4.12 is completed.
3. If $P_x$ lies not in $P_{AH_1}$’s ID range,
   1. then
      1. $P_{AH_1}$ looks up the responsible peer for $P_x$ in the DHT, denoted as $P_n$.
      2. $P_{AH_1}$ creates a new_first_account_holder($AA^{ID}$, $P_x$) message and adds a list of the current account holders to it.
      3. $P_{AH_1}$ sends the new_first_account_holder($AA^{ID}$, $P_x$) to $P_n$.
   2. else $P_n = P_{AH_1}$.

4. $P_n$ creates a shift_aggregation_account($AA^{ID}$) message, adds the list of current account holders to the message and it adds a list of new account holders to the message containing its own peerID. It sets its hop counter to 1, and forwards the message to its successor on the DHT, $\text{succ}(P_n)$.

5. $P_n$ remembers the hop counter as its own position within the account holder set.

6. Whenever a peer $P_s$ receives a shift_aggregation_account message,
   1. $P_s$ checks if it received this message before. If so it sets $AS = h$ and proceeds to step 6.6.
   2. $P_s$ checks if it is already account holder for account $AA^{ID}$. If not, it instantiates a new aggregation account for $AA^{ID}$.
   3. It adds its peerID to the message and increases the message’s hop counter $h$ by one.
   4. It remembers the list of current account holders.
   5. It remembers the hop counter as its own position within the new account holder set.
   6. Let $AS$ be the preferred account holder set size in the p2p system.
      1. If $h = AS$ then
         • $P_s$ creates an shifting_aggregation_account($AA^{ID}$) message containing the list of new account holders from the received shift_aggregation_account message.
         • It remembers the list of new account holders in the aggregation account.
         • It sends this message to all peers in the new account holder list.
      2. else, $P_s$ forwards the shift_aggregation_account to its successor on the DHT, $\text{succ}(P_s)$.

7. Whenever a peer $P_p$ receives an shifting_aggregation_account($AA^{ID}$) message it
   1. remembers the list of new account holders contained in the message in the aggregation account.
   2. if $P_p$ is not an account holder in the current account holder set, then
      1. $P_p$ chooses from the list of current account holder the peer with the account holder set position equal to its own account holder set position in the new account holder set, denoted $P_t$.
         If this position is not assigned in the current account holder set $P_p$ chooses a random peer from the current account holder set as $P_t$.
      2. $P_p$ creates a request_account_transfer($AA^{ID}$) message and sends it to $P_t$.

8. When $P_n$ receives the shifting_aggregation_account($AA^{ID}$) message it forwards it to all peers in the current account holder list.

9. Whenever a peer $P_t$ receives a request_account_transfer($AA^{ID}$) message from a peer $P_p$ contained in the new account holder list, $P_t$ creates an account_transfer($AA^{ID}$) message, containing all account data of account $AA^{ID}$ and sends this message to peer $P_t$. 

D.1. Account Holder Set
10. Whenever a peer $P_p$ receives an account_transfer($AA^{ID}$) message and stores the contained account information successfully, $P_p$ sends a transfer_complete($AA^{ID}$) message to $P_t$.

11. Whenever a peer $P_t$ receives a transfer_complete($AA^{ID}$) message, it removes the aggregation account $AA^{ID}$.

### D.1.1.6 Graceful Aggregation Account Handover

In order to avoid a complete new account assignment when an account holder leaves the p2p system an account holder should handover the accounts it hosts to another peer. Whenever an account holder plans to leave the system it will execute Algorithm 4.10 in order to do a clean log off. This is called Graceful Handover. The peer leaving the p2p system $P_o$ has to do the graceful handover for all the aggregation accounts it hosts (step 2).

First, $P_o$ first informs all account holders about its planned leave, using a leaving_system-message. When an account holder leaves, it means that the account holder set must be extended at the end of the set. If it would add a new peer the front, the aggregation account was shifted one hop towards the peer responsible for the aggregation account ID on the DHT. Therefore, the last account holder in the set $P_l$ is sending an account_handover-message to its successor on the DHT, denoted $P_m$. $P_m$ will replace $P_o$ in the account holder set. In order to ensure that the account is transferred to the correct peer, $P_m$ will now contact $P_o$ using handover_peer-message. This message serve as authentication to $P_o$ that it should transfer its account to $P_m$. $P_o$ uses an account_transfer-message to transfer all account information to $P_m$. After a successful account transfer $P_m$ will inform all account holders including $P_o$ that it is now storing the aggregation account using a transfer_complete-message. All account holders will update their account holder list and $P_o$ may leave the p2p system.

#### Algorithm 4.10 Graceful Aggregation Account Handover

This algorithm is executed by an account holder that plans to leave the p2p system in order to transfer its accounts to other peers.

1. Peer $P_l$ plans to leave the p2p system.

2. For each aggregation account $P_l$ hosts do
   1. $P_o$ sends a leaving_systems($P_o$, $AA^{ID}$) message to all account holders of account $AA^{ID}$.
   2. The last peer in the account holder set, denoted $P_l$, creates a account_handover($AA^{ID}$, $P_o$) message
   3. $P_l$ sends the account_handover($AA^{ID}$, $P_o$) message to its successor on the DHT, denoted $P_m$.
   4. $P_l$ sends a handover_peer($AA^{ID}$, $P_m$) message to $P_o$.
   5. $P_o$ creates an account_transfer($AA^{ID}$) message.
   6. $P_o$ sends the account_transfer($AA^{ID}$) message to $P_m$.
   7. $P_m$ creates a transfer_complete($AA^{ID}$) message to all account holders of aggregation account $AA^{ID}$, including $P_o$.
   8. all account holders for aggregation account $AA^{ID}$ update their account holder list.

3. $P_o$ leaves the p2p system.
D.1.2 Information Storage

D.1.2.1 Storing Information in an Account Holder Set

Algorithm 4.11  Post Account Information at an Account Holder Set

Peer $P_A$ needs to contact account aggregation account $AA^{ID}$ for requesting account information or posting account information. This algorithm ensures, that the message is delivered to the account holder set, and that the responses are delivered to peer $P_A$.

1. Peer $P_A$ creates a aa_request message. This is either an aa_read_request($AA^{ID}$, query) message for requesting account information. Or it is an aa_write_request($AA^{ID}$, post) message for posting account information.

2. $P_A$ looks up the peerID of $P_C$, the peer responsible for $AA^{ID}$ in the DHT.

3. $P_A$ sends the aa_request message to $P_C$.

4. Each peer $P_i$ receiving an aa_request message,
   1. $P_i$ checks if the message signee is permitted to request or post account information for account $AA^{ID}$. If not, $P_C$ drops the message and files a reputation report.
   2. If $P_i$ is permitted to send the aa_request message, $P_i$ checks if it hosts account $AA^{ID}$.
      1. If $P_i$ hosts account $AA^{ID}$, $P_i$ transforms the aa_request message into an enqueue_aa_request($AA^{ID}$) message. $P_i$ forwards the enqueue_aa_request message($AA^{ID}$) to all account holders for account $AA^{ID}$. It enqueues the aa_request.
      2. If not, $P_i$ forwards the aa_request message to its successor on the DHT.

5. Whenever an account holder $P_{AH^{ID}}$ for account $AA^{ID}$ receives an enqueue_aa_request($AA^{ID}$) message it will enqueue it in the account queue and process it when appropriate.

   1. $P_{AH^{ID}}$ will create an aa_response($AA^{ID}$) message.

6. If the aa_response message is of type aa_write_response, then
   1. $P_{AH^{ID}}$ sends the response to $P_{AH}$, where $P_{AH}$ is the peer that send the enqueue_aa_request($AA^{ID}$) message.
   2. $P_{AH}$ will collect all aa_response($AA^{ID}$) messages.
      1. If $P_{AH}$ receives more than $AHSS^{ID}/2$ consistent aa_response($AA^{ID}$) messages, where $AHSS^{ID}$ is the current account holder set of aggregation account $AA^{ID}$.
         A. $P_{AH}$ will lookup the peer responsible for $AA^{ID}$ on the overlay, $P_A$.
         B. $P_{AH}$ will send the aa_response($AA^{ID}$) message to $P_A$.
         C. If $P_{AH}$ received inconsistent aa_response($AA^{ID}$) messages, it executes Algorithm 4.12.
      2. Else if $P_{AH}$ receives less than $AHSS^{ID}/2$ aa_response($AA^{ID}$) messages,
         A. $P_{AH}$ will send an re-enqueue_aa_request($AA^{ID}$) message to all account holders for account $AA^{ID}$.
         B. $P_{AH}$ will execute Algorithm 4.5 and any further required account management algorithms.
3. Else if $P_{AH_i}$ receives less than $AHSS^{ID}/2$ consistent $aa\_response(AA^{ID})$ messages, it
   A. $P_{AH_i}$ will send an re-enqueue_aa_request($AA^{ID}$) message to all account holders for account $AA^{ID}$.
   B. $P_{AH_i}$ executes Algorithm 4.12.
3. Each account holder peer $P_{AH_X}$
   1. After account management algorithms or the account consensus algorithm have terminated, $P_{AH_X}$ processes a re-enqueue_aa_request($AA^{ID}$) message from the FIFO-queue as $P_{AH_i}$ starting at step 4.
4. Each peer $P_{C}$ receiving an $aa\_response(AA^{ID})$ message forwards it to $P_{A}$.
7. Else, $P_{AH_{ID}}$ sends the response to $P_{A}$.
8. An account holder $P_{AH_X}$ receiving a re-enqueue_aa_request($AA^{ID}$) message will
   1. halt processing events.
   2. enqueue the $aa\_request$ contained in the re-enqueue_aa_request($AA^{ID}$) message at the first position of the FIFO-queue.
   3. After the next consistency algorithm (Algorithm 4.12) has been finalised, processing events is continued.

---

**D.1.3 Consensus Mechanism**

**D.1.3.1 Account Holder Set Consensus Mechanism**

**Algorithm 4.12  Aggregation Account Consistency Algorithm**

The algorithm ensures that all replicas in an account holder set store consistent information. The consistency algorithm is initialised by one quorum peer, $P_{AH_1}$. $P_{AH_i}$ has successfully requested a lock for aggregation account $AA^{ID}$ using Algorithm 4.7.

1. While there are inconsistencies in the account do
   1. $P_{AH_1}$ creates an aggregation account summary $S_{A}^{ID}$.
   2. $P_{AH_1}$ creates a request_consistency_check($AA^{ID}$, account summary) message and sends it to all account holders.
   3. Whenever an account holder $P_{AH_X}$ receives a request_consistency_check($AA^{ID}$, account summary) message it
      1. creates an aggregation account summary $S_{X}^{ID}$
      2. compares the received account summary $S_{X}^{ID}$ against its own one $S_{X}^{ID}$ line by line.
      3. if there are any line inconsistencies or $S_{X}^{ID}$ has additional lines in $S_{X}^{ID}$, $P_{AH_X}$ creates a request_update($AA^{ID}$, diff($S_{1}^{ID}$, $S_{X}^{ID}$)) message. $P_{AH_X}$ sends the message to all account holders.
      4. If $S_{1}^{ID} = S_{X}^{ID}$, then $P_{AH_X}$ creates a consistent_account($AA^{ID}$) message and sends it to all account holders.
4. Each account holder $P_{AH}$ waits to receive the request_update($AA^{ID}$, diff($S_1^{ID}$, $S_X^{ID}$)) messages or consistent_account($AA^{ID}$) messages.

5. Each account holder $P_{AH}$ accepts line changes, if the simple majority of peers agree on this information.

6. Each account holder $P_{AH}$ accepts a new line in its account, if $> p_{nl}\%$ of all account holders store that line in their account.

7. Each account holder that stores a line that $< p_{nl}\%$ of all account holders also store, has to delete this line.

2. End While

---

### D.2 Trusted Peers

#### D.2.1 Finding Trusted Peers

The algorithm to locate a trusted peer is described in Algorithm 4.13. The peer $P_x$ is looking for the trusted peer next to $ID_s$. First it needs to lookup the peer $P_s$ responsible for $ID_s$. To $P_s$ it sends a get-next_TP-message to $P_s$. $P_s$ will respond with the overlay address of the next trusted peer TP. If TP left in the meantime, then $P_s$ will use this algorithm to retrieve the next trusted peer of TP’s successor in the DHT before responding to $P_x$.

**Algorithm 4.13 Locate Trusted Peer**

This algorithm locates the first trusted peer after the overlay address $ID_s$ on the system overlay. Each peer maintains a finger to the next trusted peer $TP$ in the DHT. If the peer is a trusted peer itself, the finger points to itself.

1. $P_s$ does a lookup on the DHT of $ID_s$. It receives $P_s$ as return.

2. $P_s$ creates a get-next-TP($P_s$)-message and sends it to $P_s$.

3. Any peer $P_t$ receiving the get-next-TP($P_s$)-message will:
   1. If $P_t$ is a trusted peer, $P_t$ sends a next-TP($P_t$) message to $P_s$.
   2. Else,
      1. If $TP$ is still alive, $P_t$ sends a next-TP($TP$) message to $P_s$.
      2. Else,
         A. $P_t$ runs this Algorithm for ID $TP$ resulting in $TP_{new}$.
         B. sends a next-TP($TP_{new}$) message to $P_s$. 
D.3 Secure Aggregation Protocol

D.3.1 Protocol Details

Algorithm 4.14 Secure Token Aggregation

A peer $SP$ swaps collected foreign tokens against new own tokens. The trusted peer creating the fresh tokens as well as the quorum peers signing the fresh tokens partially are selected randomly using information stored in $SP$’s aggregation account. The account holders check that only the selected trusted peers take part in the aggregation process.

1. $SP$ sends $b$ foreign tokens to random peer $TP_1$ using a $aggregate_token(F_1, ..., F_b)$-message.

2. $TP_1$ sends a $request_account_stamp(AAS_{SP})$ message to $SP$’s account holder set $AHS_{SP}$.
   1. $AHS_{SP}$ calculates a hash value $h_{SP}$ of the account information.
   2. $AHS_{SP}$ uses Algorithm 4.13 to locate the next trusted peer of $h_{SP}$ in the system overlay, $TP_2$.
   3. If $TP_2 = TP_1$, $AHS_{SP}$ uses Algorithm 4.13 to locate the next trusted peer of $inverse(h_{SP})$ in the system overlay $TP_2$.
   4. $AHS_{SP}$ sends an $account_stamp_result(TP_2)$ message to $TP_1$.

3. $TP_1$ forwards the $aggregate_token(F_1, ..., F_b)$-message to $TP_2$.

4. $TP_2$ will act as aggregation administrator. $TP_2$ calculates the number of new token $N = A(F_1, ..., F_b)$ to be created. $TP_2$ creates $n$ new token $U_1, ..., U_n$.

5. $TP_2$ sends the new tokens’ issuing information to $AHS_{SP}$ using a $new_tokens(AAS_{SP}, (U_1, ..., U_n))$ message.
   1. $AHS_{SP}$ check if the message was sent by $TP_2$. If not, a reputation report is filed against the sender.
   2. $AHS_{SP}$ calculates a hash value $h′_{SP}$ of the account information.
   3. $l = h′_{SP}$
   4. for $i = 1$ to $t$
      1. $AHS_{SP}$ uses Algorithm 4.13 to locate the next trusted peer of $l$ in the system overlay, $QP_i$.
      2. $l = QP_i + 1$
   5. $AHS_{SP}$ send a $new_tokens_quorum(QP_1, ..., QP_t)$ message to $TP_2$.

6. $TP_2$ sends $t$ fresh_tokens($SP$, $(U_1, ..., U_n)$) message to $QP_1, ..., QP_t$.

7. Each quorum peer $QP_1, ..., QP_t$ sends a $new_partial_tokens(AAS_{SP}, (U_1, ..., U_n))$ message to $AHS_{SP}$.
   1. If $AHS_{SP}$ receives $new_partial_tokens(AAS_{SP}, (U_1, ..., U_n))$ messages from $QP_1, ..., QP_t$, then $AHS_{SP}$ add the fresh tokens $U_1, ..., U_n$ to $AAS_{SP}$.
   2. If $AHS_{SP}$ receives a $new_partial_tokens(AAS_{SP}, (U_1, ..., U_n))$ messages from a different peer, $AHS_{SP}$ files a reputation report against the sending peer.
   3. The quorum peers $QP_1, ..., QP_t$ sign the fresh tokens $U_1, ..., U_n$ partially.

Appendix D. Supplementary Details to Chapter 4 269
8. The quorum peers $Q_P_1, ..., Q_P_t$ send a new_tokens($P_1, ..., P_n$) message to $SP$. $SP$ creates $n$ new tokens from $P_1, ..., P_n$. 

D.3. Secure Aggregation Protocol
E Supplementary Simulation Results

E.1 Account Holder Set Size Simulations

E.1.1 Summary of Results of Account Holder Set Size Simulations

Table E.1: Required Holder Set Sizes for Different Churn Models – Details

<table>
<thead>
<tr>
<th>$p_{gh}$</th>
<th>$d_{ga}$</th>
<th>$k$</th>
<th>$\tilde{k}_{k=3}$</th>
<th>$\tilde{k}_{k=4}$</th>
<th>$\tilde{k}_{k=5}$</th>
<th>$\tilde{k}_{k=6}$</th>
<th>$\tilde{k}_{k=7}$</th>
<th>$\tilde{k}_{k=8}$</th>
<th>$\tilde{k}_{k=9}$</th>
<th>$\tilde{k}_{k=10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weibull</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>14511.2</td>
<td>4.19E+06</td>
<td>6.01E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/2</td>
<td>6</td>
<td>139.25</td>
<td>60536.15</td>
<td>8.02E+06</td>
<td>5.97E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>6</td>
<td>839.05</td>
<td>125505.1</td>
<td>1.17E+07</td>
<td>5.93E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>7</td>
<td>4.2</td>
<td>2751.8</td>
<td>3.98E+05</td>
<td>1.56E+07</td>
<td>5.87E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>7</td>
<td>91.3</td>
<td>6551.35</td>
<td>5.39E+05</td>
<td>2.32E+07</td>
<td>5.81E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/2</td>
<td>7</td>
<td>20.9</td>
<td>2001.8</td>
<td>8.69E+04</td>
<td>2.52E+06</td>
<td>4.87E+07</td>
<td>5.54E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/4</td>
<td>9</td>
<td>479.3</td>
<td>1.65E+04</td>
<td>3.81E+05</td>
<td>6.52E+06</td>
<td>7.49E+07</td>
<td>5.23E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/4</td>
<td>10</td>
<td>0</td>
<td>137.4</td>
<td>3.65E+03</td>
<td>6.81E+04</td>
<td>1.12E+06</td>
<td>1.28E+07</td>
<td>1.01E+08</td>
<td>4.90E+08</td>
<td>–</td>
</tr>
<tr>
<td>Mixed Log-normal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>1</td>
<td>6</td>
<td>21.1</td>
<td>16484.4</td>
<td>4.24E+06</td>
<td>6.01E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/2</td>
<td>6</td>
<td>301.1</td>
<td>606831</td>
<td>8.54E+06</td>
<td>5.96E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>7</td>
<td>47.2</td>
<td>2089.1</td>
<td>2.18E+05</td>
<td>1.49E+07</td>
<td>5.90E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/4</td>
<td>7</td>
<td>4.35</td>
<td>4554.2</td>
<td>3.85E+05</td>
<td>1.98E+07</td>
<td>5.85E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>7</td>
<td>52.5</td>
<td>7318.5</td>
<td>5.56E+05</td>
<td>2.36E+07</td>
<td>5.81E+08</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/2</td>
<td>9</td>
<td>0</td>
<td>92.5</td>
<td>4.60E+03</td>
<td>1.55E+05</td>
<td>3.59E+06</td>
<td>5.68E+07</td>
<td>5.44E+08</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/4</td>
<td>9</td>
<td>0</td>
<td>545.75</td>
<td>2.13E+04</td>
<td>4.88E+05</td>
<td>7.74E+06</td>
<td>8.05E+07</td>
<td>5.16E+08</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1/4</td>
<td>9</td>
<td>44.3</td>
<td>2220.7</td>
<td>6.32E+04</td>
<td>1.10E+06</td>
<td>1.29E+07</td>
<td>1.01E+08</td>
<td>4.90E+08</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

* with $k = 9$ there happened a $\tilde{k} = 2$.
** with $k = 6$ there happened a $\tilde{k} = 2$.

This table summarises the simulation results for determining the required account holder set sizes $k$, where $\tilde{k}$ denotes the actual actual account holder set size in the simulation and $\tau$ denotes the accumulated time a specific actual account holder set size was observed on average.
### E.1.2 Individual Results of Account Holder Set Size Simulations by Experiments

#### Table E.2: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 1$, $p_{\text{log-off}} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>14.5112</td>
<td>12.66929255</td>
<td>16.35310745</td>
</tr>
<tr>
<td>5</td>
<td>4186.18535</td>
<td>4134.405231</td>
<td>4237.965469</td>
</tr>
<tr>
<td>6</td>
<td>600599.3035</td>
<td>600547.7174</td>
<td>600650.8895</td>
</tr>
</tbody>
</table>

#### Table E.3: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.5$, $p_{\text{log-off}} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.13925</td>
<td>0.065083433</td>
<td>0.213416567</td>
</tr>
<tr>
<td>4</td>
<td>605.3615</td>
<td>55.73625269</td>
<td>65.33604731</td>
</tr>
<tr>
<td>5</td>
<td>8015.91645</td>
<td>7954.948449</td>
<td>8076.884451</td>
</tr>
<tr>
<td>6</td>
<td>596723.4082</td>
<td>596660.7724</td>
<td>596786.0439</td>
</tr>
</tbody>
</table>

#### Table E.4: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.333$, $p_{\text{log-off}} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.83905</td>
<td>0.456680494</td>
<td>1.221419506</td>
</tr>
<tr>
<td>4</td>
<td>125.5051</td>
<td>120.1931457</td>
<td>130.8170543</td>
</tr>
<tr>
<td>5</td>
<td>11678.8654</td>
<td>11583.06884</td>
<td>11774.66196</td>
</tr>
<tr>
<td>6</td>
<td>592994.7905</td>
<td>5928977.799</td>
<td>593091.801</td>
</tr>
</tbody>
</table>

#### Table E.5: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.25$, $p_{\text{log-off}} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.0042</td>
<td>-0.0030618</td>
<td>0.0114618</td>
</tr>
<tr>
<td>4</td>
<td>2.7518</td>
<td>2.038542122</td>
<td>3.465057878</td>
</tr>
<tr>
<td>5</td>
<td>307.7595</td>
<td>293.8814196</td>
<td>321.6375804</td>
</tr>
<tr>
<td>6</td>
<td>17638.4362</td>
<td>17546.49684</td>
<td>17730.37556</td>
</tr>
<tr>
<td>7</td>
<td>586851.0483</td>
<td>586759.6963</td>
<td>586942.4003</td>
</tr>
</tbody>
</table>
Table E.6: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 1, p_{\text{log-off}} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.0913</td>
<td>-0.018932188</td>
</tr>
<tr>
<td>4</td>
<td>6.55135</td>
<td>5.71499812</td>
</tr>
<tr>
<td>5</td>
<td>538.80695</td>
<td>528.445932</td>
</tr>
<tr>
<td>6</td>
<td>23232.5741</td>
<td>23087.52213</td>
</tr>
<tr>
<td>7</td>
<td>581021.9763</td>
<td>580873.8217</td>
</tr>
</tbody>
</table>

Table E.7: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.5, p_{\text{log-off}} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.0209</td>
<td>-0.000634609</td>
</tr>
<tr>
<td>4</td>
<td>2.0018</td>
<td>1.577086944</td>
</tr>
<tr>
<td>5</td>
<td>85.90765</td>
<td>81.34118568</td>
</tr>
<tr>
<td>6</td>
<td>2518.18265</td>
<td>2489.881911</td>
</tr>
<tr>
<td>7</td>
<td>48656.188</td>
<td>48489.10259</td>
</tr>
<tr>
<td>8</td>
<td>553537.699</td>
<td>553359.9735</td>
</tr>
</tbody>
</table>

Table E.8: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.333, p_{\text{log-off}} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.4793</td>
<td>0.296143561</td>
</tr>
<tr>
<td>5</td>
<td>16.49105</td>
<td>14.98341095</td>
</tr>
<tr>
<td>6</td>
<td>381.4272</td>
<td>371.5331553</td>
</tr>
<tr>
<td>7</td>
<td>6521.73415</td>
<td>6475.609043</td>
</tr>
<tr>
<td>8</td>
<td>74945.12875</td>
<td>74752.42166</td>
</tr>
<tr>
<td>9</td>
<td>522934.7396</td>
<td>522723.302</td>
</tr>
</tbody>
</table>

Table E.9: Detailed Simulation Results for Weibull Distribution, $d_{\text{var}} = 0.25, p_{\text{log-off}} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.1374</td>
<td>0.053829663</td>
</tr>
<tr>
<td>5</td>
<td>3.6524</td>
<td>3.047345097</td>
</tr>
<tr>
<td>6</td>
<td>68.1156</td>
<td>65.50443581</td>
</tr>
<tr>
<td>7</td>
<td>1121.5752</td>
<td>1106.302276</td>
</tr>
<tr>
<td>8</td>
<td>12811.54405</td>
<td>12730.55114</td>
</tr>
<tr>
<td>9</td>
<td>100565.3186</td>
<td>100324.2878</td>
</tr>
<tr>
<td>10</td>
<td>490229.6568</td>
<td>489919.3045</td>
</tr>
<tr>
<td>Set size</td>
<td>Duration [sec]</td>
<td>Confidence interval (90%) [sec]</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.0211</td>
<td>-0.006378506</td>
</tr>
<tr>
<td>4</td>
<td>16.4844</td>
<td>14.78385328</td>
</tr>
<tr>
<td>5</td>
<td>4239.9669</td>
<td>4192.853456</td>
</tr>
<tr>
<td>6</td>
<td>600543.5276</td>
<td>600496.4344</td>
</tr>
</tbody>
</table>

Table E.11: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.5, p_{log-off} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.3011</td>
<td>0.13847708</td>
</tr>
<tr>
<td>4</td>
<td>66.831</td>
<td>63.63869262</td>
</tr>
<tr>
<td>5</td>
<td>8542.75385</td>
<td>8480.526971</td>
</tr>
<tr>
<td>6</td>
<td>596190.1141</td>
<td>596128.274</td>
</tr>
</tbody>
</table>

Table E.12: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.333, p_{log-off} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.0472</td>
<td>-0.03225182</td>
</tr>
<tr>
<td>4</td>
<td>2.0891</td>
<td>1.586739654</td>
</tr>
<tr>
<td>5</td>
<td>218.34095</td>
<td>212.1255874</td>
</tr>
<tr>
<td>6</td>
<td>14896.7875</td>
<td>14792.12777</td>
</tr>
<tr>
<td>7</td>
<td>589682.7349</td>
<td>589579.2939</td>
</tr>
</tbody>
</table>

Table E.13: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.25, p_{log-off} = 0.9$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.00435</td>
<td>-0.000911203</td>
</tr>
<tr>
<td>4</td>
<td>4.5542</td>
<td>3.7869363</td>
</tr>
<tr>
<td>5</td>
<td>384.99785</td>
<td>376.2815899</td>
</tr>
<tr>
<td>6</td>
<td>19764.2178</td>
<td>19692.4969</td>
</tr>
<tr>
<td>7</td>
<td>584646.2258</td>
<td>584572.1543</td>
</tr>
</tbody>
</table>

Table E.14: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 1, p_{log-off} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Confidence interval (90%) [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower Bound</td>
</tr>
<tr>
<td>3</td>
<td>0.0525</td>
<td>0.013866984</td>
</tr>
<tr>
<td>4</td>
<td>7.3185</td>
<td>6.53659705</td>
</tr>
<tr>
<td>5</td>
<td>556.14745</td>
<td>545.4125728</td>
</tr>
<tr>
<td>6</td>
<td>23571.0399</td>
<td>23473.26605</td>
</tr>
<tr>
<td>7</td>
<td>580665.4417</td>
<td>580561.2539</td>
</tr>
</tbody>
</table>
Table E.15: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.5, p_{log-off} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.0925</td>
<td>0.007521203</td>
<td>0.177478797</td>
</tr>
<tr>
<td>5</td>
<td>4.5964</td>
<td>3.955899984</td>
<td>5.236900016</td>
</tr>
<tr>
<td>6</td>
<td>154.63935</td>
<td>149.2204756</td>
<td>160.0582244</td>
</tr>
<tr>
<td>7</td>
<td>358.95735</td>
<td>355.67284</td>
<td>361.24186</td>
</tr>
<tr>
<td>8</td>
<td>5679.534</td>
<td>5658.75167</td>
<td>5699.31633</td>
</tr>
<tr>
<td>9</td>
<td>54425.1804</td>
<td>54405.3023</td>
<td>54446.0585</td>
</tr>
</tbody>
</table>

Table E.16: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.333, p_{log-off} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.54575</td>
<td>0.393253579</td>
<td>0.698246421</td>
</tr>
<tr>
<td>5</td>
<td>21.3146</td>
<td>19.4080752</td>
<td>23.2211248</td>
</tr>
<tr>
<td>6</td>
<td>488.2935</td>
<td>480.6382651</td>
<td>495.9487394</td>
</tr>
<tr>
<td>7</td>
<td>7736.7252</td>
<td>7697.447219</td>
<td>7776.003181</td>
</tr>
<tr>
<td>8</td>
<td>8047.56075</td>
<td>8029.35164</td>
<td>8065.38986</td>
</tr>
<tr>
<td>9</td>
<td>51607.5602</td>
<td>51589.3946</td>
<td>51626.7258</td>
</tr>
</tbody>
</table>

Table E.17: Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.25, p_{log-off} = 0.5$

<table>
<thead>
<tr>
<th>Set size</th>
<th>Duration [sec]</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.0443</td>
<td>-0.001209189</td>
<td>0.089809189</td>
</tr>
<tr>
<td>4</td>
<td>2.2207</td>
<td>1.685776111</td>
<td>2.755623889</td>
</tr>
<tr>
<td>5</td>
<td>63.23645</td>
<td>60.18966383</td>
<td>66.28323617</td>
</tr>
<tr>
<td>6</td>
<td>1103.952</td>
<td>1088.993221</td>
<td>1118.910779</td>
</tr>
<tr>
<td>7</td>
<td>12907.32945</td>
<td>12849.57859</td>
<td>12965.08031</td>
</tr>
<tr>
<td>8</td>
<td>100665.5566</td>
<td>100482.7898</td>
<td>100848.3233</td>
</tr>
<tr>
<td>9</td>
<td>490057.6606</td>
<td>489846.6223</td>
<td>490268.6988</td>
</tr>
</tbody>
</table>
E.2 Token-based Accounting Scheme Simulations

E.2.1 Overall Traffic

E.2.1.1 Overall Traffic by Quorum Size

Overall Traffic for $t = 17$, $k = 10$, $d_{var} = 1$

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.1: System-wide traffic for $t = 17$, $k = 10$, $d_{var} = 1$

Overall Traffic for $t = 14$, $k = 10$, $d_{var} = 1$

(a) Complete traffic vs accounting traffic

(b) Traffic breakdown

Figure E.2: System-wide traffic for $t = 14$, $k = 10$, $d_{var} = 1$
Overall Traffic for $t = 10, k = 10, d_{var} = 1$

(a) Complete traffic vs accounting traffic

(b) Traffic breakdown

Figure E.3: System-wide traffic for $t = 10, k = 10, d_{var} = 1$

Overall Traffic for $t = 7, k = 10, d_{var} = 1$

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.4: System-wide traffic for $t = 7, k = 10, d_{var} = 1$
E.2.1.2 Overall Traffic by Account Holder Set Size

Overall Traffic for $t = 17$, $k = 12$, $d_{var} = 1$

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.5: System-wide traffic for $t = 17$, $k = 12$, $d_{var} = 1$

Overall Traffic for $t = 17$, $k = 8$, $d_{var} = 1$

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.6: System-wide traffic for $t = 17$, $k = 8$, $d_{var} = 1$
Overall Traffic for $t = 17, k = 6, d_{var} = 1$

Figure E.7: System-wide traffic for $t = 17, k = 6, d_{var} = 1$

E.2.1.3 Overall Traffic by Churn Factor

Overall Traffic for $t = 17, k = 10, d_{var} = 0.5$

Figure E.8: System-wide traffic for $t = 17, k = 10, d_{var} = 0.5$
Overall Traffic for \( t = 17, k = 10, d_{\text{var}} = 0.33 \)

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.9: System-wide traffic for \( t = 17, k = 10, d_{\text{var}} = 0.33 \)

Overall Traffic for \( t = 17, k = 10, d_{\text{var}} = 0.25 \)

(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure E.10: System-wide traffic for \( t = 17, k = 10, d_{\text{var}} = 0.25 \)
E.2.2 Traffic Breakdown

E.2.2.1 Traffic Breakdown by Quorum Size

Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 1$

![Graph](image1)

(a) System-wide traffic

![Graph](image2)

(b) Average traffic per active peer

Figure E.11: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 1$

![Graph](image3)

(a) Traffic ratios including service traffic

![Graph](image4)

(b) Traffic ratios of token-based accounting

Figure E.12: System-wide traffic ratios for $t = 17$, $k = 10$, $d_{var} = 1$
Traffic Breakdown for $t = 14, k = 10, d_{var} = 1$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.13: System-wide token-based accounting traffic breakdown for $t = 14, k = 10, d_{var} = 1$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.14: System-wide traffic ratios for $t = 14, k = 10, d_{var} = 1$
Traffic Breakdown for $t = 10, k = 10, d_{var} = 1$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.15: System-wide token-based accounting traffic breakdown for $t = 10, k = 10, d_{var} = 1$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.16: System-wide traffic ratios for $t = 10, k = 10, d_{var} = 1$
Traffic Breakdown for $t = 7, k = 10, d_{var} = 1$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.17: System-wide token-based accounting traffic breakdown for $t = 7, k = 10, d_{var} = 1$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.18: System-wide traffic ratios for $t = 7, k = 10, d_{var} = 1$
E.2.2.2 Traffic Breakdown by Account Holder Set Size

Traffic Breakdown for \( t = 17, k = 12, d_{var} = 1 \)

![Traffic by TbAS Mechanisms]

(a) System-wide traffic

![Traffic per Peer by TbAS Mechanisms]

(b) Average traffic per active peer

Figure E.19: System-wide token-based accounting traffic breakdown for \( t = 17, k = 12, d_{var} = 1 \)

![Traffic Ratios of TbAS Mechanisms Incl. Serv. Traffic]

(a) Traffic ratios including service traffic

![Traffic Ratios of TbAS Mechanisms w/o Serv. Traffic]

(b) Traffic ratios of token-based accounting

Figure E.20: System-wide traffic ratios for \( t = 17, k = 12, d_{var} = 1 \)
Traffic Breakdown for $t = 17$, $k = 8$, $d_{\text{var}} = 1$

Figure E.21: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 8$, $d_{\text{var}} = 1$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.22: System-wide traffic ratios for $t = 17$, $k = 8$, $d_{\text{var}} = 1$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting
Traffic Breakdown for $t = 17$, $k = 6$, $d_{\text{var}} = 1$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.23: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 6$, $d_{\text{var}} = 1$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.24: System-wide traffic ratios for $t = 17$, $k = 6$, $d_{\text{var}} = 1$

E.2.2.3 Traffic Breakdown by Churn Factor
Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 0.5$ (Without Lookups)

Figure E.25: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 0.5$

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.26: System-wide traffic ratios for $t = 17$, $k = 10$, $d_{var} = 0.5$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting
Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 0.33$ (Without Lookups)

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.27: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 0.33$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.28: System-wide traffic ratios for $t = 17$, $k = 10$, $d_{var} = 0.33$
Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 0.25$ (Without Lookups)

(a) System-wide traffic

(b) Average traffic per active peer

Figure E.29: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 0.25$

(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure E.30: System-wide traffic ratios for $t = 17$, $k = 10$, $d_{var} = 0.25$
E.2.3 Maintenance Traffic Breakdown

E.2.3.1 Maintenance Traffic Breakdown by Quorum Size

Maintenance Traffic Breakdown for $t = 17, k = 10, d_{\text{var}} = 1$

![System-wide AHS Maintenance Traffic by Mech., $t=17, k=10, d_{\text{var}}=1$](image1)

(a) Maintenance traffic breakdown

![AHS Maintenance Traffic per Peer by Mech., $t=17, k=10, d_{\text{var}}=1$](image2)

(b) Per peer maintenance traffic breakdown

Figure E.31: Maintenance traffic breakdown for $t = 17, k = 10, d_{\text{var}} = 1$

Maintenance Traffic Breakdown for $t = 14, k = 10, d_{\text{var}} = 1$

![System-wide AHS Maintenance Traffic by Mech., $t=14, k=10, d_{\text{var}}=1$](image3)

(a) Maintenance traffic breakdown

![AHS Maintenance Traffic per Peer by Mech., $t=14, k=10, d_{\text{var}}=1$](image4)

(b) Per peer maintenance traffic breakdown

Figure E.32: Maintenance traffic breakdown for $t = 10, k = 10, d_{\text{var}} = 1$
Maintenance Traffic Breakdown for $t = 10, k = 10, d_{var} = 1$

![System-wide AHS Maintenance Traffic by Mech., $t=10$, $k=10$, $d_{var}=1$](image1)

(a) Maintenance traffic breakdown

![AHS Maintenance Traffic per Peer by Mech., $t=10$, $k=10$, $d_{var}=1$](image2)

(b) Per peer maintenance traffic breakdown

Figure E.33: Maintenance traffic breakdown for $t = 7, k = 10, d_{var} = 1$

Maintenance Traffic Breakdown for $t = 7, k = 10, d_{var} = 1$

![System-wide AHS Maintenance Traffic by Mech., $t=7$, $k=10$, $d_{var}=1$](image3)

(a) Maintenance traffic breakdown

![AHS Maintenance Traffic per Peer by Mech., $t=7$, $k=10$, $d_{var}=1$](image4)

(b) Per peer maintenance traffic breakdown

Figure E.34: Maintenance traffic breakdown for $t = 7, k = 10, d_{var} = 1$
E.2.3.2 Maintenance Traffic Breakdown by Account Holder Set Size

Maintenance Traffic Breakdown for \( t = 17, k = 12, d_{\text{var}} = 1 \)

(a) Maintenance traffic breakdown

(b) Per peer maintenance traffic breakdown

Figure E.35: Maintenance traffic breakdown for \( t = 17, k = 12, d_{\text{var}} = 1 \)

Maintenance Traffic Breakdown for \( t = 17, k = 8, d_{\text{var}} = 1 \)

(a) Maintenance traffic breakdown

(b) Per peer maintenance traffic breakdown

Figure E.36: Maintenance traffic breakdown for \( t = 17, k = 8, d_{\text{var}} = 1 \)
Maintenance Traffic Breakdown for $t = 17$, $k = 6$, $d_{var} = 1$

(a) Maintenance traffic breakdown

Figure E.37: Maintenance traffic breakdown for $t = 17$, $k = 6$, $d_{var} = 1$
E.2.3.3 Maintenance Traffic Breakdown by Churn Factor

Maintenance Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 0.5$

![Diagram](image1.png)

(a) Maintenance traffic breakdown
(b) Per peer maintenance traffic breakdown

Figure E.38: Maintenance traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 0.5$

Maintenance Traffic Breakdown for $t = 17$, $k = 10$, $d_{var} = 0.33$

![Diagram](image2.png)

(a) Maintenance traffic breakdown
(b) Per peer maintenance traffic breakdown

Figure E.39: Maintenance traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 0.33$
**Maintenance Traffic Breakdown for** $t = 17, k = 10, d_{var} = 0.25$

![System-wide AHS Maintenance Traffic by Mech., $t=17, k=10, d_{var}=0.25$](image1)

![AHS Maintenance Traffic per Peer by Mech., $t=17, k=10, d_{var}=0.25$](image2)

(a) **Maintenance traffic breakdown**

(b) **Per peer maintenance traffic breakdown**

Figure E.40: Maintenance traffic breakdown for $t = 17, k = 10, d_{var} = 0.25$
E.2.4 Peers’ Upload Queue Length

E.2.4.1 Peers’ Upload Queue Length by Quorum Size

Peers’ Upload Queue Length for $t = 7$, $k = 10$, $d_{var} = 1$

Figure E.41: Peers’ upload queue length distribution for $t = 7$, $k = 10$, $d_{var} = 1$

(a) Proportions of upload queue lengths, unfocused (b) Proportions of upload queue lengths, focused to lower 2% (c) Proportions of upload queue lengths, focused to lower 0.3%
Figure E.42: High peer loads by simulation time for $t = 7$, $k = 10$, $d_{var} = 1$

Peers’ Upload Queue Length for $t = 10$, $k = 10$, $d_{var} = 1$

Figure E.43: Peers’ upload queue length distribution for $t = 10$, $k = 10$, $d_{var} = 1$
Figure E.44: High peer loads by simulation time for $t = 10$, $k = 10$, $d_{var} = 1$

Peers’ Upload Queue Length for $t = 14$, $k = 10$, $d_{var} = 1$

Figure E.45: Peers’ upload queue length distribution for $t = 14$, $k = 10$, $d_{var} = 1$
Peers’ Upload Queue Length for $t = 17, k = 10, d_{var} = 1$

Figure E.46: High peer loads by simulation time for $t = 14, k = 10, d_{var} = 1$

(a) Proportions of upload queue lengths, unfocused

(b) Proportions of upload queue lengths, focused to lower 2%

(c) Proportions of upload queue lengths, focused to lower 0.25%

Figure E.47: Peers’ upload queue length distribution for $t = 17, k = 10, d_{var} = 1$
Figure E.48: High peer loads by simulation time for $t = 17$, $k = 10$, $d_{var} = 1$
**E.2.4.2 Peers’ Upload Queue Length by Account Holder Set Size**

**Peers’ Upload Queue Length for** $t = 17$, $k = 6$, $d_{var} = 1$

(a) Proportions of upload queue lengths, unfocused

(b) Proportions of upload queue lengths, focused to lower 2%

(c) Proportions of upload queue lengths, focused to lower 0.3%

Figure E.49: Peers’ upload queue length distribution for $t = 17$, $k = 6$, $d_{var} = 1$
Figure E.50: High peer loads by simulation time for $t = 17, k = 6, d_{var} = 1$

Peers’ Upload Queue Length for $t = 17, k = 8, d_{var} = 1$

(a) Proportions of upload queue lengths, unfocused
(b) Proportions of upload queue lengths, focused to lower 2%

(c) Proportions of upload queue lengths, focused to lower 0.3%

Figure E.51: Peers’ upload queue length distribution for $t = 17, k = 8, d_{var} = 1$
Peers’ Upload Queue Length for $t = 17, k = 12, d_{\text{var}} = 1$

(a) Proportions of upload queue lengths, unfocused (b) Proportions of upload queue lengths, focused to lower 2%

(c) Proportions of upload queue lengths, focused to lower 0.25%

Figure E.53: Peers’ upload queue length distribution for $t = 17, k = 12, d_{\text{var}} = 1$
Figure E.54: High peer loads by simulation time for $t = 17$, $k = 12$, $d_{var} = 1$
E.3 Simulation of Key Management Traffic

E.3.1 Update And Self Initialisation Traffic by Message Group

Figure E.55: Simulation results for $L_1 = 99, 99\%$
Update And Self Initialisation Traffic by Message Group, $T=100$, $t=15$

Overlay Maintenance
Discovery
Update
Recovery

Traffic in kBytes
Peer [node ID]

(a) Traffic by message group, $T=100$, $t=16$

Update And Self Initialisation Traffic by Message Group, $T=500$, $t=17$

Overlay Maintenance
Discovery
Update
Recovery

Traffic in kBytes
Peer [node ID]

(b) Traffic by message group, $T=500$, $t=17$

Update And Self Initialisation Traffic by Message Group, $T=1000$, $t=17$

Overlay Maintenance
Discovery
Update
Recovery

Traffic in kBytes
Peer [node ID]

(c) Traffic by message group, $T=1000$, $t=17$

Update And Self Initialisation Traffic by Message Group, $T=2000$, $t=17$

Overlay Maintenance
Discovery
Update
Recovery

Traffic in kBytes
Peer [node ID]

(d) Traffic by message group, $T=2000$, $t=17$

Figure E.56: Simulation results for $L_2 = 99, 999\%$

Appendix E. Supplementary Simulation Results
List of Algorithms

4.1 Information Storage and Retrieval for Aggregation Accounts ......................... 73
4.2 Check for Aggregation Account Existence .................................................. 255
4.3 Aggregation Account Creation ................................................................. 255
4.4 Aggregation Account Initialization ............................................................ 256
4.5 Detection of Current Account Holder Set Size ........................................... 258
4.6 Detection of correct account holder set position ....................................... 259
4.7 Account Holder Set Locking ................................................................. 261
4.8 Detect New Account Position ................................................................. 262
4.9 Move Account to New Position ................................................................. 263
4.10 Graceful Aggregation Account Handover .................................................... 265
4.11 Post Account Information at an Account Holder Set .................................. 266
4.12 Aggregation Account Consistency Algorithm ............................................ 267
4.13 Locate Trusted Peer ................................................................................. 268
4.14 Secure Token Aggregation ........................................................................ 269
List of Figures

1.1 Phases of commercial transactions (Sch04) ................................................. 4
1.2 Functional requirements for commercial and community p2p applications .......... 4
2.1 Components of CDNs (cf. (BEH+01)) ......................................................... 13
2.2 CDN accounting peering architecture (cf. (BEH+01)) ................................ 14
2.3 Classification of accounting systems for autonomous distributed systems ....... 20
2.4 Concept of “Karma” and “Bucking Free-riders” (cf. (VCS03, ABOS03), ........... 23
2.5 Classical reconstruction of a secret .............................................................. 33
2.6 Threshold signature without revealing the secret .......................................... 37
2.7 Graphical representation of the update phase .............................................. 42
3.1 Basic token-based accounting concept ....................................................... 53
3.2 Solution space for token issuing responsibility ........................................... 53
3.3 Preliminary token-based accounting framework with threshold cryptography .... 54
3.4 Extended token-based accounting framework ............................................. 55
3.5 Complete token-based accounting scheme framework .................................. 56
3.6 Token-based accounting scheme – architecture overview ............................. 58
3.7 Token structure ......................................................................................... 60
3.8 Basic transaction procedure ..................................................................... 61
3.9 Token aggregation ..................................................................................... 62
3.10 Detection of double spending .................................................................. 63
3.11 Trustworthy transaction procedure ........................................................... 65
4.1 Allowed account holder set positions ......................................................... 71
4.2 Aggregation account position and routing to an account .............................. 72
4.3 Aggregation account creation algorithm ...................................................... 78
4.4 Trusted peer overlay and system overlay ..................................................... 88
4.5 Secure token aggregation protocol - part 1 ............................................... 92
4.6 Secure token aggregation protocol - part 2 ............................................... 93
4.7 Secure token aggregation protocol - part 3 ............................................... 94
4.8 Tree-based system key update distribution strategy .................................... 105
4.9 Limited update and self-initialisation distribution strategy ......................... 107
5.1 Required quorum size according to equation 5.1 ....................................... 118
5.2 Account holder set size simulation results for \( d_{var} = 0.5, p_{gh} = 0.9 \) ........ 120
5.3 PeerFactSim.KOM: Layered architecture .................................................. 126
5.4 TokenAccounterRole implementation ....................................................... 128
5.5 TrustedPeerRole implementation ............................................................... 130
5.6 AccountHolderRole implementation .......................................................... 131
5.7 TokenUserRole implementation .................................................................. 135
5.8 System-wide traffic for \( t = 17, k = 10, d_{var} = 1 \) ..................................... 142
5.9 System-wide token-based accounting traffic breakdown for \( t = 17, k = 10, d_{var} = 1 \) .... 144
5.10 System-wide traffic ratios for \( t = 17, k = 10, d_{var} = 1 \) ............................... 144
5.11 Token statistics, transactions, token aggregations for \( t = 17, k = 10, d_{var} = 1 \) .... 145
5.12 Maintenance traffic breakdown for \( t = 17, k = 10, d_{var} = 1 \) ....................... 146
5.13 Upload queue lengths percentage for the base experiment ....................... 147
5.14 High peer loads by simulation time .......................................................... 147
5.15 System-wide aggregation traffic by quorum size ...................................... 148
5.16 System-wide lookup traffic by quorum size ........................................... 149
5.17 Variable traffic by quorum size ............................................................. 150
5.18 Fixed traffic by quorum size ................................................................. 150
5.19 Peers' upload queue length proportions by quorum size, \( k = 10 \) .... 151
5.20 Total token-based accounting traffic by AHS size ................................. 152
5.21 Total traffic per peer and per transaction by AHS size ......................... 152
5.22 Reasons for volatile aggregation traffic for AHS size \( k = 6 \) .................. 153
5.23 Peer's upload queue length proportions by AHS size, \( t = 17 \) .......... 154
5.24 Payment traffic by AHS size ................................................................. 154
5.25 Payment traffic per peer and per transaction ........................................ 155
5.26 Token aggregation traffic by AHS size .................................................. 156
5.27 Token aggregation traffic per peer and per transaction by AHS Size ........ 157
5.28 Account holder set maintenance traffic by AHS size .............................. 158
5.29 Account holder set check size traffic by AHS size ................................. 159
5.30 Account holder set check position traffic by AHS size ........................... 160
5.31 Lock aggregation account traffic by AHS size ....................................... 161
5.32 Aggregation account consistency traffic by AHS size ............................ 162
5.33 Aggregation account handover traffic by AHS size ............................... 163
5.34 Account movement traffic by AHS size ................................................ 164
5.35 Lookup traffic by AHS size ................................................................. 165
5.36 Variable traffic by AHS size ................................................................. 166
5.37 Fixed traffic by AHS size ..................................................................... 166
5.38 Total token-based accounting traffic by churn factor \( d_{var} \) ................. 168
5.39 Total token-based accounting traffic without lookup traffic by churn factor \( d_{var} \) ................................................................. 169
5.40 Payment traffic by churn factor ............................................................ 170
5.41 Token aggregation traffic by churn factor .............................................. 171
5.42 Account holder set maintenance by churn factor .................................. 172
5.43 Account holder set check size traffic by churn factor ............................ 173
5.44 Account holder set check position traffic by churn factor ...................... 174
5.45 Locking aggregation account traffic by churn factor ............................ 175
5.46 Aggregation account consistency traffic by churn factor ....................... 176
5.47 Aggregation account handover traffic by churn factor .......................... 177
5.48 Aggregation account movement traffic by churn factor ....................... 178
5.49 Chord maintenance traffic by churn factor .......................................... 179
5.50 Variable traffic by churn factor ............................................................ 180
5.51 Fixed traffic by churn factor ............................................................... 180
5.52 Token-based accounting traffic and AHS maintenance traffic per peer by system size ................................................................. 182
5.53 Payment and token aggregation traffic per transaction by system size ........ 183
5.54 URSA update simulation implementation concept .................................. 187
5.55 Durations for update and self initialisation .......................................... 189
5.56 Update and self initialisation traffic by message type ............................ 190
5.57 Traffic statistics for the update and self initialisation strategy .................. 191
5.58 Karma transaction procedure (cf. (VCS03)) ......................................... 194
6.1 Required quorum size according to Equation 5.1 ................................... 198
6.2 Generated overhead traffic ................................................................. 204
6.3 Simulation results for symmetric aggregation function ............................ 208
6.4 Number of transactions and file sharing behaviour ............................... 209
6.5 Development of own tokens ............................................................... 209
C.1 Blakely's secret sharing (from (RSA04)) ................................................ 250
E.1 System-wide traffic for \( t = 17, k = 10, d_{var} = 1 \) .................................. 276
<p>| E.2 | System-wide traffic for $t = 14, k = 10, d_{var} = 1$ | 276 |
| E.3 | System-wide traffic for $t = 10, k = 10, d_{var} = 1$ | 277 |
| E.4 | System-wide traffic for $t = 7, k = 10, d_{var} = 1$ | 277 |
| E.5 | System-wide traffic for $t = 17, k = 12, d_{var} = 1$ | 278 |
| E.6 | System-wide traffic for $t = 17, k = 8, d_{var} = 1$ | 278 |
| E.7 | System-wide traffic for $t = 17, k = 6, d_{var} = 1$ | 279 |
| E.8 | System-wide traffic for $t = 17, k = 10, d_{var} = 0.5$ | 279 |
| E.9 | System-wide traffic for $t = 17, k = 10, d_{var} = 0.33$ | 280 |
| E.10 | System-wide traffic for $t = 17, k = 10, d_{var} = 0.25$ | 280 |
| E.11 | System-wide token-based accounting traffic breakdown for $t = 17, k = 10, d_{var} = 1$ | 281 |
| E.12 | System-wide traffic ratios for $t = 17, k = 10, d_{var} = 1$ | 281 |
| E.13 | System-wide token-based accounting traffic breakdown for $t = 14, k = 10, d_{var} = 1$ | 282 |
| E.14 | System-wide traffic ratios for $t = 14, k = 10, d_{var} = 1$ | 282 |
| E.15 | System-wide token-based accounting traffic breakdown for $t = 10, k = 10, d_{var} = 1$ | 283 |
| E.16 | System-wide traffic ratios for $t = 10, k = 10, d_{var} = 1$ | 283 |
| E.17 | System-wide token-based accounting traffic breakdown for $t = 7, k = 10, d_{var} = 1$ | 284 |
| E.18 | System-wide traffic ratios for $t = 7, k = 10, d_{var} = 1$ | 284 |
| E.19 | System-wide token-based accounting traffic breakdown for $t = 17, k = 12, d_{var} = 1$ | 285 |
| E.20 | System-wide traffic ratios for $t = 17, k = 12, d_{var} = 1$ | 285 |
| E.21 | System-wide token-based accounting traffic breakdown for $t = 17, k = 8, d_{var} = 1$ | 286 |
| E.22 | System-wide traffic ratios for $t = 17, k = 8, d_{var} = 1$ | 286 |
| E.23 | System-wide token-based accounting traffic breakdown for $t = 17, k = 6, d_{var} = 1$ | 287 |
| E.24 | System-wide traffic ratios for $t = 17, k = 6, d_{var} = 1$ | 287 |
| E.25 | System-wide token-based accounting traffic breakdown for $t = 17, k = 10, d_{var} = 0.5$ | 288 |
| E.26 | System-wide traffic ratios for $t = 17, k = 10, d_{var} = 0.5$ | 288 |
| E.27 | System-wide token-based accounting traffic breakdown for $t = 17, k = 10, d_{var} = 0.33$ | 289 |
| E.28 | System-wide traffic ratios for $t = 17, k = 10, d_{var} = 0.33$ | 289 |
| E.29 | System-wide token-based accounting traffic breakdown for $t = 17, k = 10, d_{var} = 0.25$ | 290 |
| E.30 | System-wide traffic ratios for $t = 17, k = 10, d_{var} = 0.25$ | 290 |
| E.31 | Maintenance traffic breakdown for $t = 17, k = 10, d_{var} = 1$ | 291 |
| E.32 | Maintenance traffic breakdown for $t = 10, k = 10, d_{var} = 1$ | 291 |
| E.33 | Maintenance traffic breakdown for $t = 7, k = 10, d_{var} = 1$ | 292 |
| E.34 | Maintenance traffic breakdown for $t = 7, k = 10, d_{var} = 1$ | 292 |
| E.35 | Maintenance traffic breakdown for $t = 17, k = 12, d_{var} = 1$ | 293 |
| E.36 | Maintenance traffic breakdown for $t = 17, k = 8, d_{var} = 1$ | 293 |
| E.37 | Maintenance traffic breakdown for $t = 17, k = 6, d_{var} = 1$ | 294 |
| E.38 | Maintenance traffic breakdown for $t = 17, k = 10, d_{var} = 0.5$ | 295 |
| E.39 | Maintenance traffic breakdown for $t = 17, k = 10, d_{var} = 0.33$ | 295 |
| E.40 | Maintenance traffic breakdown for $t = 17, k = 10, d_{var} = 0.25$ | 296 |
| E.41 | Peers’ upload queue length distribution for $t = 7, k = 10, d_{var} = 1$ | 297 |
| E.42 | High peer loads by simulation time for $t = 7, k = 10, d_{var} = 1$ | 298 |
| E.43 | Peers’ upload queue length distribution for $t = 10, k = 10, d_{var} = 1$ | 298 |
| E.44 | High peer loads by simulation time for $t = 10, k = 10, d_{var} = 1$ | 299 |
| E.45 | Peers’ upload queue length distribution for $t = 14, k = 10, d_{var} = 1$ | 299 |
| E.46 | High peer loads by simulation time for $t = 14, k = 10, d_{var} = 1$ | 300 |
| E.47 | Peers’ upload queue length distribution for $t = 17, k = 10, d_{var} = 1$ | 300 |
| E.48 | High peer loads by simulation time for $t = 17, k = 10, d_{var} = 1$ | 301 |
| E.49 | Peers’ upload queue length distribution for $t = 17, k = 6, d_{var} = 1$ | 302 |
| E.50 | High peer loads by simulation time for $t = 17, k = 6, d_{var} = 1$ | 303 |
| E.51 | Peers’ upload queue length distribution for $t = 17, k = 8, d_{var} = 1$ | 303 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.52</td>
<td>High peer loads by simulation time for $t = 17, k = 8, d_{var} = 1$</td>
<td>304</td>
</tr>
<tr>
<td>E.53</td>
<td>Peers’ upload queue length distribution for $t = 17, k = 12, d_{var} = 1$</td>
<td>304</td>
</tr>
<tr>
<td>E.54</td>
<td>High peer loads by simulation time for $t = 17, k = 12, d_{var} = 1$</td>
<td>305</td>
</tr>
<tr>
<td>E.55</td>
<td>Simulation results for $L_1 = 99.99%$</td>
<td>306</td>
</tr>
<tr>
<td>E.56</td>
<td>Simulation results for $L_2 = 99.999%$</td>
<td>307</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Overview Existing P2P Accounting Mechanisms .................................................. 29
4.1 Potential Attacks Involving the Account Holder Set ............................................... 69
4.2 Required Hops for Aggregation Account Shift for $Z = 10000$ ............................... 75
4.3 Account Holder Set Information ............................................................................. 84
4.4 Aggregation Information Stored In Aggregation Accounts ..................................... 84
4.5 Probabilities for Byzantine Consensus Compliant Account Holder Sets for System Size of 1000 Peers ................................................................. 86
4.6 Notation for Analytical Comparison of Share Distribution Strategies ....................... 95
4.7 System Key Update - Maximum Peer Traffic Comparison for $T = 1000$ and $t = 17$ ........ 108
4.8 System Key Update - Complete Traffic Comparison for $T = 1000$ and $t = 17$ ............. 109
5.1 Churn According to (SENB07b): Weibull Distribution Parameters ......................... 119
5.2 Churn According to (BQ04, Dar05): Mixed Log-normal Distribution Parameters .......... 119
5.3 Required Holder Set Sizes For Different Churn Models ........................................... 120
5.4 Simulation Scenario Parameters ............................................................................. 138
5.5 Simulation Message Size Parameters ...................................................................... 139
5.6 Experiments Overview ......................................................................................... 140
5.7 Resulting Quorum Sizes $t$ for System-key Maintenance Simulations ......................... 188
5.8 Experiments for System-key Maintenance .............................................................. 188
E.1 Required Holder Set Sizes for Different Churn Models – Details ............................... 271
E.2 Detailed Simulation Results for Weibull Distribution, $d_{var} = 1$, $p_{log-off} = 0.9$ ........ 272
E.3 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.5$, $p_{log-off} = 0.9$ .......... 272
E.4 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.333$, $p_{log-off} = 0.9$ ......... 272
E.5 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.25$, $p_{log-off} = 0.9$ ........... 272
E.6 Detailed Simulation Results for Weibull Distribution, $d_{var} = 1$, $p_{log-off} = 0.5$ ............. 273
E.7 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.5$, $p_{log-off} = 0.5$ ............. 273
E.8 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.333$, $p_{log-off} = 0.5$ ............ 273
E.9 Detailed Simulation Results for Weibull Distribution, $d_{var} = 0.25$, $p_{log-off} = 0.5$ ............ 273
E.10 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 1$, $p_{log-off} = 0.9$ .. 274
E.11 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.5$, $p_{log-off} = 0.9$ . 274
E.12 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.333$, $p_{log-off} = 0.9$ 274
E.13 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.25$, $p_{log-off} = 0.9$ 274
E.14 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 1$, $p_{log-off} = 0.5$ . 274
E.15 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.5$, $p_{log-off} = 0.5$ . 275
E.16 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.333$, $p_{log-off} = 0.5$ 275
E.17 Detailed Simulation Results for Mixed Log-Normal Distribution, $d_{var} = 0.25$, $p_{log-off} = 0.5$ 275