Liquid Democracy

Rheinisch-Westfälische Technische Hochschule Aachen
Informatik 4 ComSys

Diploma Thesis

Angel Tchorbadjiiski

Advisors:

Dipl. Inf.	Jó Ágila Bitsch
Dr. rer. nat.	Tobias Heer
Prof. Dr.-Ing.	Klaus Wehrle
PD Dr. rer. nat.	Walter Unger

Registration date:	10. November	2011
Submission Date:	30. March	2012

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Aachen, den 30. März 2012

Abstract

In this thesis I present an approach for implementing the liquid democracy concept in a secure, anonymous and publicly verifiable manner via the Internet. Liquid democracy is gaining traction in the context of open government and civil participation, as it allows vote delegation and thereby presents a hybrid form of direct and representative democracy.

Combining RSA public-key cryptography with hash-based data structures, this system allows the acquisition of anonymous voting tokens, their delegation while still allowing outvoting, and secret, anonymous voting. Voters can verify their own vote, while the public can check the consistency of the overall voting results. The solution thereby enables the wider use of liquid democracy to further enhance civil participation in the government process, while having a high resilience against vote manipulation, i.e. manipulation will be discovered with a probability of higher than 99% with less than 1% of the voters verifying their vote.

Zusammenfassung

Diese Diplomarbeit stellt einen Ansatz zur Implementierung des Liquid-Democracy-Konzepts vor, um dieses auf eine sichere, anonyme und öffentlich überprüfbare Weise über das Internet verwenden zu können. Das Liquid-Democray-Konzept gewinnt an Bedeutung im Kontext von Open Government und Bürgerbeteiligung am Regierungsprozess, da dieses eine Stimmendelegierung ermöglicht und damit eine Mischform zwischen direkte und representative Demokratie darstellt.

Über die Kombination von asymmetrischer RSA Kryptographie und hash-basierten Datenstrukturen erlaubt das System den Bezug anonymer Wahltokens, ihre Delegierung mit vorbehaltenem Überstimmungsrecht und eine sichere, anonyme Stimmabgabe. Jeder Wähler ist in der Lage, seine eigene Stimme zu überprüfen, während die Allgemeinheit die Konsistenz des kompletten Wahlergebnisses verifizieren kann. Die hier vorgestelle Lösung ermöglicht die verbreitete Nutzung des Liquid-Democracy-Konzepts um die öffentliche Teilname an dem Regierungsprozess weiter zu erhöhen. Diese hat eine hohe Resistenz gegen Stimmenmanipulationen — diese werden mit einer Wahrscheinlichkeit $> 99\%$ erkannt, selbst bei einer Prüferanzahl von $\approx 1\%$ der Wahlbeteiligten.

Acknowledgments

First of all I want to thank my beloved family for all their faith in me and the support, advice and understanding they give me all the time. Next I'd like to thank especially my supervisor Jó Ágila Bitsch Link for devising the topic of my diploma thesis, all his support and kind guidance, all the afternoons spent in discussions and all the technical advice he gave me – I learned a lot new things during the time. Another thank you goes to Tobias Heer for all the enhancements he suggested and the time he spend in discussing the topic together with Jó and myself.

A big thank you goes to my good friend Jivko Vantchev for all the time he spend reading my not so easy to read thesis and all the corrections he suggested. Furthermore I'm grateful to all the friends and acquaintances, who showed interest in my thesis topic and discussed it with me – this helped me a lot to find design inaccuracies and be more precise by describing all the details.

Last but not least, I'd like to thank Professor Klaus Wehrle for the possibility to write my thesis at his chair and all the fun I had during this time.

Contents

1 Introduction

Online services introduce on the one hand a possibility to carry out everyday life duties in a comfortable and fast way. Besides that they are accessible 24/7 and from any place of the world which additionally grants their users a great flexibility. This enables the implementation of electronic voting systems allowing a democracy concept, where the population can directly influence the government process of its county. Though such systems should be designed with caution, as user privacy and information secrecy problems can arise.

1.1 Motivation

In this thesis we introduce the `Liquid Democracy` concept. It represents a mixture of both direct and indirect democracy for a decision-making process and allows every participant to decide how involved in this process he wants to be. For every election taking place it is possible to either take part directly or delegate the own voting rights to a representative/expert. This way the voters are not limited to taking one decision for legislative period as opposed to indirect (representative) democracy, but are able to actively and continuously take part in the decision-making process. This concept furthermore integrates ways of collaborative decision making.

Currently existing implementations of `Liquid Democracy` focus mainly on collaborative decision making and pretty much neglect aspects like secrecy and anonymity of the voting process. They save enough information about the participating users making it possible for a system operator to find out how each of them voted in a given decision-making round.

This thesis focuses on the design and implementation of a system, which allows secure and anonymous voting in such a way that it is not possible, even for the system operator, to find out the identity of a voter or to prevent certain voters (for example minority groups) from casting a ballot. Anonymity is guaranteed through the blind signature of a user generated token at the voting register, which turns this token into a voting credential. As none of these credentials are readable on the server side, it is impossible to connect the user's identity with the voting decision even if both the databases of voting register and voting computer are combined. Public-key cryptography is used for securing the data transmission, peer verification and verifiable voting token retrieval.

Only the owner of given voting credentials is able to verify with confidence, what these were used for. This combined with the fact that the voting credentials and results are published online by the voting server, makes the overall results verifiable on a probabilistic basics. As in normal paper based elections, a possibility of cheating still exists – a given subset of votes could be manipulated by the voting computer. The detection rate in this case depends on the number of voters checking the correctness of their own votes and is high (over 80%) even for a small amount (about 10%) of checkers. Vote selling and coercion can't fully be eliminated. A bystander present at the moment of voting can observe the voter's choice. Furthermore as the results are verifiable by everyone, extortion or buying of voting credentials and verification of the vote validity after the election is possible.

The reliability of a voting system against attacks is an important aspect to be considered. To minimize/mitigate the risk of (distributed) Denial-of-Service attacks it is possible to use a distributed network of *Voting Computer* and *Voting Register* servers. In such a scenario no user supplied information needs to be shared between the system instances, but a strict time synchronisation is needed to prevent cheating.

There are three possible scenarios, how a voting round using our system can be carried out:

1. In kiosk mode at a supervised location (voting booths, etc.) only.
2. Using own device (desktop PC, laptop, smartphone, etc.) over the Internet only.
3. A combination of (1) and (2).

The system is applicable using ordinary voting booths, but allows also remote voting over the Internet. This grants the voters a great flexibility without sacrificing the security and anonymity. Nobody has to go to a predefined voting booth but everyone can vote while on the go, e.g. on a business trip to a destination thousands of kilometers away.

1.2 Thesis Structure

Chapter 2 contains the background knowledge needed for understanding the concept of this work. It gives an overview of direct and indirect democracy and uses them to define the `Liquid Democracy` concept. As the implementation makes heavy use of cryptography, a whole subsection introduces some basic building blocks and then shows their usage in complex structures like hash chains and Merkle trees. One aspect of securing the voting process bases on public-key cryptography and blind signatures, they are explained in detail to ease the understanding of system design.

Chapter 3 presents related implementations and introduces both their strengths and weaknesses. Chapter 4 contains the detailed design of the system. It gives examples of possible attack vectors and what design decisions are taken to circumvent the corresponding security related problems. A simple python implementation of the concept is present in Chapter 5. It is thought of as Proof-of-Concept code and includes only the basic features needed to show that the design works as desired. Chapter 6 describes possible problems in the system, gives an estimate on the scalability and performance and compares it to the implementations described in Chapter 3. This is meant to give you a good understanding of the strengths and weaknesses of my design and implementation of the `Liquid Democracy` concept. Chapter 7 summarises the content of my theses and gives ideas of possible future work in this field.

2

Background

This chapter gives an overview of all topics needed for understanding the rest of my thesis. First I describe the democracy concepts leading to the existence of `LD`, then present the voting regulations in Germany. Next I briefly discuss electronic voting and give then a short overview of cryptographic basics needed to secure my design presented in Chapter 4. Finally, a detailed introduction to the topics of public-key cryptography, connection encryption and anonymity conclude the this chapter.

2.1 Democracy Concepts

Democracy is a government form originating from ancient Greece. In the Greek language it is a compound of the two words *δῆμος* `(demos)` meaning people and *κράτος* `(kratos)` meaning power/strength which can be translated as `rule of the people`. Nowadays the three types `direct`, `indirect` and `semi-direct` or Liquid Democracy (**LD**) can be distinguished.

2.1.1 Direct Democracy

Direct democracy, also referred to as `pure democracy`, has evolved in Athens around 500BC. It is a type of government where every citizen is allowed to directly take part in the decision-making process implying their active involvement. This required a possibility for all of them to take part in direct voting procedures. Because of this fact it is not trivial to support such a concept even for a small modern country like the Principality of Monaco with its 36 thousand citizens.

The most important advantage of direct democracy over other systems is that people directly represent their own opinion on every decision to be taken. Furthermore it also limits the influence of small, but powerful groups (lobbying), which lies in the fact, that positive sum proposals [1] are more likely to be accepted, as they cover the interest of the largest voters group. The most important disadvantage is the fact, that it is complicated and costly to support this concept through standard voting procedures. In ancient Athens the 30 to 60 thousand citizens were the upper limit, which modern countries exceed even within small towns. The government process is hence inflexible and slow as every decision has to be taken from a large amount of people.

The Swiss Confederation is an example of a modern country using the direct democracy concept on a large-scale. Despite the fact that there is an indirect government layer, Swiss citizens are able to directly influence the government process of their country through `tools` like recalls, referendums and initiatives.

2.1.2 Indirect Democracy

In indirect or representative democracy voters elect a group of individuals for a legislative period[2], which then rules on behalf of the voters and represents their interests. This makes elections the only point in time, when citizens influence the governance course directly.

Compared to direct democracy, the representative form is more flexible. Elections are held only at regular intervals of typically 4 to 5 years making this form also cost efficient. The government process is much simpler and persistent as decisions are taken from a small group of people. The individuals forming the government ideally are highly qualified, have a good overview on current topics and excel in decision making, which isn't expected from the average voter. Reality tends to be less optimal. A more important disadvantage is that representatives are not bound to promises given in their election campaigns or views they claimed to have. While in government, they don't need to answer to the people who have elected them.

2.1.3 Liquid Democracy

A lot of concepts try to combine the strengths of direct and indirect democracy to improve government, however this thesis considers only `LD`. The Adhocracy developers define it as "a collective term for different approaches to making democracy more liquid, more transparent and more flexible. What all these approaches have in common is the concept of delegating your vote"[5]. This means that you as a voter can directly vote on every topic under consideration, but also have the possibility to delegate (proxy) your own vote to a `trusted entity`[3]. This delegation process reduces the number of people taking part in the decision-making process, but weights their voting power according to the number of delegations they received, as shown in Figure 2.1. Forwarding is possible in many different ways. In the left part of the

[1]A positive-sum (win-win) game is a "game which is designed in a way that all participants can profit from it in one way or the other"[6]. In this sense a positive-sum proposal means that the sum of all benefits exceeds the sum of all costs when measured across the society as a whole.

[2]The time span between two elections for an institution, for example country government.

[3]A better informed and qualified person (an expert) on a given topic or a political party of choice.

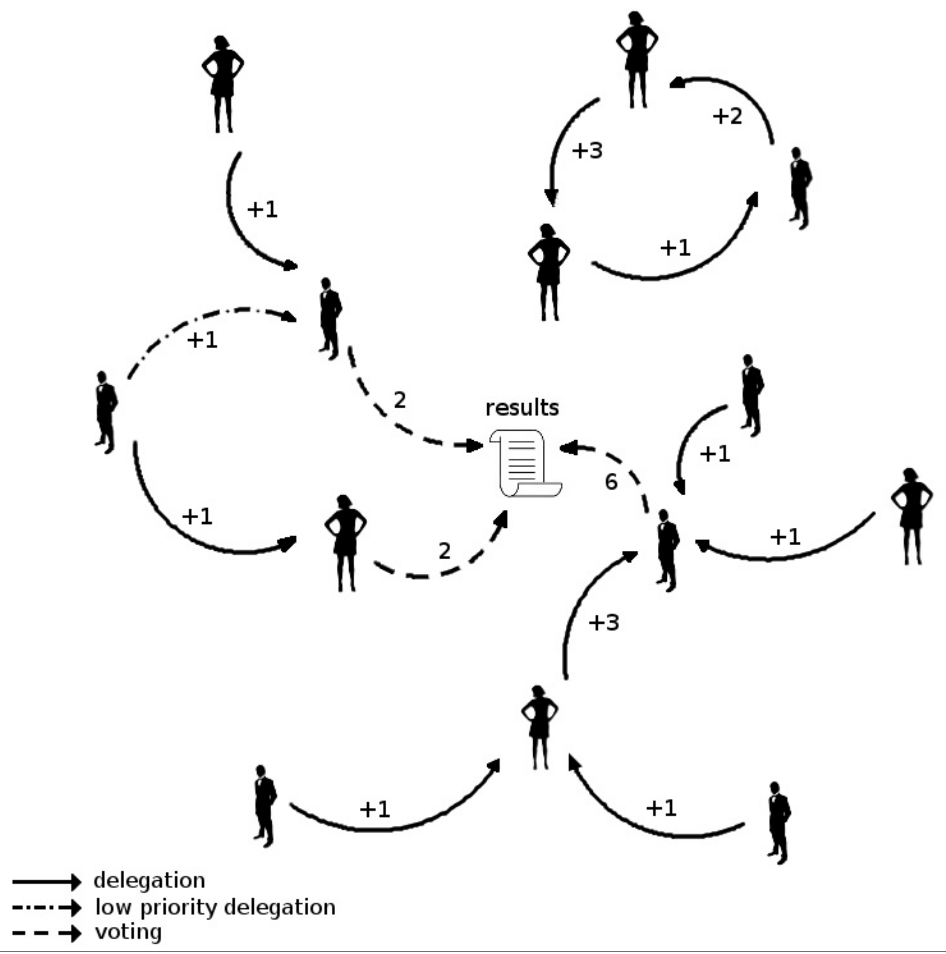


Figure 2.1 The `Liquid Democracy` concept allows single or multiple delegation. Furthermore priorities and delegation of delegated votes are allowed.

figure we can see the same voter delegating to multiple recipients, where one of the delegations has a low priority and is only accounted, if the high priority delegation is not used. The upper right corner exhibits a circular forwarding, so eventually none of the delegators submits a ballot. In the bottom two voters delegate to a trustee, which then further delegates the delegated votes to another proxy (delegation chain).

A key feature of proxy voting, mentioned in most of the concepts, is its reversibility – you as a voter can revisit the delegation, take it back and vote for yourself. This action is effective for all ballots that are not already closed at the moment of revocation.

`LD` provides great flexibility. You do not have to decide yourself on the program of a political party, which only suits some aspects of your opinion. You have the freedom to express your views on every topic to be voted on or possibly delegate to someone you trust and hold best qualified to decide on an issue. As a result, all voters can choose between direct and indirect democracy creating a hybrid government form suiting their own views.

2.2 Voting Regulations in Germany

The government form of Germany is a federal parliamentary republic. The Constitution (Grundgesetz) defines the basic rights of German citizens and the judical and political basic order. "Alle Staatsgewalt geht vom Volke aus. Sie wird vom Volke in Wahlen und Abstimmungen und durch besondere Organe der Gesetzgebung, der vollziehenden Gewalt und der Rechtsprechung ausgeübt. (All state authority is derived from the people. It shall be exercised by the people through elections and other votes and through specific legislative, executive, and judicial bodies.)"[22, Article 20, Paragraph 2] This describes the fact, that in the Federal Republic of Germany the citizens govern the country indirectly through the parliament (Bundestag). Its representatives are appointed through elections and the requirements on them are the following:

"Die Abgeordneten des Deutschen Bundestages werden in allgemeiner, unmittelbarer, freier, gleicher und geheimer Wahl gewählt (Members of the German Bundestag shall be elected in general, direct, free, equal, and secret elections.) [...]"[22, Article 38, Paragraph 1]

The five election characteristics are described below:

- **Free** – Nobody can be coerced to cast a certain vote
- **Equal** – The votes of all participants have equal weight.
- **Secret** – Nobody knows what other people voted for.
- **General** – An election Involves all constituencies in the selection of the candidates.
- **Direct** – Ballots are cast for the representative entity (person or political party) to be elected.

Although a general description of the voting characteristics appears in the German constitution, no exact definition of the election process is given. The overview of the electoral system and its bodies, the franchise are election process are described in the Federal Electoral Law (BundesWahlGesetz, **BWG**). Section 5 of the BWG is of further interest, as it depicts the three possible ways of casting a ballot:

- paper ballot ([1, Article 34])
- voting machines ([1, Article 35])
- postal ballot ([1, Article 36])

Paper ballot is the default way of voting and covers all of the requirements stated by the German constitution. It is carried out in supervised locations (voting booth) on election day. In recent years we see a trend toward the postal ballot, as in the last elections in 2009 about 21.5%[12, Postal Ballot in 2009] of the Germans used this way to vote. In comparison in 2005 the percentage was only 18.7. Even though this type of voting probably violates both the **secret** and **free** requirements, it is more comfortable and less time consuming for voters to take part in the process from home.

The use of voting machines covered by Article 35 causes still ongoing discussions. On the 3rd of March 2009 the Federal Constitutional Court (Bundesverfassungsgericht, BVerfG) judged against electronic voting machines used in the elections in 2005[7, BVerfG Judgement: Wahlcomputer unconstitutional, 2009], though didn't declare the Article 35 as unconstitutional. The problem was that the voting machines used did not allow every citizen, taking part in the election, to verify the fundamental steps in the voting procedure. According to the courts view, it is easier to manipulate the data available only in electronic form. Compared to that, manipulation of paper ballots is possible[10, Election Manipulation in Dachau (Bayern), 2002], but seen as much more expensive and difficult to achieve. Nevertheless the use of electronic voting has appealing advantages compared to paper ballots, so the next section will introduce electronic voting schemes and their benefits.

2.3 Electronic voting

Electronic voting, also referred to as `e-voting`), is characterised through the use of electronic devices for both the vote casting and counting. Compared to paper based ballots, its advantages are the faster, flexible, cost effective and accessibility suitable ballot procedures. Though currently existing `e-voting` systems suffer a major problem - vote manipulation is not detectable.

Electronic voting can be divided into two types:

1. traditional polling stations using electronic equipment
2. voting over the Internet (a.k.a. `i-voting`)

In case of 1) the voting process is supervised through government representatives or independent authorities, which guarantees the smooth course of the voting procedure. This is not possible in case of 2) as all voters use their own electronic equipment (desktop PC, laptop, mobile phone, etc.) and can even be abroad at the moment of voting. For this reasons vote selling or coercion present another major problem in such systems.

In recent past a large number of electronic voting systems were proven insecure ([20], [25], [21], [32]). This shows how important it is to design the system with main focus on security.

2.4 Cryptographic Basics

This section introduces the cryptographic basics needed to understand the Chapter 4 and 5. The main focus in on data structures created with the help of a hash function and the RSA algorithm for blind signatures. As the RSA encryption algorithm is a well known, it is only summarised shortly.

2.4.1 Hash Function

A cryptographic hash function $h()$ is defined as a "computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values" [30, Menezes, Oorschot, Vanstone]. The resulting hash values represent a compression of a variable length input set to a fixed length output set. As a result collisions arise – different input values x_1 and x_2 generate the same output $y = h(x_1) = h(x_2)$. Furthermore has every cryptographic hash function the two important characteristics `one-way` and `collision-resistant`. The former expresses, that it is `easy` to compute the result $h(x)$ given the binary string value x, but `hard` to compute x for a given hash $h(x)$. In this context the terms `easy` and `hard` refer to the algorithmic complexity and describe if a polynomial time algorithm to solve a given problem exists or not. The latter characteristic implies that finding two values x, y with $x \neq y$ that hash to the same value $h(x) = h(y)$ is a `hard` problem. For more details refer to [30].

As reversing the output of such functions is not efficiently possible with current mathematical approaches and finding a matching input hashing to the same value is `hard`, this class of functions can be used for example as randomisation functions (password hashing), checksumming (data integrity) and digital fingerprints (source verification).

2.4.2 Hash Chains

A hash chain can be generated from a starting value x using the cryptographic hash function $h()$ (see Subsection 2.4.1) in the way presented in Figure 2.2

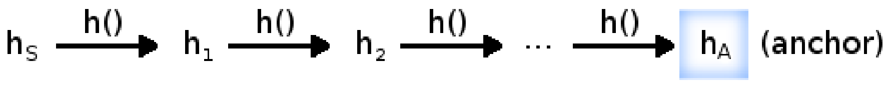

$$\begin{tikzcd} h_S \arrow[r, "h()"] & h_1 \arrow[r, "h()"] & h_2 \arrow[r, "h()"] & \cdots \arrow[r, "h()"] & \boxed{h_A} \text{ (anchor)} \end{tikzcd}$$

Figure 2.2 Hash chains are generated with the help of a cryptographic hash function repeatedly applied to an input value h_S. The input value and the last element h_A, which is called anchor, define the hash chain.

First of all, the supplied input x is fed into the function $h()$. The resulting output h_1 is then fed again as input to the hash function and this process is iterated up to a number of steps n. This way a hash chain $H_n(x) = y$ with starting element x, depth n and output y, also called `anchor`, is generated. Because of the characteristics of the hash function $h()$ there is no way to find out the starting element x only knowing the anchor y, so giving away the value y leaks no information about any of the previous elements in the chain.

A standard application area for hash chains is password protection in insecure environments as described by Lamport in [9].

2.4.3 Hash Trees

A hash tree is build, alike a hash chain, with the help of a hash function $h()$, as shown in Figure 2.3. This data structure is developed by Ralph C. Merkle and is therefore also called Merkle tree ([30, Page 464]).

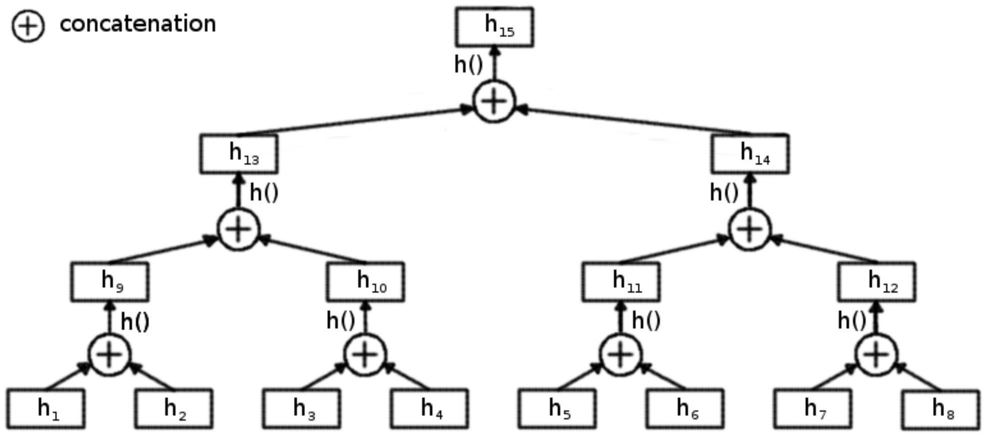


Figure 2.3 A hash tree again requires a cryptographic hash function, but the construction has the form of a binary tree. The string values of the nodes on position 2^i and 2^{i+1} are concatenated and hashed together to generate their parent element in the tree. The most significant element is the tree root.

The generation process starts with a list containing a power of two number of input values, which are hashed to generate the lowest level nodes (`leaves`) of the tree. These are then pairwise concatenated and hashed to generate the upper level of the tree. This way every transition to an higher level has twice as few nodes as the previous and in $\log(n)$ steps the complete tree is build. Constructing a tree with the same root node without knowing the starting values and their arrangement in the lowest level is exactly so hard as reversing the hash function $h()$, as all transitions base on it.

2.5 Public-key Cryptography

In the following two different public-key algorithms will be presented. The first one is the Diffie-Hellman (**DH**) algorithm, which can be used only for establishing a shared encryption key over an insecure communication network. The second is the RSA algorithm, which allows encryption and decryption of information and also can be used for digitally signing messages for the purpose of sender verification.

2.5.1 Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange (**DHKE**) algorithm allows two parties communicating over an insecure network to exchange encryption keys in a secure manner. No third party having access to the messages exchanged is able to efficiently generate the same key. The base of the algorithm is the difficulty of computing a discrete logarithm of a given number in a finite field, however the exponent in the same finite field is a computationally easy operation. The algorithm can be used by two or more parties and is executed on the same way, so only an easy to understand example from Schneier [31, Chapter 22] is presented in the following.

Algorithm steps:

1. Alice and Bob agree over a possibly insecure channel on two large primes n,g (public key parts), where g is primitive mod n, over a possibly insecure channel
2. Alice chooses a random large integer x (private key Alice) and sends Bob $X = g^x \; mod \; n$
3. Bob chooses a random large integer y (private key Bob) and sends Alice $Y = g^y \; mod \; n$
4. Alice computes $k = Y^x \; mod \; n$
5. Bob computes $k' = X^y \; mod \; n$ (shared secret key)

At this point $k = k' = g^{xy} \; mod \; n$ so Alice and Bob have generated the same key. An adversary observing the communication between them has the values g, n, X and Y but is not able to compute either x nor y - discrete logarithm computation in finite fields is an NP problem - thus it can neither compute $g^{xy} \; mod \; n$.

The algorithm is practical in many situations as no secure channel for information exchange is needed and the parties do not need any information about each other, but it has a serious problem if an adversary is in the MitM position. As no mutual authentication between the parties takes place, the attacker Mallory can pretend to Alice to be Bob and to Bob to be Alice. This way Mallory generates common keys with Alice and with Bob gaining the possibility to decrypt messages from Alice to Bob and the other way round. If the attack is to remain undetected, he just has to retransmit the messages to the right recipients using the keys he shares with them. To remedy this problem, some form of authentication has to be done before DHKE is used.

2.5.2 Encryption and Decryption with RSA

The name RSA is a combination of the initials of the surnames of his developers – Rivest, Shamir and Adleman. RSA bases on the factoring large integers problem, which is supposed to be NP-hard at the current state of mathematics knowledge.

The RSA algorithm allows the encryption and decryption of data with the mathematically connected key pair (K_{pub}, K_{priv}).The first part (K_{pub}) is used for data encryption and consists of the modulus n and the exponent e, which are prime to each other. This key is to be shared with everyone willing to securely communicate with the key owner and is therefore called public. The second part (K_{priv}) is the decryption key, has to be kept in secret and is hence called private key. It consists of the same modulus n and the multiplicative inverse $d = e^{-1}$. Though they are mathematically connected, there is no efficient way to compute the private key from the public key.

This method gives two arbitrary parties the possibility to communicate over an insecure network without having to exchange common encryption key over a secure channel, but it is about 1000 times slower than symmetric algorithms. If used directly on the data to be encrypted it is also vulnerable to so called chosen-plaintext attack. This means that an attacker having the key K_{pub} is able to encrypt chosen texts and compare them to a intercepted ciphertext (encrypted text block). This way a match between the ciphertext and the attacker's encrypted value reveals the encrypted content of an intercepted message. This method does not allow the decryption key to be recovered, even if a successful attack can be mounted.

2.5.3 Digital Signatures with RSA

According to Schneier [31, Chapter 19.3], the RSA scheme can also be used to digitally sign a message M so that a receiver can verify the identity of the sender this message. This time the encryption operation is used for signature verification and the decryption operation matches the signing of a message. As both tuples (signing, encryption) and (verification, decryption) are reverse operations, caution is to be taken when the same key is used for both encryption and signature purposes. If an attacker is able to influence the input and get the resulting output, attacks on these schemes are possible. Examples like chosen ciphertext, common modulus and low exponent value attacks and further problems connected with this type of cryptography are described in detail in [31, Chapter 19.3].

In this subsection the principles of asymmetric cryptography were presented and the RSA algorithm for data encryption and digital signing was explained. In the next section an algorithm based on RSA will be presented, which makes it possible to achieve a digital signature from a third party on your own data, without having to disclose the data to it.

2.5.4 Blind Signatures

The concept of blind signatures was invented by David Chaum [31, Chapter 23.12] and uses the already presented RSA scheme for digital message signing. It allows one party to receive a signature on a transformed (blinded) message, which can later be transformed back to the original message with a signature. This way the signing party does not see what is the content of the message to be signed, which makes this approach appealing for privacy-concerning applications like online voting systems.

To give an example of the algorithm, the two parties Alice and Bob are again involved in communication. Alice uses the private key (d, n) and Bob is in possession of the corresponding public key (e, n) (see Section 2.5.2). So if Bob wants Alice to blindly sign a message that Bob generated, the following steps are to be executed (following Schneier [31, Chapter 23.12]:

1. Bob choses a random number k, where $1 < k < n$.

2. next he blinds the message M by computing
   $t = Mk^e \ mod \ n$.

3. Alice signs the blinded result t by computing
   $t^d = (Mk^e)^d \ mod \ n$.

4. Bob reverses the blinding procedure by multiplying t^d (signed message) with k^{-1}:
   $s = t^d k^{-1} \ mod \ n = (M^d k^{ed})k^{-1} \ mod \ n = M^d \ mod \ n$
   ($k^{ed} \ mod \ n = k$, see above).

As this approach bases on the RSA algorithm, the same concerns on the security apply here too.

2.6 Connection Security

To ensure the integrity and secrecy of any data transferred over a network the data should be transmitted only in combination with a `Message Authentication Code (MAC)` and be in encrypted form. This way no adversary is able to manipulate or read the information while in transmission.

The MAC value is generated with the help of a special cryptographic hash function, which compared to normal ones takes an additional parameter. This is a secret key known only by the parties communicating, making it impossible for adversaries to generate a valid MAC for a given message. For the encryption of the traffic both symmetric or asymmetric cryptography can be used. The advantages and disadvantages for both types are shown in Table 2.1.

Symmetric cryptography is easy to use, as the algorithm is straightforward, the shared key is generated randomly and the encryption/decryption functions are about thousand times faster than the asymmetric counterparts. Though key negotiation/exchange has to happen over a side channel or an already existing secure channel over the insecure network e.g. over a trusted third party. If communication with an unknown party is needed, there is no way to verify that only the right person got the shared key which makes MitM attacks possible.

Asymmetric cryptography enables the parties having each others public keys to establish a secure key (thus secure channel) over a public network without any difficulty. As the keys needed for this purpose are `public`, they can be uploaded to servers accessible by anybody or even send unencrypted per e-mail. Though using public-key cryptography is really slow. Furthermore it has to be verified, that the key does belong to the right person.

Because of these weaknesses both methods are not directly suitable for ensuring connection security. If a connection with large data volume is to be secured, no asymmetric cryptography should be used. In contrast, if only a small data volume is to be transferred, the enormous costs needed to share a key for a symmetric algorithm are not justifiable. Thus hybrid algorithms evolved, which use asymmetric cryptography to establish shared keys over an insecure network and then symmetric

	`Symmetric Cryptography`	`Asymmetric Cryptography`
Pros	• fast encryption/decryption • easy to use scheme • shared key generated at random (one-time usage)	• usable on insecure networks • no secrecy needed by public key distribution
Cons	• access to shared secure channel for key distribution • key distribution problematic • communication with unknown parties complicated (no secure channel for key exchange)	• slow encryption/decryption • public key of communication party has to be available • identity of public key owner not directly apparent

Table 2.1 Comparison of the advantages and disadvantages of symmetric and asymmetric cryptography.

algorithms are used to secure the data. But this is still not enough, as no MAC information for data authentication/integrity is contained.

A for this purpose suitable cryptographic protocol is `Transport Layer Security (TLS)`, the successor of the `Secure Socket Layer (SSL)` protocol. It exhibits a hybrid behaviour negotiating an encryption key over asymmetric methods, supporting a lot of symmetric ciphers for the actual connection encryption and providing hash functions allowing MAC generation for integrity protection. As TLS works on layer 6 in the ISO/OSI network model, no changes in the application functionality are needed to implement it, but only the socket of a newly established connection has to be extended to use the protocol. As SSL/TLS had a number of security related problems ([13] and [11]) in recent years, the protocol has to be used with caution, so that no security issues are introduced in the implementation.

2.7 Anonymity

In this section two different ways will be presented which allow users to guard their privacy while using a network like the Internet. There are anonymous message boards and anonymous proxies. The former focus on not saving any connection data which can be used for identification. The latter are servers operated by independent organisations, which users can use as packet relays. They forward (proxy) packets on behalf of the users and hide the real connection data from the server the user wants to communicate with.

2.7.1 Anonymous Message Boards

In the sense of computer science, a message or bulletin board is piece of software allowing users to post messages, which are available to all visitors. Nobody (besides the system operator) is able to modify or delete them. If the message board system does not exhibit the information needed to connect a given post to a user identity, the system is called Anonymous Message Board (**AMB**), hence allowing anonymous postings to all reading the respective board. Such systems are mainly used in electronic voting systems, where a adversary has to make a commitment in an anonymous way.

2.7.2 Anonymous Proxy Servers

An Anonymous Proxy Server (**AP**, **APS**) or anonymiser is a intermediate entity providing a packet forwarding service on behalf of the clients using it. Furthermore the IP address information of every client is hidden and it seems that the anonymiser is the connection endpoint. This allows users to anonymously use the Internet making their online activities untraceable, which is the main reason for using an APS. An additional purpose could be to circumvent access control (filtering) based on geographical location, nationality and so on.

Though the client requests coming through the APS are indistinguishable for a service provider, the IP address information of every client is available on the APS. Because of this fact a certain degree of trust in the APS operators is required, when using an APS.

To reduce the needed trust in a single operator/authority a high number of anonymisers can be used simultaneously to build a chain of APSs to communicate with a given service. In this scenario the information of all used proxies is needed to be able to correlate the clients with the services they used. This way the privacy is enhanced proportionally to the number of APS nodes used. The Tor[4] project and JAP[5] are a nice example of software implementations allowing chaining of anonymous proxies and using also other innovative concepts like dynamic chains (proxy servers used change in time) to improve the anonymity of their users.

However there are still some security relevant problems when using an APS with an unencrypted connection. The proxy server (or proxy chain) is able to read all data transmitted through it as it is in the man-in-the-middle (**MitM**) position. This is a security risk if confidential data is exchanged. Furthermore if adversaries have access to the local network of a given area, they can check if a specific server is being contacted through observing the traffic on the network. As it is generally a good idea to uses encrypted connections, it is much more important in the case where anonymisers are used, as the anonymity/privacy is obviously highly prioritised.

[4]https://www.torproject.org/
[5]http://anon.inf.tu-dresden.de/

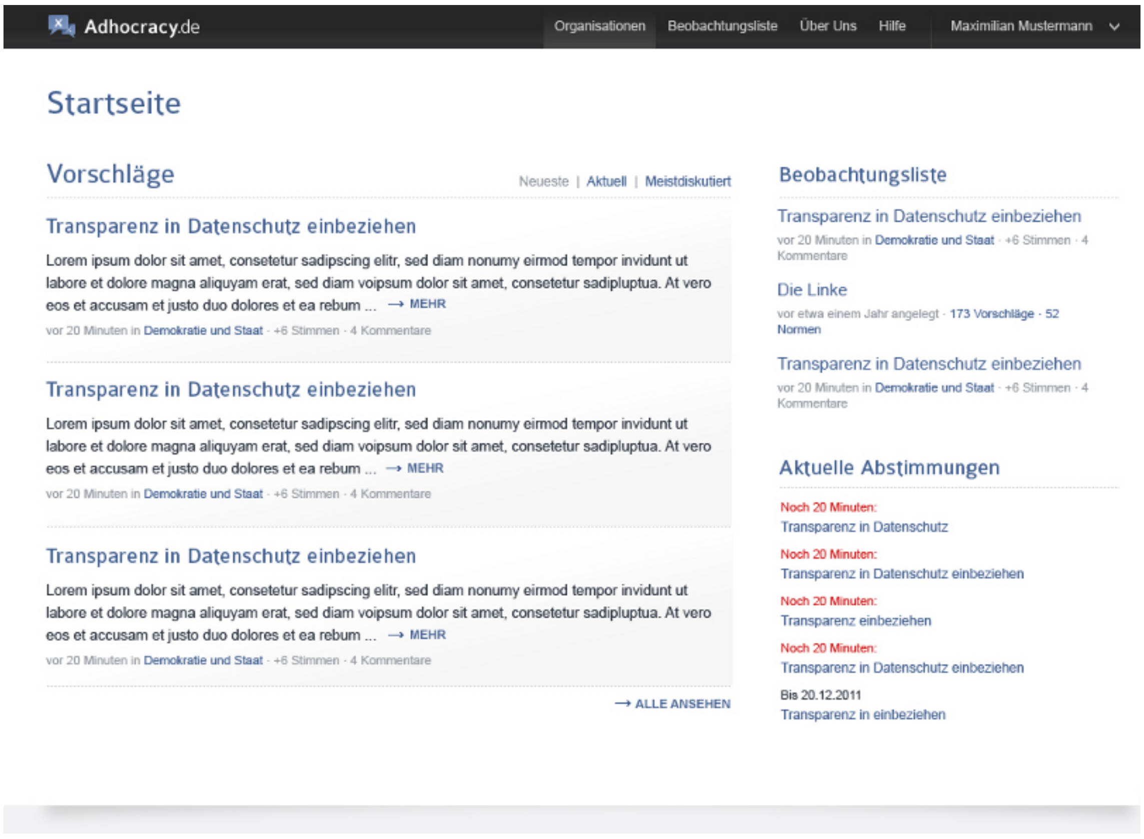
3 Related Work

The rising interest of more and more citizens in directly influencing the government course gives rise to several new implementations of `LD`. Currently more than 20 projects exist and most of them are actively developed. In this chapter I will present only the three mostly used and best-known of them, which are `Adhocracy`, `Votorola` and `LiquidFeedback`. The reader will be able to see the advantages and disadvantages in these systems and be able to compare them to my design described in Chapter 4. I will come back to these approaches in my evaluation in Chapter 6.

3.1 Adhocracy

Adhocracy is an open-source project developed by the Liquid Democracy association and is available under the `Affero GPL v3 (AGPLv3)` license[1] meaning, that the source code is included and it can be freely used, redistributed and modified. At the time of writing version **1.2** of the project is available in the code repository. The development is done in the high-level scripting language Python[2] with the help of the `Pylons Framework`[3]. Though Python is not extraordinary fast in execution time compared to low-level programming languages, it allows rapid and straightforward software development. Furthermore the usage of software like `Apache Solr`[4] and `Memcached`[5] used normally in scalable, high-performance systems suggests, that the performance of Adhocracy is good even by reasonably large ($>$ 100k) amount of participants. Much work is invested in the graphical user interface - it's default appearance is simple and appealing, as shown in Figure 3.1. This allows even unexperienced users to operate the system with ease after a short adaptation phase.

[1]http://www.gnu.org/licenses/agpl.html
[2]http://www.python.org
[3]http://www.pylonsproject.org/projects/pylons-framework/about
[4]http://lucene.apache.org/solr/
[5]http://memcached.org/



Startseite

Vorschläge

Neueste | Aktuell | Meistdiskutiert

Transparenz in Datenschutz einbeziehen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voipsum dolor sit amet, consetetur sadipluptua. At vero eos et accusam et justo duo dolores et ea rebum ... → MEHR

vor 20 Minuten in Demokratie und Staat · +6 Stimmen · 4 Kommentare

Transparenz in Datenschutz einbeziehen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voipsum dolor sit amet, consetetur sadipluptua. At vero eos et accusam et justo duo dolores et ea rebum ... → MEHR

vor 20 Minuten in Demokratie und Staat · +6 Stimmen · 4 Kommentare

Transparenz in Datenschutz einbeziehen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voipsum dolor sit amet, consetetur sadipluptua. At vero eos et accusam et justo duo dolores et ea rebum ... → MEHR

vor 20 Minuten in Demokratie und Staat · +6 Stimmen · 4 Kommentare

→ ALLE ANSEHEN

Beobachtungsliste

Transparenz in Datenschutz einbeziehen

vor 20 Minuten in Demokratie und Staat · +6 Stimmen · 4 Kommentare

Die Linke

vor etwa einem Jahr angelegt · 173 Vorschläge · 52 Normen

Transparenz in Datenschutz einbeziehen

vor 20 Minuten in Demokratie und Staat · +6 Stimmen · 4 Kommentare

Aktuelle Abstimmungen

Noch 20 Minuten:
Transparenz in Datenschutz

Noch 20 Minuten:
Transparenz in Datenschutz einbeziehen

Noch 20 Minuten:
Transparenz einbeziehen

Noch 20 Minuten:
Transparenz in Datenschutz einbeziehen

Bis 20.12.2011
Transparenz in einbeziehen

Figure 3.1 The simple and appealing design of a sample Adhocracy installation.

“The core process of Adhocracy is focussed around the creation and collaborative development of (policy) Proposals” [15, Platform Basics]. **Proposals** are actually objects consisting of **Text** objects. The system allows suggestions on both types to be made through the use of the **Comment** objects. So a discussion about the strengths and weaknesses of one Proposal as a whole can be started, but also a minor problem concerning only a part (one or more Text objects) of the Proposal can be addressed. Voting procedure can be started for any of the three types of objects. For the Text or Comment objects it rates their popularity, whereby only the users voting in favour or against the object are counted. As a result the object is either included in or rejected from the Proposal. In the case of a Proposal, the voting procedure is considered successful only if a quorum of all system participants is reached (e.g. 66%) and hold for a minimal amount of time (e.g. 1 week).

Both of this voting procedures described above allow proxying of the own vote. The delegation types supported by the system are one-time, multiple, only for an area (e.g. politics, health care) or a global, not limited in time delegation. In case of a multiple delegation, only if the same decision is made by all, the vote is counted. The vote owners have the right to outvote their delegatees at any time.

Every Adhocracy installation allows a large number of instances to coexist on the same system. User registration is required for system access, but no encryption and integrity protection is supplied in any form. “Adhocracy does not allow for any kind of secret voting. A full public voting record is available for all users, at any time.”[16, SecretBallots]. On the topic of system security, the Adhocracy developers deliberately do not deploy complicate protocols and cryptographic routines to secure the voting procedure. They state:

> "Unfortunately, most of these proposals are extremely complicated and require the user to install special software on their PC, follow non-trivial processes and to manage cryptographic keys. Yet, even these tools do not guarantee perfect security: shit just happens."[16]

which makes clear how highly improbable it is, that the developers focus on security aspects any time soon. Despite the developer's attitude with respect to anonymity and security, Adhocracy is the most commonly used and tested system by the political parties in Germany, regarding `LD`. Bündnis 90/die Grünen (Alliance 90/The Greens), die Linke (the Left), SPD and FDP are experimenting for more than a year with the software. It was also used in the `Munich Open Government Day`[6] to collect proposals for the Munich Open Government initiative. On the one hand, this shows the necessity of a `LD` implementation, but on the other hand, it raises the question, why the most common of the implementations does not even try to design a secure and anonymous voting system.

3.2 LiquidFeedback

The next software project I will discuss here has the name `LiquidFeedback` and is developed by the Public Software Group association. The system is split in two packages – the core (backend) and the user interface (frontend).

According to the Public Software Group their implementation "focuses on structured feedback and the voting process itself while leaving the means of discussion within an initiative (alliance or party as far as an issue is concerned) to the choice of a given initiative"[18]. As these are also key aspects in Adhocracy, a similar approach is applied in LiquidFeedback. The key difference is the hierarchical layer, which is used to categorise the initiatives in `themes`. Initiatives are the equivalent to proposals in Section 3.1 and are used as the main tool in the consolidated decision making. A user willing to start a new initiative has to find the best matching theme and initiate it there. Comments and competing initiatives can be issued there as well.

Before the initiatives are mature enough to start a voting procedure, the theme containing them has to run through a number of the following statuses:

1. **NEW** - A theme with newly added initiative(s).
2. **DISCUSSED** - Comments on and changes of initiatives are allowed.
3. **FROZEN** - No modifications of the theme are allowed.
4. **VOTING** - Voting procedure on the initiatives currently takes place.
5. **CLOSED** - Voting procedure has finished.

[6]http://mogdy.adhocracy.de/instance/mogdy

6. **ABORTED** - Theme didn't reach the minimal requirements for a voting procedure to be initiated and the theme has been closed.

These statuses are bound on quorum levels and time spans, which are defined by the operator of the LiquidFeedback instance. Typical numbers for the quorum can be from 20 to 30%. and 2 to 4 weeks for the time spans.

A new theme with shortly added initiatives first has to constitute a quorum by exceeding a minimum user interest level for a specified period of time. If one or more initiatives of the theme fulfill these conditions, the theme state changes to **DISCUSSED**, otherwise it is closed and the status is changed to **ABORTED**. In this state every user is able to comment the initiatives and add new possibly competing initiatives. After another time span, the theme status changes to **FROZEN**. This phase has a proportionally short duration and its purpose is to hinder last minute changes in the text of every initiative. It furthermore includes the condition, that every initiative has to reach a specified quorum to be added to the list with choices for the vote. This way the users are not flooded with too many possibilities, but have a small amount of well supported initiatives to choose from. After a theme has reached the **VOTING** status and at least two initiatives reached the quorum, a voting procedure can take place. As the system implements the `LD` concept, it implicitly allows vote delegation. For example a user Alice can delegate her voting rights to one or more experts simultaneously, whereby the vote can be used only by the one of them. Furthermore she can delegate her voting right only for a certain theme to another user. This way she doesn't have to bother any more about this particular decision, but still actively takes part in other voting processes. An important difference to the other systems presented in this chapter is the time-bound delegation – after a given period of time this delegation is automatically canceled. This allows users to think over their decision in regular intervals. The cancellation can also be issued by the vote owner, allowing to recall the delegation at any time.

While a voting procedure is ongoing, related informations are kept private, but after its end all the information is made public. This allows users to check how their delegations have been used, and check the correctness of the voting results. Consequentially no anonymous vote can be held with the help of LiquidFeedback. The direct connection between user account and vote can be established by everyone having access to this public data, which also makes it impossible to hold a secret election. Recently the use of pseudonyms has been implemented to counter this problem. Although the mapping between real name and pseudonym can still be achieved by the system operator, which would allow vote filtering based on the user's identity. As no receipts are issued, nobody is able to prove that a manipulation has taken place.

To safeguard the secrecy and integrity of information exchanged with the LiquidFeedback system TLS can be used on the web server, which weakens the chance of a MitM attack. Although no security concerns are discussed on the homepage of the system and there is no infrastructure to detect modification in the database or en route.

LiquidFeedback is primarily used by the Pirate Party in Germany, therefore the developers focus on implementing features interesting for this party. Still, the software is open source and published under the MIT license[7]. This allows everyone to acquire the source code, distribute or modify it and use it for his purposes.

The primary users of LiquidFeedback are the local branches of the Pirate Party, which set up on-site instances of the system and use them for internal decision making. Since August 2010 a nationwide Pirate Party LiquidFeedback instance[8] is available. At the time of writing, it has about 7000 users with 1400 themes open for discussions and comments and is the largest installations in existence, showing the largest interest in `LD`.

3.3 Votorola

Finally, I discuss the software project Votorola. In comparison to both previous systems, it is developed by a small group of only three active contributors and is not connected to any political party or association. Votorola is being developed in Java, which makes it OS independent. The system architecture is modular and contains among others the following modules:

- Vote engine
- User interface (**UI**)
- Voting register
- Discussion platform

Because of this structure it is possible to exchange single modules without modifying the complete system. Distributed voting approach is feasible, as the modules don't have to run on the same server nor to have access to the same database. Currently the user interface looks similar to Mediawiki[9], as Votorola uses it as a discussion platform. Though the implementation is still a prototype and is actively tested and developed, the architecture seems promising. Compared to Adhocracy and LiquidFeedback, it uses another approach for consolidated decision making called `communicative delegation`[10], which is presented in Figure 3.2.

In the first step of this process participants make drafts with their views publicly available. Next every participant searches for similar views, is able to discuss them with the *owner* and possibly modify them, so that they correspond to common views. To support this new draft the panellists delegate their own votes to the owner of the draft, who becomes a proxy/expert for their views. Stepwise every group searches for other groups having similar views and cooperates with them. Agreeing to compromises and creating new drafts, all can achieve greater support for their cause in form of more delegated votes to the new draft. This repetitive proxying results in a number of delegation trees, which contain all drafts. As every time compromises are made to get further support in the next tree level, at some

[7]http://www.opensource.org/licenses/mit-license.php

[8]https://lqfb.piratenpartei.de/

[9]http://www.mediawiki.org

[10]http://zelea.com/w/User:ThomasvonderElbe_GmxDe/Communicative_Delegation

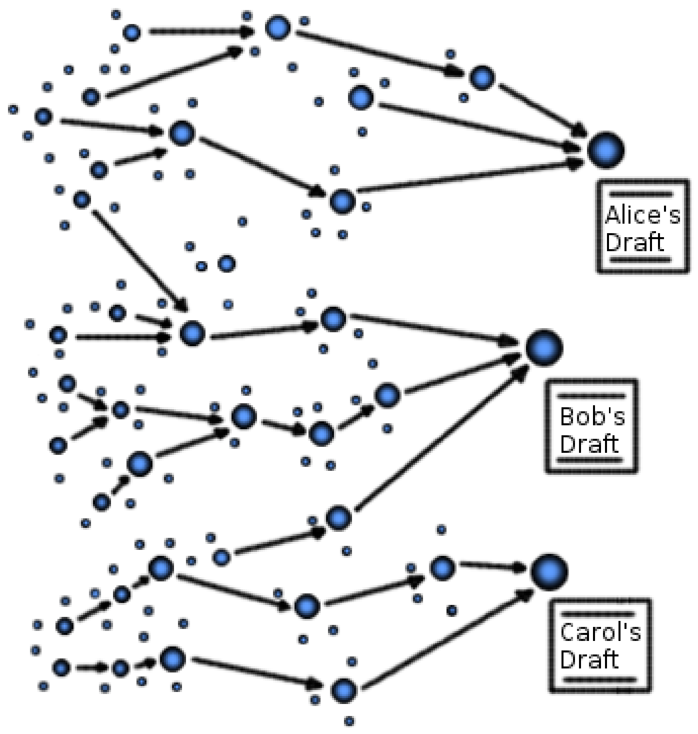


Figure 3.2 Communicative delegation process, where a large amount of small groups discuss proposals and cooperate on shared-interest drafts. At the end only a reasonable amount of drafts is available for voting.

point in time the interests of a individuals supporting the draft in tree nodes, e.g. in the leaves, are not present any more. This voters can withdraw their support and hence delegation and search for a new draft better suiting their views. After a number of iterations this process stabilises and results in small amount of final drafts (tree roots) with large support, which are then used as possible choices for the voting procedure.

As a result of its architecture, Votorola has support for delegation of delegated votes, as this is the way the support in the tree is represented. Delegation is possible only to one entity. Delegation to multiple entities would make it hard to oversee in which tree branch a delegation is really counted and if it is counted only once. The system does support outvoting and also a permanent domain delegation is planned, but not yet implemented. This would allow a proxy to permanently (with outvoting still possible) use a vote in a given domain e.g. privacy, politics, health care.

The Votorola system is designed in such a way, that voting procedures are fully transparent. Every participant is able to access the votes of others and reconstruct the proxy chain for their votes. This facilitates a publicly verifiable voting results but also makes it impossible to hold a secret or anonymous election. Changes allowing secret or anonymous voting make the Votorola system unfeasible. Furthermore no measures have been planned to secure the votes in case of system state manipulations or allow the voters to prove their voting decisions using receipts.

3.4 Summary

The currently existing projects implementing delegated voting concepts focus mainly on the collaborative decision making and allowing verifiable results through making the voting information public. This way no anonymous and secret voting can take place. Furthermore none of the systems issues receipts in any form to supply the user with a way to prove manipulation by the system operator. No cryptographic procedures are used to secure data on the transport way or in the system making it possible for attackers to influence the result of such a voting procedure without being detected. None of the presented systems even roughly satisfies the voting requirements put in place by the German Constitution. Consequently, they are not useful as a electoral procedure on the state or regional level.

In the following chapters I will describe a system implementing vote delegation, which strongly bases on cryptographic routines and focuses mainly on the security aspects of the voting procedure. My aim is to have a system conforming with German Constitution and allowing a publicly verifiable voting results in a secure and anonymous way.

4 Design

This chapter introduces the design of the system and is divided in five sections. First I formulate the requirements for the system and then present the architecture on a conceptual level. This is followed by a detailed description of the voting process and a section dedicated to further improvements of the concept. The last section shortly summarises the system design.

4.1 System Requirements

The main goal of my thesis is to create a voting architecture based on the `LD` concept. The focus is on the security of the system and the anonymity of the participating users. Both attributes are not guaranteed in the current implementations discussed in Chapter 3 as these mainly concentrate on consolidated decision making. Furthermore I aim at conforming to national voting regulations and laws in Germany, which would allow the concept to be used for elections on the regional or state level. This leads to the following requirements, which have to be satisfied by the design:

1. **Voting over the Internet** - The voting procedure can be carried over an insecure network like the Internet.
2. **Anonymity** - The information available in the system doesn't allow the identification of individual voters or the correlation between them and their votes.
3. **Secrecy and integrity of transferred data** - The data is transferred only over encrypted and integrity protected connections, so no third party is able to intercept plain text data or manipulate it unnoticed.
4. **Voting results integrity** - Manipulation of voting data is not possible or highly improbable.

5. **User authentication/authorisation** - The identity of the individual can be checked at the moment of issuing voting credentials and the right to vote can be verified during voting.

6. **Generation of credentials only to valid users** - The server can generate valid credentials only on request of an authorised user. If it randomly generates legit credentials, the chance of being detected is high.

7. **Non-repudiation** - Voting credentials are acquired in such a way, that the client can't deny having acquired such.

8. **Proxying** - The own voting rights and delegated votes can be further delegated to one or more entities, albeit only a limited number of times. (demanded by `LD`)

9. **Proxy-Reversing** - A delegator can revise a vote-forwarding decision at any time before the election is over. (demanded by `LD`)

10. **Public voting results** - The voting results are public and verifiable by everyone.

The German Constitution (see Section 2.3) regulates the voting procedure and requires, that every election for the parliament is **general**, **direct/immediate**, **free**, **equal** and **secret**. General, free and equal are fulfilled through the anonymity-requirement (2.) as this prevents the system to distinguish between different users. Discrimination is theoretically possible (though prohibited by law) on the side where voting credentials are acquired. But as this is comparable with the state-of-the-art voting procedures, there should be no drawbacks for the concept based on this fact. Fulfilling the requirements (2.) and (3.) makes it impossible for an observer with (insider) or without knowledge (outsider) of system information to know what a voter has voted for. Though bystanders, who are able to repress/threaten the voter, can be present at the moment of voting. This is mitigated by the requirements (8.) and (9.), which would allow a repeated voting, but would suppose that the bystander is no longer on site. This problem also exists in the postal voting mechanism, which conforms to the German Constitution. For this reason the way to eliminate it will not be discussed in the main concept part but is introduced in Section 4.4, where a further improvement of the design is presented.

The system architecture presented in the next section fulfills the above defined requirements and should satisfy the regulations in the German Constitution with respect to voting. It should therefore be usable for parliamentary elections.

4.2 Conceptual Overview

The architecture of my system is presented in Figure 4.1. There are three servers employed in the process – the `Voting Register (VR)`, the `Voting Computer (VC)` and the `Anonymous Proxy`. The VR contains all the information needed to verify the identity of the voters, supplies them with voting credentials and records who did get credentials and who did not. The VC is responsible for verifying the validity of

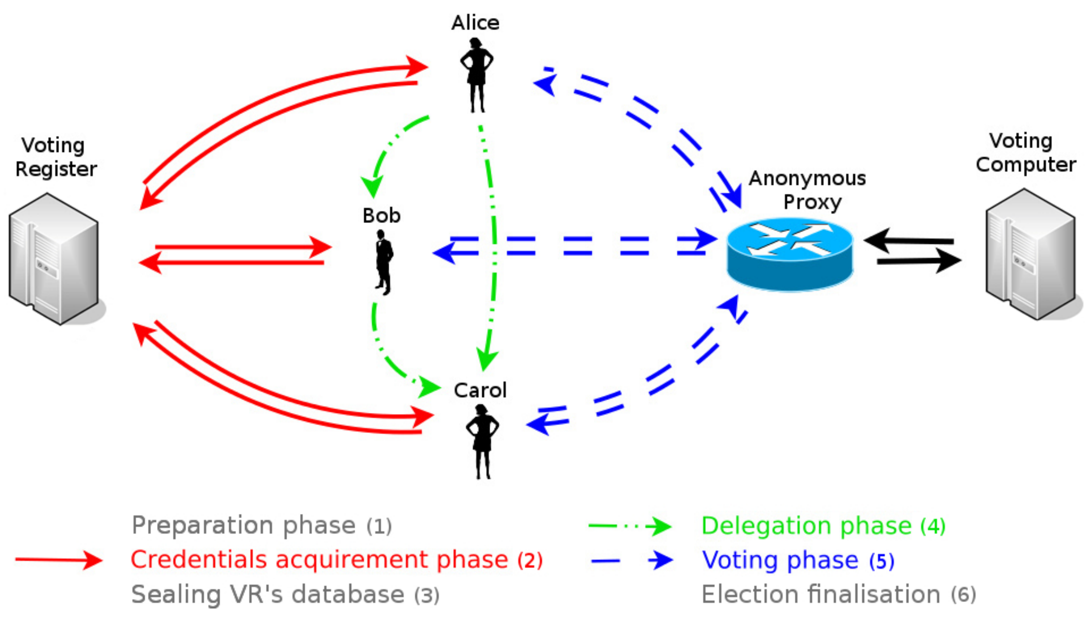


Figure 4.1 System design showing the relations between the entities Voting Register (VR), Anonymous Proxy (AP), Voting Computer (VC) and the three voters. Alice, Bob and Carol can interact between each other and the servers through the operations having an arrow in front of their name. The three remaining operations are executed only from VR and VC and don't require any interaction.

the voting credentials, recording the ballot and issuing a receipt for it. The AP is needed so that the VC is not able to tell voters apart (relying for instance on IP address information) and not able to discriminate them.

To be able to take part in a voting procedure, valid voting credentials are needed. Acquiring these can happen either through directly interacting with the VR (1) or through proxying (2). The actual voting (3) is the concluding step, that if successfully carried out, results in a receipt from VC and implies a successful completion of the procedure.

The following section describes in detail the three processes from Figure 4.1 and adds two additional steps to round the procedure off. Subsection 4.3.1 presents the pre-processing needed, for an actual voting to be able to take place. Subsection 4.3.2 shows how voting credentials can be acquired anonymously and securely, followed by Subsection 4.3.3 showing the vote forwarding process, focusing on the security issues which can arise, if the voting credentials are not kept secure. Section 4.3.4 and presents the actual voting consisting of verification procedures and receipt generation. The following Section 4.3.5 describes the post-processing actions like voting results calculation and publishing.

4.3 Voting Procedure

The design of the voting system presented in Figure 4.1 can be divided into the following six phases:

1. Preparation (Section 4.3.1)
2. Interaction with VR (Section 4.3.2)
3. Sealing VR's database (described in election finalisation, see Section 4.3.5)
4. Delegation process (Section 4.3.3)
5. Interaction with VC (Section 4.3.4)
6. Election finalisation (Section 4.3.5)

Each of them describes in detail the exact actions users and servers, involved in the election process, are allowed and able to carry out and the interaction that can or must occur between them. Furthermore I describe the problems in terms of secrecy and anonymity and present solutions for them. If further issues arise by the deployment of a specific solution, their impact and severity is subsequently discussed.

Before we can proceed with the vote preparations section, a problem arising from requirement (7.) in Section 4.1 has to be discussed. The non-repudiation feature of the VR can be achieved through the use of RSA keys (see Subsection 2.5.2, Chapter 2) as an authentication mechanism. The only person able to decrypt/sign a given message is the owner of the private key, so the identity can be determined unambiguously. As a consequence, every citizen willing to take part in voting procedures has to possess a RSA key pair and the corresponding public key has to be known by the VR. The procedure of acquiring the public part of the user's key can be handled by the authority responsible for the register lists and should require the verification of the user's identity prior to updating the saved key. The exact procedure of supplying the user's public key to VR is out of scope of this thesis. It is assumed, that the register lists needed for the initialisation of every voting procedure include the personal information of all citizens allowed to vote and their public keys. The way these public keys are used to guarantee non-repudiation by acquiring voting credentials is described in Section 4.3.2.

4.3.1 Preparation Phase

Before a voting procedure can be started, both VR and VC have to generate fresh RSA key pairs and reinitialise their internal databases. This explicitly invalidates all voting credentials issued in former rounds. Then, the VR has to acquire a fresh copy of the register list containing all citizens and their public keys from the responsible authorities. This list is then signed with the fresh private key of VR and both the signed list version and its public key are made public. This ends the preparation phase for the voting.

Publishing the signed register list allows citizens to verify whether they are on the list and if the appropriate public key is saved for them. As the VR is allowed to generate credentials only for users on this list, which is sealed through the signature, the upper limit of voters and their particular keys are publicly known. If the VR manipulates the list and modifies a public key for a person, this person will, with high probability, detect the changed key and can prove that the VR tried to cheat. The reason is, that the list is acquired from a trusted third party. In the next phase we will see, why the VR is not able to randomly generate and sign credentials, though having the ability to issue valid credentials.

The possibility to acquire voting credentials is time limited. After the period has elapsed, the VR has to publish the list of all citizens, who have acquired voting credentials. This way there is no possibility to add additional credentials later and the number of voters is delimited again. This makes it more probable to detect a manipulation by VC.

4.3.2 Voting Register Interaction

After the initialisation procedures communication with the VR is possible and every user on the already published register list can acquire voting credentials. Figure 4.2 describes this process and presents the information exchanged between clients and server. To start the procedure the client initiates an encrypted and integrity protected connection to the VR. The exact type of encryption is not important, as long as no content is transferred in plain over the network. Then, an authentication procedure takes place to determine the identity of the client, which allows the VR to check if the user is allowed to acquire voting credentials (authorisation) and didn't do that already. The procedure is carried out with the help of the public key mapped to the user and consists of the following steps:

1. The client sends an identification token to the server, which states the identity of the user.
2. The VR encrypts a randomly generated secret with the public key of the specified user identity.
3. The VR sends the encrypted content over to the client.
4. The client uses the private key to decrypt the secret and sends it back to the server.
5. The VR checks, if its saved value is equal to the value returned from the client and if so, has verified the client's identity.
6. The VR notifies the client if the process was successful or not.

The only person able to decrypt this secret is the one in possession of the private key matching the public key saved on the VR. This way the identity of the client can be proven unambiguous, as long the private key is kept secure.

Hash Chain Generation

At this point the VR has verified the voter's identity, e.g. Alice, and has to check if she has already acquired credentials. As these are signed blindly, there is no means of differentiating if two voting credentials were issued for the same user or

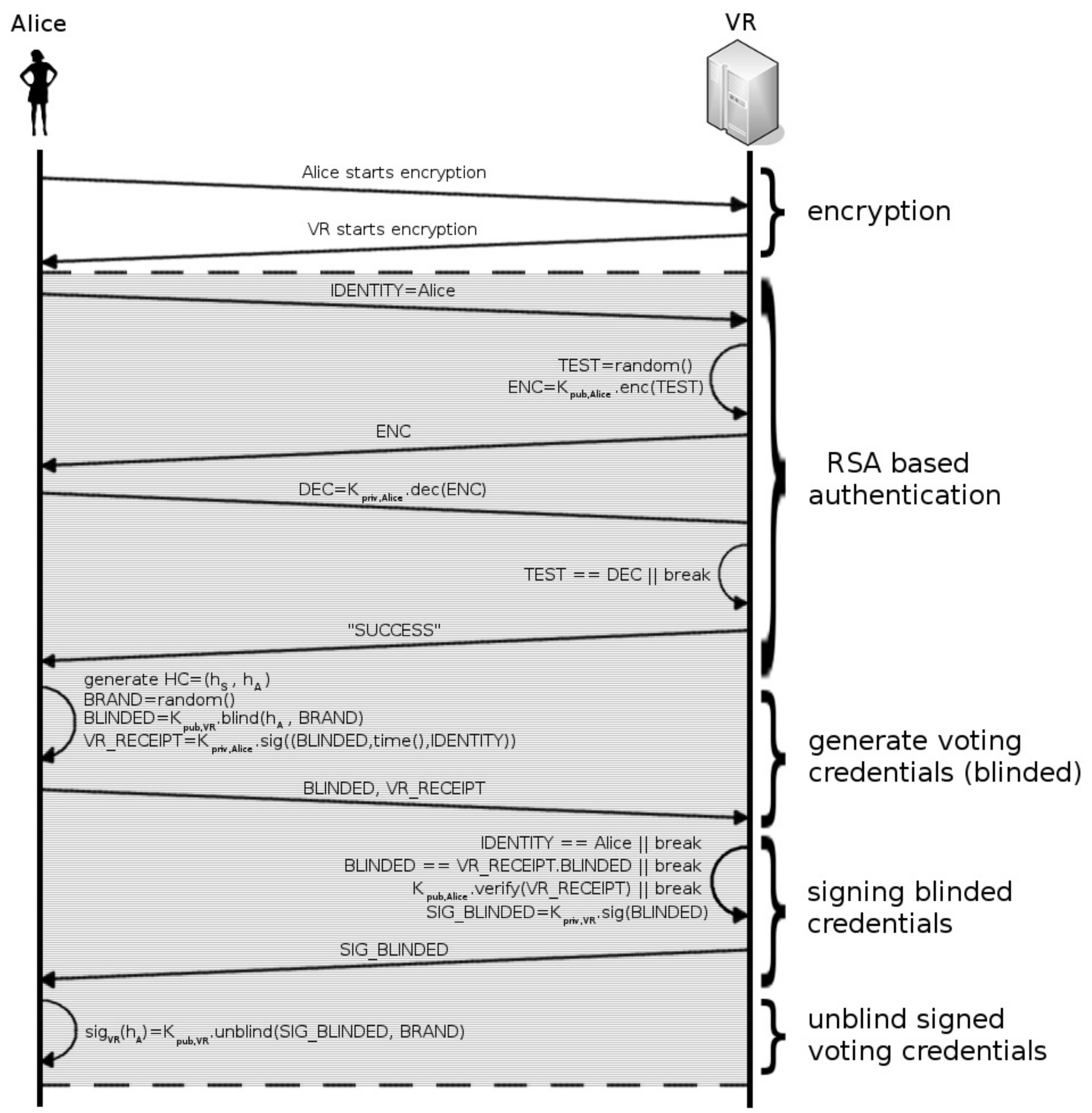

Alice
VR
Alice starts encryption
VR starts encryption
encryption
IDENTITY=Alice
TEST=random()
ENC=K_pub,Alice.enc(TEST)
ENC
DEC=K_priv,Alice.dec(ENC)
RSA based authentication
TEST == DEC || break
"SUCCESS"
generate HC=(h_S, h_A)
BRAND=random()
BLINDED=K_pub,VR.blind(h_A, BRAND)
VR_RECEIPT=K_priv,Alice.sig((BLINDED,time(),IDENTITY))
generate voting credentials (blinded)
BLINDED, VR_RECEIPT
IDENTITY == Alice || break
BLINDED == VR_RECEIPT.BLINDED || break
K_pub,Alice.verify(VR_RECEIPT) || break
SIG_BLINDED=K_priv,VR.sig(BLINDED)
signing blinded credentials
SIG_BLINDED
sig_VR(h_A)=K_pub,VR.unblind(SIG_BLINDED, BRAND)
unblind signed voting credentials

Figure 4.2 Encrypted communication between Alice and the VR with the purpose of acquiring anonymous voting credentials in a secure way.

for two different users. Hence she has to acquire only one successful blind signature otherwise cheating will be possible. Next a hash chain is generated on the client side through the following procedure:

1. Generate random data $RAND$.
2. Calculate $h_S = h(RAND)$ with the help of a cryptographic hash function (see Section 2.4.1).
3. Use h_S to calculate a hash chain HC with an anchor element h_A and depth D (see Section 2.4.2).
4. Save the elements h_S and h_A as they define the hash chain HC.

In step 2) the RAND can directly be used as h_S, though to ensure all elements of the HC have the same length and structure, the $h()$ is used on this value. As a cryptographic hash function is used to generate HC, only the user in possession of the starting element is able to achieve the same result. Having an element h_I between h_S and h_A allows the generation of the part (h_I, h_A) (see Figure 4.3), but does not divulge any information about the preceding elements. This characteristic of hash chains is used for the delegation. The starting first element is kept secret, other elements can be delegated. The distance from the anchor defines the priority of the delegated votes – the further away an element is, the higher the voting priority.

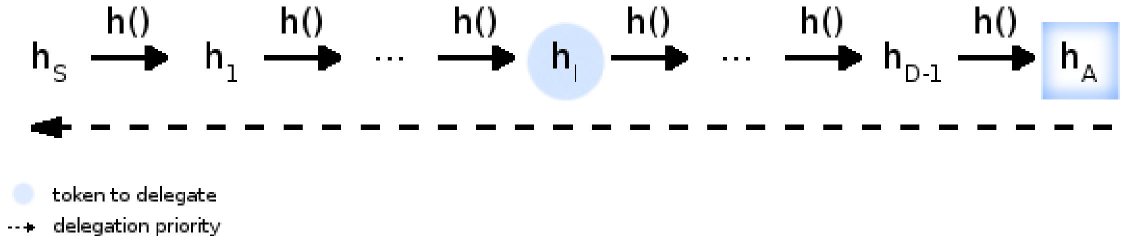


Figure 4.3 Alice possesses the Hash Chain (h_S, h_A). To delegate a Vote, she generates a Hash h_I positioned between h_S and h_I and supplies it with $sig_{VR}(h_A)$ to the Delegatee.

Blind Signature Acquirement

To make it possible for the VC to verify, that the credentials supplied are valid, a signature from VR on the anchor element is used. As the voting procedure has to be anonymous, no connection between the voter's identity and the credentials used should exist. To ensure this Alice's anchor h_A is blindly signed through the following procedure:

1. Alice generates a random value $RAND_B$ for blinding and uses it in combination with the public key of VR (see Section 2.5.4) to scramble h_A producing $BLINDED$.

2. Alice signs the triple $(BLINDED, TIMESTAMP, IDENTITY)$ with her private key resulting in $VR\_RECEIPT$.

3. Alice sends $BLINDED$ and $VR\_RECEIPT$ to VR.

4. The VR checks the signature on $VR\_RECEIPT$ and if successful, saves the value into its database, signs $BLINDED$ and returns the resulting string $SIG\_BLINDED$ to Alice.

5. The VR additionally makes $(BLINDED, TIMESTAMP, IDENTITY)$ and $SIG\_BLINDED$ available on a web page and reflects in the database, that Alice got credentials.

As the blinding operation uses a random value $RAND_B$, whose length VR neither knows nor is able to brute force, its security is comparable with that of the one-time pad scheme. Furthermore the only value VR can observe is the blinded anchor, so it can't deduce the value of the anchor h_A. This makes it impossible for both VR and VC to map this blinded hash value to a user's identity and meets for this reason the anonymity requirement.

The signed triple $VR\_RECEIPT$ is used to preclude VR from issuing randomly generated voting credentials. As the triple contains a fresh timestamp and the actually used blinded value and only Alice is able to generate a valid signature on it, she has actively participated in the protocol, so no cheating by VR is possible. The TIMESTAMP value is used as a nonce and counters replay attacks by VR.

After signing $BLINDED$, the VR transmits the result to Alice and makes the signed triple public. As Alice's public key is available in the voters list, everyone can check if she really requested a signature on the blinded value (triple is signed by her). If this is not the case, it is clear that a manipulation occurred.

On the point before Alice has received the signed value VR is able to interrupt the protocol, as it is in possession of the receipt (signed value) from Alice in form of the signed triple. To circumvent this problem, the communication protocol can be changed so, that the VR is required to post the value $sig(BLINDED)$ as a message on an anonymous message board and forward the link to Alice. If the data behind the link doesn't contain the right signature, Alice can prove VR cheated (commitment scheme).

Once the communication with VR is over Alice is definitely in possession of the signed and blinded value $SIG\_BLINDED$. With the help of $BRAND$ the blinding can be stripped and the result is a signature on the anchor element h_A denoted here as $sig_{VR}(h_A)$. A triple in the form (h_A, $sig_{VR}(h_A)$, h_I), where $h_I \in HC$ and $h_I \neq h_A$, represents the voting credentials for VC.

4.3.3 Vote Delegation

To demonstrate the delegation procedure the users Alice, Bob and Carol will be used. Alice, who is in possession of the hash chain $HC_{Alice} = (h_S, h_A)$, decides to delegate her voting rights to both Bob and Carol, where she sees Bob as the trustworthier. The result of this process is presented in Figure 4.4.

Alice takes the starting hash h_S and computes the hash value h_{64}, which is the 65th hash in HC ($h()$ applied 64 times). So the new hash chain HC_{Bob} is defined through the elements h_{64} and h_A and is a suffix of HC_{Alice}. Supplying Bob with HC_{Bob} in combination with $sig_{VR}(h_A)$ allows him to vote on behalf of Alice, as the new starting element h_{64} can be used to generate h_A in $d - 64$ steps and the signature $sig_{VR}(h_A)$ is still valid for the new chain, as the last element is the same. He can also use HC_{Bob} to delegate Alice's voting rights on the exact same way, allowing further forwarding (proxying) of the delegated voting rights, which is limited only by the depth of the hash chain in possession.

At a later time Alice thinks the matter over, and decides to delegate her voting right to Carol. Alice repeats the procedure described above but uses smaller number of

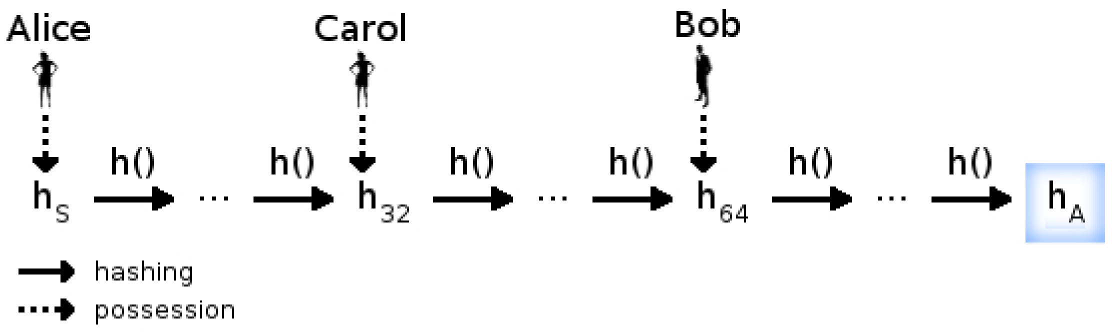


Figure 4.4 Alice first delegates the credentials $(h_{64}, h_A, sig_{VR}(h_A))$ to Bob, then she reconsiders her decision and supplies Carol with the higher priority credentials $(h_{32}, h_A, sig_{VR}(h_A))$.

steps, for example 32. The new hash chain HC_{Carol} is then defined as (h_{32}, h_A) and has the depth $d - 32$ thus having 32 elements more then HC_{Bob}. Alice supplies both tokens HC_{Carol} and $sig_{VR}(h_A)$ to Carol using a `secure` and possibly `anonymous` channel, for example encrypted e-mail, encrypted posting on an AMB, randomly generated URL on a private web server unknown to anyone else or a side channel like telephone.

As the voting rights of Alice are now available to three different voters, the priority of the tokens needs to be unambiguously defined, so no problems occur and no cheating is possible. In the example described here Alice has the highest priority allowing her to outvote both Bob and Carol. As Carol has the second longest hash chain with the same anchor element and signature, she is able to outvote Bob, but not Alice. For Bob to be able to successfully vote with the credentials he acquired from Alice, both Alice and Carol should not use this credentials in the voting procedure.

4.3.4 Voting Computer Interaction

After acquiring credentials directly by the VR or through proxying Alice can connect to the VC and take part in the voting procedure. This process is represented in Figure 4.5. First Alice uses an anonymous proxy server (`APS`) to connect to the VC. This way her identity can not be determined though the IP address of the computer she uses. Next an encrypted and integrity protected connection is initiated between Alice and the VC. This way the APS is not able to read any plaintext traffic although in a MitM position. To start the authorisation procedure Alice sends the anchor and the corresponding signature to VC. This happens anonymously, as neither VR nor VC knows whom this (anchor,signature) tuple belongs to. This prevents malicious users without valid credentials to connect to the VC and cause useless computations beyond signature checking, which would exhaust system resources. To further mitigate the possibility of Denial-of-Service (**DoS**) attacks, it is even possible as a precautionary measure to block users with valid (anchor,signature) tuples, who connect too often or establish a lot of simultaneous connections to the VC.

The demand for public and verifiable voting presents one problem with information contained in the results published online. Participants delegating their own credentials have all the information available to see for what the delegated vote has been used. This means that the authorisation tokens in form of the hash chain and the voting decision taken by the delegatee must not be directly connected on the web page as this would otherwise directly contradict the vote secrecy. Though the users should be provided with the possibility to verify only their own voting decisions.

For this purpose every time a connection between a user and the VC takes place, a fresh random value (`nonce`) is generated with the help of the DHKE algorithm, as described in 2.5.1. This nonce is only known to the actual voter so if published together with the voting decision, it allows verification and does not break the secrecy of the voting. As active participation is needed from both sides to generate it, no client-side replay attacks are possible. This also mitigates VC's possibility to

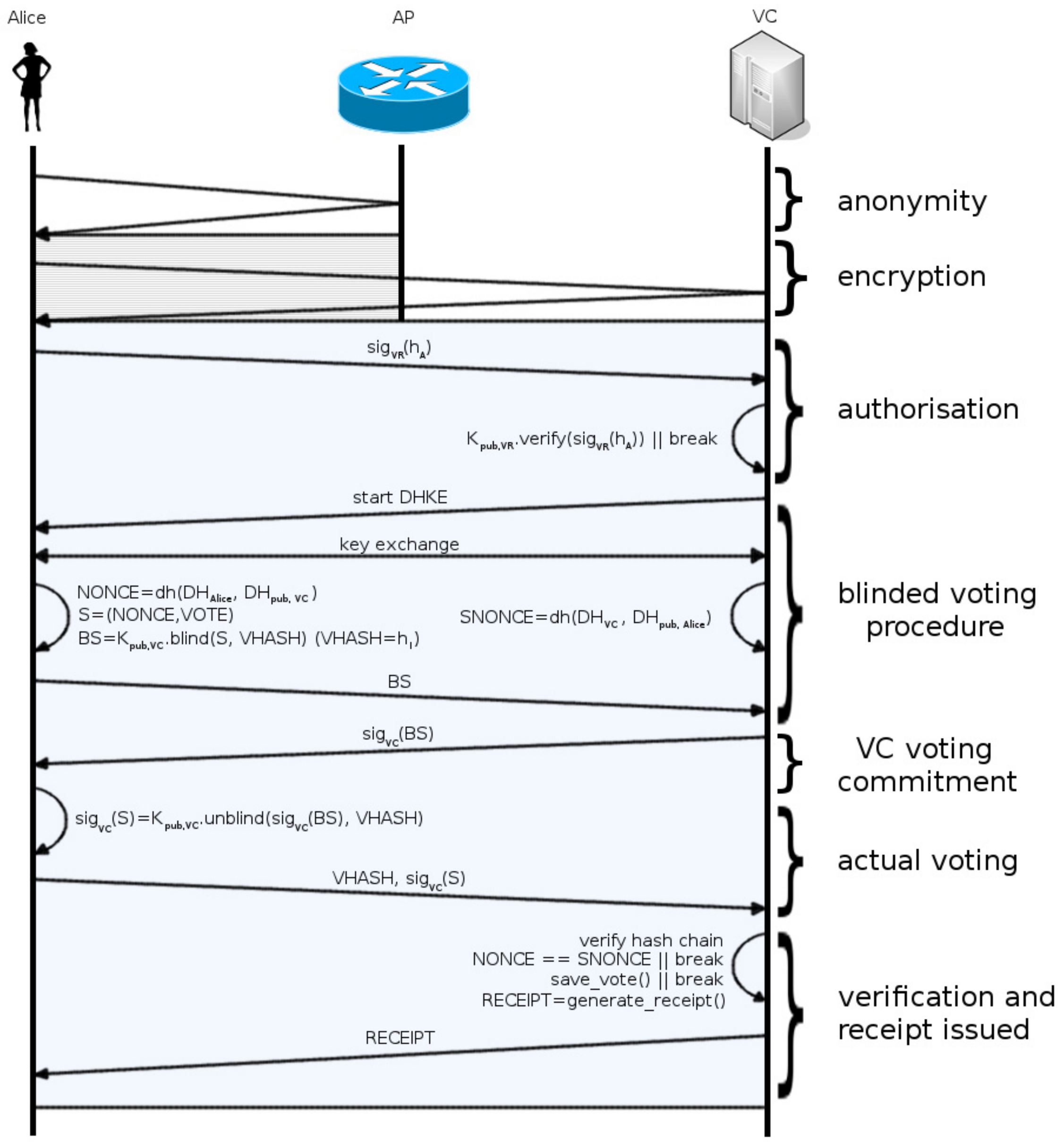


Figure 4.5 Encrypted communication between the voter Alice and the VC. Alice uses an AP to disguise her identity to VC, and acquires a receipt after the voting procedure is completed, which can be used to verify the correctness of her own vote.

generate collisions in the random values used for the check. A collision in this value would make it impossible for the clients having the same value to verify their choices unambiguously.

At the end of the DHKE both Alice and VC are in possession of the same value **NONCE**. Alice concatenates it with her voting decision to a string S. With the help of VC's public key she blinds S using the starting hash **VHASH** of her hash chain as a secret and transfers the resulting value BS (blinded string) to VC over the existing encrypted connection. Through this procedure the risk of VC filtering based on the choice of a voter is mitigated, as VC commits to the choice without seeing it and moreover still has no access to the actual voting credentials (only anchor known). VC saves the value BS temporarily (until the end of connection), signs BS with its private key (commitment) and returns the result to Alice. She reverses the blinding and acquires a signature of the vote she would like to submit. As the VC can't read the submitted vote, she has to send it the value `VHASH` used for scrambling S to BS for the vote to become valid.

At this point in the protocol VC is in possession of both $VHASH$ and $sig_{VR}(h_A)$, which are the voting credentials, and could interrupt without supplying a receipt to Alice. This allows vote falsification, as Alice isn't able to proof she submitted a vote without the receipt. To avoid this problem, Alice can encrypt the values ($sig_{VC}(S)$, $VHASH$, $sig_{VR}(h_A)$) she exchanged with the voting computer using its public key and post them on an AMB. She then sends the link to the encrypted message to VC. This way if VC is accused of cheating it can decrypt this message, which is an evidence if this is really the case or not.

Now that the VC is in possession of the `VHASH`, it first checks if the hash chain was not already used for voting. If this is not the case VC verifies that `VHASH` is really the secret used for blinding and also that it allows the generation of the signed anchor presented for authorisation in a maximum of n_{max} steps. If all this is the case, the voting values are checked for correctness - the value `NONCE` has to be equal to the one generated on the server side and the vote has to be contained in the list of voting options. When this last check is satisfied, a new receipt depending on the last issued receipt is generated with the help of another cryptographic hash function, all information (meaning: anchor, signature, `VHASH`, `NONCE`, vote, receipt) is saved in the database and the receipt it returned to the user. The (anchor,signature,`VHASH`) values are added in a black list. This way voting with the same credentials is no more possible.

With this procedure VC is able to block voters from taking part in the procedure through not accepting valid credentials. Though as every user is anonymous, this can be done based on the information send to VC and is hence reproducible. A user can demonstrate the problem and prove this way, that VC cheats.

4.3.5 Election Finalisation

After the period assigned to the voting has elapsed, the results have to be sealed. This way no further modification is possible. For this purpose two lists are generated by VC, signed with its private key and made publicly available. The first contains all the triples (voting hash,anchor,signature). Its purpose is to allow everyone to check, if only unique anchors are used and verify the corresponding signatures. Furthermore the voting hash to anchor relation (the hash chain) can be tested. The second list consists of the (nonce,vote) pairs. This allows voters to check if their submitted values match the recorded ones, though no connection between vote and (anchor,vhash) pair is publicly present. The finalisation process ends when the saved data is signed by a trusted third party and is copied to a secure location. All the generated data can now be cleared and the VC instance is ready to start a new voting round.

4.3.6 Structure of the Public Lists

Through the voting procedure VR and VC generate lists containing information important for the verification process. This section gives an overview of their structure in chronological order.

First, the structure of the list generated by the VR is presented in Figure 4.6. The voter's identity and the corresponding RSA public key are available from the

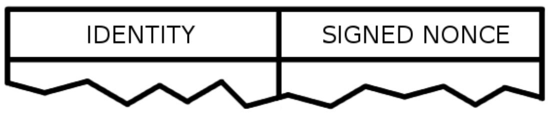


Figure 4.6 Header of the list generated by the VR, which contains the required information to allow the public to verify, that VR issued credentials only to valid voters.

voters list. With their help and the information from the list presented here, every ballot verifier is able to check, if only real voters acquired voting credentials or a manipulation took place. To not allow analysis of the data based on timings, it is published ordered by identity.

Next, the VC has to generate and publish two further lists, which structure is presented in Figure 4.7. With the help of the VHASH and signed ANCHOR values

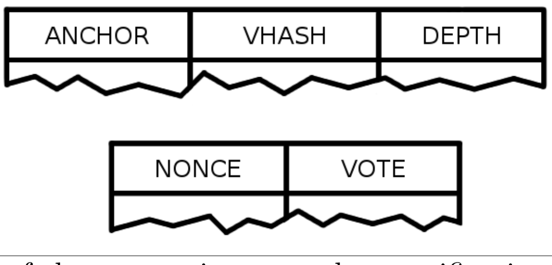


Figure 4.7 Header of the two voting procedure verification lists generated by the VC. The first allows the public to verify, that only valid credentials were used in the voting process. The second facilitates the check, if the own vote is present and counted in the overall results.

from the first list the hash chain can be checked. The DEPTH value is an easy way to verify if outvoting took place or not. Furthermore, if a valid signature is present on the ANCHOR, it is clear that the VR has issued the voting credentials and this is a valid ballot.

The second list has a twofold ability. On the one side it allows voters to verify that the votes are correctly recorded using their receipts. On the other side the verifiers can calculate the overall voting results from this data and assure no modification or cheating has taken place.

4.4 Further Improvements

This section discusses two improvements of the design presented above. The first one describes a way of mitigating the vote-buying/coercion problem. The second one aims at extending the delegation possibilities presented in Section 4.3.3. It describes the use of a data structure more complex than the hash chains, and enables period and topic based delegation.

Coercion/Vote-Buying Remedy

In the design discussed in Section 4.3 vote buying or coercion is possible as the voting results are public and all participants can check their own votes. To counter this problem a trusted third party can be used to verify the results on behalf of the users. Supplying it with all the voting information (credentials, nonce, vote and receipt) allows the review of the voting results. If an error is detected, the trusted third party can issue a complaint and so the identity of the user is not connected with the modified vote. As the use of only one such party results in a single point of failure, it is better to allow the users to choose for example 3 or 5 parties, which they trust in and supply them all with the voting information. This increases the chances for an error to be found.

For the connection to the trusted third party an APS could be used again. This way the identity of the user is not leaked.

Another solution can be to use a variable depth hash chain, which would allow users to outvote the adversary before the voting procedure is closed. Nevertheless, this doesn't mitigate the problem, as in case of publicly available results the adversary is able to verify if outvoting took place or not.

Period and Topic Delegation

The system presented in Section 4.3 allows single and multiple delegations, but the voter has no way to delegate voting credentials only for a given topic. Besides that no period (global but timely limited) delegation is possible. With the help of a Merkle tree (see Section 2.4.3) combined with hash chains, the system can easily be extended to simultaneously support all these features. For this purpose, first N hash chains are generated. Next the hash chain anchors $h_{A,1}, \cdots, h_{A,N}$ are used as leaves for the Merkle tree generation. Then the resulting tree is divided into two different subtrees (not necessarily equal once), as shown in Figure 4.8. The first subtree, e.g. the left one, is used for the topic delegation and each of its leaves corresponds to a given subject, e.g. politics, health care, economics, renewable energy. The leaves of the second (right) tree allow period delegations and describe the time spans, e.g.

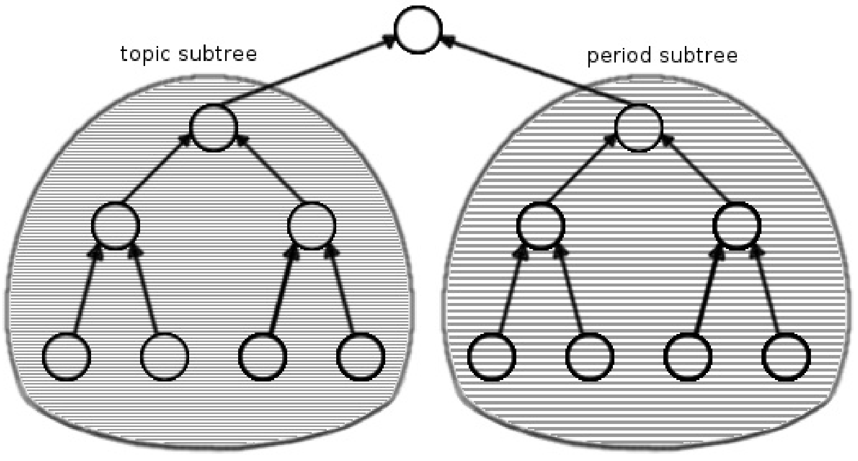


Figure 4.8 Merkle tree structure divided into two equivalent subtrees. The left one is defined as topic and the right one as period delegation.

the first validity week, second validity week, first validity month. The depth of the Merkle tree and the division in subtrees is defined by the authority operating the VC and voters have to conform with that in order to get their ballots accepted by the system. Instead of blindly signing all hash chain anchors, it is sufficient to only acquire a blind signature by VR on the Merkle tree root node (t_R).

To demonstrate the delegation and verification process supported by this new extension, lets simulate it by means of the data structure presented in Figure 4.9. Alice, who is in possession of this structure would like to delegate to Bob the topic `renewable energy`, which is represented by the hash chain $(h_{S,1}, h_{A,1})$. She uses the same mechanism for generating a suffix chain $HC_{Bob} = (h_{I,1}, h_{A,1})$ as described in Section 4.3.3. This time it is not enough to forward HC_{Bob} to Bob, as only the tree root t_R has been signed by VR. To generate the complete Merkle tree Bob needs also the elements $h_{A,2}, t_2$ and t_6 (solid filled elements in Figure 4.9). This means, that Bob has to acquire from the delegator the following three items:

1. The hash chain HC_{Bob}.
2. The remaining elements besides $h_{A,1}$ needed to generate the Merkle tree up to the root node t_R.
3. The signed tree root $sig_{VR}(t_R)$.

In this delegation scenario, Alice supplies to Bob only her hash chain corresponding to `renewable energy` and there exists no efficient way for him to generate any elements from the tree, which Alice didn't delegate. As the Merkle tree data structure is used to define the authorisation area only, delegations can be carried out as described in Section 4.3.3 for standalone hash chains. The period delegation is analogue to the process described above, however a hash chain residing in the `period` subtree needs to be utilised.

Through a combination of these two delegation types, a timely limited delegation on a specified topic can be achieved and I will simulate it with the help of Alice's credentials presented in Figure 4.9. This time she wants to delegate voting rights to

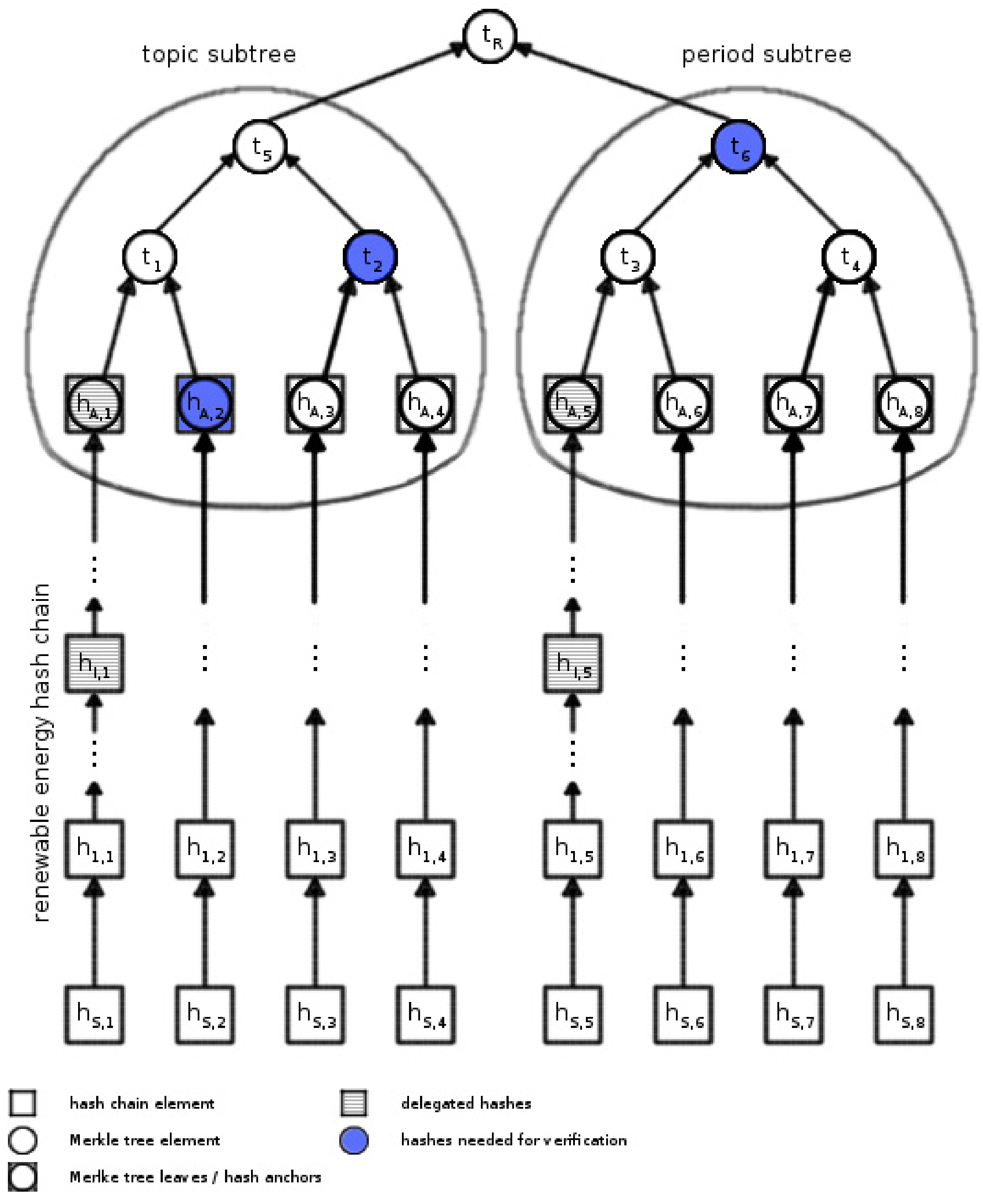

topic subtree
period subtree
t_R
t_5
t_6
t_1
t_2
t_3
t_4
$h_{A,1}$ $h_{A,2}$ $h_{A,3}$ $h_{A,4}$ $h_{A,5}$ $h_{A,6}$ $h_{A,7}$ $h_{A,8}$
renewable energy hash chain
$h_{I,1}$
$h_{I,5}$
$h_{1,1}$ $h_{1,2}$ $h_{1,3}$ $h_{1,4}$ $h_{1,5}$ $h_{1,6}$ $h_{1,7}$ $h_{1,8}$
$h_{S,1}$ $h_{S,2}$ $h_{S,3}$ $h_{S,4}$ $h_{S,5}$ $h_{S,6}$ $h_{S,7}$ $h_{S,8}$
hash chain element
Merkle tree element
Merkle tree leaves / hash anchors
delegated hashes
hashes needed for verification

Figure 4.9 An exemplary hash chain/Merkle tree construct containing four topic and four period leaves. Alice delegates to Bob only the topic chain $(h_{S,1}, h_{A,1})$ and to Dave additionally to it also the period chain $(h_{S,8}, h_{A,8})$. Furthermore, they both receive from Alice the VR signed tree root t_R and the nodes needed to calculate it.

Dave again on the topic `renewable energy`, but only for a limited amount of time. Figure 4.10 illustrates the `period` subtree and the denotes its chains exemplary. To be able to delegate the intended voting rights to Dave, Alice has to execute the following procedure:

1. Generate a delegation hash chain $HC_1 = (h_{I,1}, h_{A,1})$ belonging to the topic `renewable energy`.
2. Identify the set of elements $SET_1 = (h_{A,2}, t_2, t_6)$ needed to calculate t_R with the chain from 1)
3. Generate a delegation hash chain $HC_2 = (h_{I,5}, h_{A,5})$ belonging to the selected period `first validity week`.
4. Identify the set of elements $SET_2)(h_{A,6}, t_4, t_5)$ needed to calculate t_R with the chain from 3)

Supplying Dave with the values HC_1, SET_1, HC_2, SET_2 and $sig_{VR}(t_R)$ allows him to use Alice's voting rights for the validity period specified by the delegated hash chain belonging to the `period` subtree.

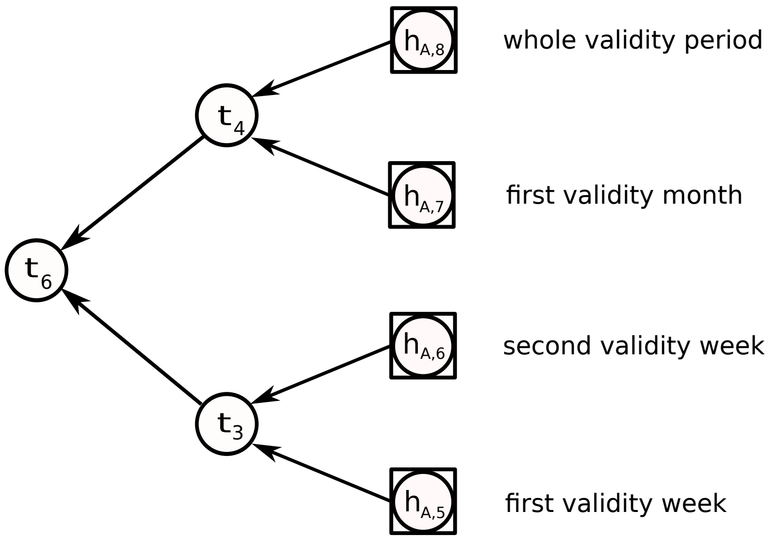


Figure 4.10 A zoomed view of the `period` subtree (see Figure 4.9). The leaf nodes are denoted with the time span, during which the delegation can be used.

The rightmost hash chain is treated differently than the other hash chains in both the `topic` and `period` subtrees, as it represents the maximal delegation rights for a subtree. This means for the `topic` subtree, that the delegatee can represent the delegator for all existing topics, and for the `period` subtree – that the maximal validity time span is being delegated.

4.5 Summary

This chapter presented a system design based on the `LD` paradigm, which allows anonymous and secret elections to be held through an insecure network like the Internet. The architecture of the systems builds upon the constructs hash chains and Merkle trees to enable the needed delegation features. This way a topic or time limited delegation to one or more entities is possible, which can be revoked at any moment. With the help of blind signatures and anonymous proxies the identities of the voters can be disguised, so that there is no possibility to circumvent the voting secrecy, even if both VR and VC cooperate. Through application of public-key cryptography the credentials acquirement and voting processes are protected, so that a manipulation in the system is obvious for every verifier. The next chapter discusses the building blocks of my Proof-of-Concept (**PoC**) implementation.

5

Implementation

This section presents in detail the PoC code to the system architecture described in Chapter 4. It first introduces the python libraries used in the implementation, then gives an overview of its structure and the way the code is separated into packages. At last the unit testing framework developed to verify the behaviour of the code is described.

5.1 Employed Tools

For implementation of the system I decided to use the high-level scripting language `Python`[3]. As it has a clear syntax and intuitive object orientation, rapid software development is possible. It further supports hierarchical packages and allows full modularity, which makes the code easy to understand and maintain. The extensive standard libraries includes implementations of network communication and encryption functions, regular expressions, base64 encoder/decoder, hashing, TLS connection encryption and argument parsing, which allowed me to directly concentrate on my main problems. Third party modules for every protocol I needed are also available. The external library `gdata` supplies the RSA based public key cryptography routines e.g. data encryption, signatures and blind signatures, `M2Crypto` contains the required DHKE code and `pysqlite` the database interface to the SQLite databases I employ. With the help of these building blocks I was able to implement complex data structures like the hash chains and Merkle trees. Different python interpreter exist for all major platforms so the code can be easily tested and further developed on any operating system. Though python also has an important drawback – compared to compiled languages like C, it can be around 100 times slower. Although the low level functions like encryption/decryption and hashing are C libraries and the python code is a wrapper to them, a performance relevant system might have to be written in C.

My development environment consisted of a Gentoo linux machine and the Vim editor. Furthermore the python interpreter in version 2.7 or 2.8 has to be installed on the system and the libraries gdata[1], M2Crypto[2] and pysqlite[3] have to be available for the code to execute properly.

5.2 Proof-of-Concept Overview

To recall the system architecture presented in Chapter 4, Figure 5.1 presents it again, but this time from the viewpoint of the user Alice. There are two types of entities

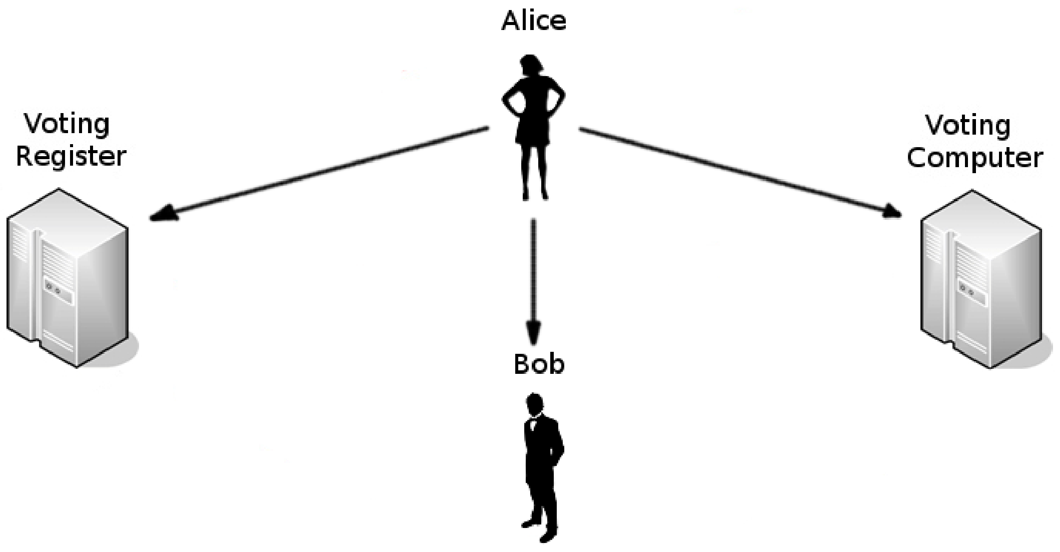


Figure 5.1 System structure and interactions from the viewpoint of the user Alice participating in the voting procedure.

in the concept – the server instances (VR and VC) having a passive role and the voters, who can interact with the servers and eventually with each other. Because of this, the PoC code is separated in different packages. In the following section the structure of these packages will be described.

5.3 Implementation Details

The implementation code is split in two main packages. The first one consists of library files containing the functions required for my design and is situated in the folder `./lib/`. This way code duplication can be avoided, as every program requiring a specific functionality can simply include these libraries and directly use it. The second one supplies the executable python scripts, which can be found in the `./bin/` folder. They use the supplied libraries and form the interface to the actual system. With the help of command line arguments their execution behaviour can be modified.

The structure of the packages and the dependencies between their files is presented in Figure 5.2. Every reference arrow describes a dependency between two subpackages

[1]http://http://packages.python.org/gdata/
[2]http://chandlerproject.org/Projects/MeTooCrypto
[3]http://readthedocs.org/docs/pysqlite/en/latest/sqlite3.html

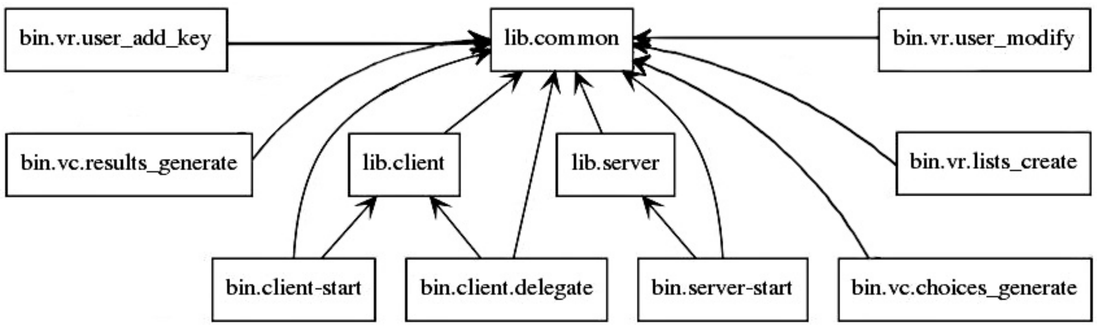


Figure 5.2 The dependency graph of the packages used in the PoC code. The library subpackages `common`, `server` and `client` contain most of the code and are required by the remaining packages.

not necessarily from the same package. It is drawn from the subpackage requiring further code to the one containing it. As we can see, all the name strings contain dots ("."). These are used as delimiter symbols and separate the location from the actual subpackage name. For example, the string **bin.vr.user_add_key** from the upper left corner of Figure 5.2 specifies that the subpackage with the name `user_add_key` is situated in the subdirectory `bin/vr/` of the package containing the executable scripts. The subpackages `client` and `server` in the `lib` directory are libraries implementing the code functions needed in the client and the server instances respectively. As functions like encryption, decryption and signing are used by both server and client, these are implemented in a shared library subpackage `common`. It is then included in further scripts, to allow no code duplication and to ease maintainability.

Figure 5.3 presents the python classes used in the PoC implementation and the functions associated with them. The class named `VoterObject` contains nine functions. With their help an encrypted connection to a remote machine can be established either directly (`connect()`) or through an AP (`proxy_connect()`). The protocol I used in the prototype for connection encryption is `TLS`, as it does both encrypt and integrity protect the data while transfered. Further functions form the `VoterObject` class make interaction with the two types of server entities possible (`interact_vr()`, `interact_vc()`) and allow the result of a voting procedure to be checked for correctness (`check_voting_results()`).

The server library defines two more classes – `ServerObject` and `HandleClient`. The former initialises through the `run()` procedure the server process as a daemon waiting for connections on a supplied port number and configures the TLS interface (`open_socket()`), so that encryption of the traffic is possible. The client connections can then be accepted and handed over to the `HandleClient` class. The `client_status()` function allows the operator to debug problems or generate statistics about the server load with regard to online users. For every incoming connection a new thread in `HandleClient` is started, so that clients can be served simultaneously. Every of these instances is able to handle both connections to VR and VC through the functions `register_server()` and `voting_server()`, though the current implementation sets a variable in the `ServerObject` indicating the type of service to be provided. This means that a server instance for every type has to be started. The functions `data_get()` and `data_send()` are wrappers used to prepare the data for processing or network transmission respectively. The remaining two functions `auth_pkey()` and `check_auth()` serve the purpose of authorising the client on the VR and VC side.

As already noticed, the `bin/` directory contains the executable python scripts. With the help of `client-start.py` the communication with VR and VC can be carried out. Accepted arguments range from action to execute (get credentials, vote,check results) to connection information e.g. IP address and port. The script `server-start.py` is used to start a server instance through creating a `ServerObject` (as described above).

5.4 Autonomic Testing

With the help of the python scripts `client-start.py` and `server-start.py` I was able to set up two different testing scenarios. In the first all instances (VR, VC and client) ran on the same machine, in the second - all were started on different machines. To be able to run the tests a way of initialising the databases and setting up the user information is needed (recall Chapter 4). For this purpose the following helper scripts under `bin/` are available:

- `cleanup` – A shell script that cleans up all data from the test instance.
- `client/delegate.py` – Take as input a voting credentials block and allow validity check and delegation from it.

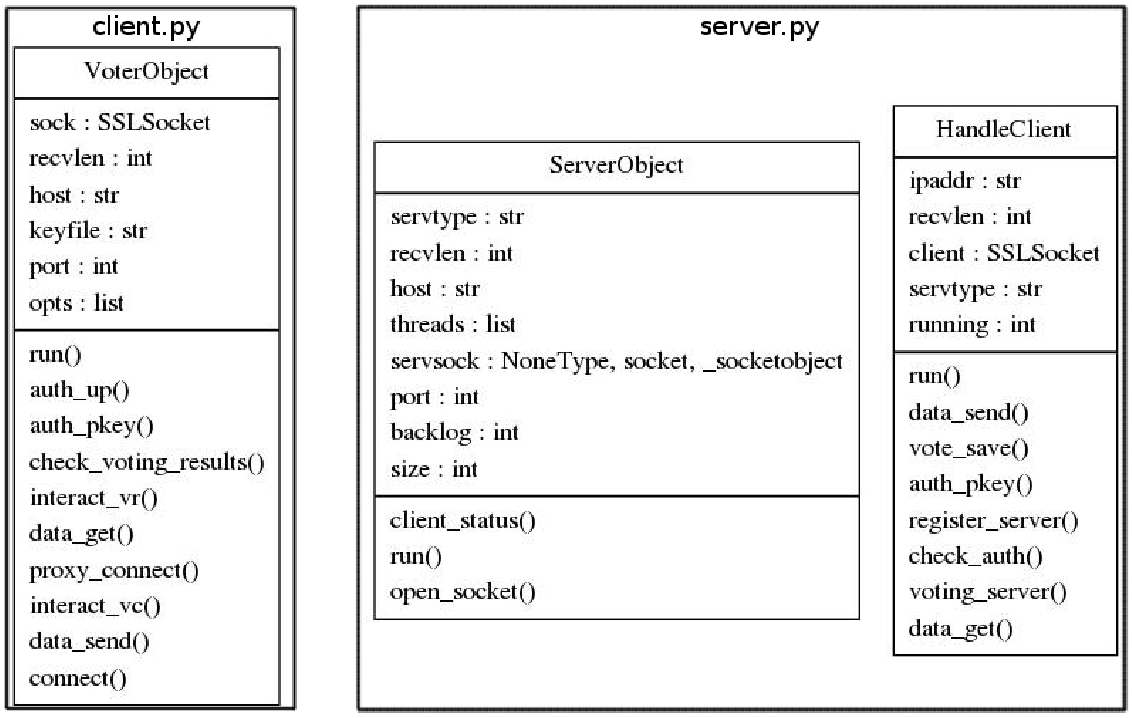


Figure 5.3 Python classes and their associated functions from the subpackages `server` and `client`. The subpackage `common` contains only functions, which are shared between both server and client scripts, and has no classes.

- `vr/user_modify.py` – Add user's information to the SQLite database, describing who is allowed to take part in a voting procedure.
- `vr/user_add_key.py` – Generate asymmetric RSA key for a user and add its public key part in VR's database.
- `vr/lists_create.py` – Generate the lists to be published as described in the design chapter (Chapter 4 , Section 4.3.1 and 4.3.5).
- `vc/choices_generate.py` – Generate the list with voting options.
- `vc/results_generate.py` – After the voting procedure ends, generate the lists allowing the public voting check.

To facilitate easy testing of the system, I wrote a simple unit tests package consisting of Bash[4] shell scripts. It can be found in the folder `unittests` in the repository base and contains the following files:

- allinone.sh - Executes all unit tests with default values, no user input needed.
- check-vc.sh - Carries out a voting through the client interface using the voting credentials supplied as command line argument.
- check-vr.sh - Acquires credentials from the VR using the client interface.
- declarations.sh (*) - A library containing variable and function definitions.
- delegate.sh - Carries out a delegation and allows the validity check of delegated tokens.
- prepare.sh - Deletes all content in the local databases and initialises both VR and VC for a new voting round.
- servers.sh (*) - Contain function definitions allowing to check if the VR and VC server are running, to start them as background processes if not, or stop them if desired.

The two files marked with an asterisk ("`*`") are not meant to be directly executed, they contain variable and function definitions required by the remaining shell scripts.

The file named `allinone.sh` allows a voting procedure to be carried out without any user input. For this purpose the rest of the files in the `unittests` folder is called in a predetermined sequence and default arguments are used as their input. It is possible to check only a given part of the code. To run the tests on the delegation procedure for example, the file `delegate.sh` has to be called with the appropriate command line options.

The next thesis chapter discusses in detail the characteristics of my system, e.g. verifiability, performance and scalability. In addition a feature comparison with the implementations presented in Chapter 3 is done to make the advantages and disadvantages of the here presented system even clearer.

[4]http://www.gnu.org/software/bash/manual/bashref.html

6 Evaluation

This chapter analyses the system design (Chapter 4) and the PoC implementation (Chapter 5) with regard to security, reliability, performance, distribution and scalability. Furthermore some details on the delegation based on Merkle trees will be discussed. To summarise the results of the thesis, I will compare in tabular form my own implementation with these presented in Chapter 3.

6.1 Manipulation detection probability

The verifiability of the design presented in Chapter 4 bases on the availability of the voting results, which are published by VC. Hence all participants are able to check, if their voting credentials and votes are correctly recorded. Furthermore, with the help of the lists published by VR containing the voting credential requests from all participants in combination with their corresponding public keys, everyone is able to verify that all requests came from valid voters and none were falsified by the VR. As even paper based ballot systems are prone to errors and manipulation[17], the important question here is how prone to manipulation my design actually is. As it is unlikely that all participants check their submitted votes, no 100% reliability can be expected. This section analyses the number of voters in percent needed to check their votes or credentials, so that it is highly improbable for a manipulation in the system to remain undiscovered.

6.1.1 Calculation Methods

Lets assume an imaginary example, where the voting community consists of 100.000 participants, the number of verifiers (vote and credential checkers) is 1.000 and exactly 100 votes or credential requests (both VR and VC could cheat) are manipulated. The experiment is then equivalent to a drawing from an urn without replacement, where from the $N = 100.000$ marbles, 100 are black ($B = 100$), the remaining are white ($W = 99.900$) and a number of $n = 1.000$ draws are made. Thereafter is the probability p_B of drawing a black marble is:

$$p_B = B/N = 100/100.000 = 0,001 (= 0,1\%) \tag{6.1}$$

As this probability distribution is discrete, the following holds:

For the calculation of the probability normally the hypergeometric distribution[1] function is used. Though as B is negligibly small compared to N, the easier to calculate binomial distribution[2] can be used instead. For a maximally x successes in n draws, the binomial function is defined as follows:

$$F(x; n, p) = P(x \leq X) = \sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i} \tag{6.2}$$

As already stated above, this is a discrete probability distribution, so the following equation holds:

$$\sum_u P(X = u) = 1 \tag{6.3}$$

The reason for this is, that the sum of the probabilities for a variable u running through all possible values for X has to be 1 (or exactly 100%). Consequently we can simplify equation (6.2), calculating the probability of `no modification being detected`. Subtracting this value from the probability `1` leads to the same values as in equation (6.2). Thus the formula we will use for all further calculations has the following form:

$$\begin{aligned} F(x; n, p) &= P(B \leq X) = 1 - P(k = 0) \\ &= 1 - (\binom{n}{k} * p_B^k * (1 - p_B)^{n-k}) \\ &= 1 - (\binom{n}{0} * p_B^0 * (1 - p_B)^{n-0}) \\ &= 1 - (1 * 1 * (1 - p_B)^n) \\ &= 1 - (1 - p_B)^n \end{aligned}$$

6.1.2 Real Life Examples/Showcases

Replacing the variables with the values from the example above results in the detection probability of 63,23% by a vote manipulation of 0,1% and a verifier quota of

[1]The hypergeometric distribution describes the probability of k successes in n draws from a finite population of size N without replacement

[2]The binomial distribution describes the probability of k successes in n draws from a finite population of size N with replacement.

1%. This shows, that even for a small amount of manipulated votes, the probability of detection is well over 50%. Though an interesting real life showcase is calculating the probability of detection, if an entity tries to manipulate the results so, that one whole seat in the elections for the German parliament is given to a non-existent party. Using official data from the elections in 2009, the parameters are defined as follows:

- $N = 44.000.000$ – Number of voters, who have taken part in the election.
- $B = N/620 = 71.000$ – Voters per parliament seat.
- $p_B = 0,001613636$ – Probability of drawing a manipulated vote or credential request.
- $n = variable$ – The number of checkers is used as a second parameter.

The results are presented in Figure 6.1. Even by a marginal amount of 1.000 verifiers (≈ 0,0023% of all votes),the probability of detecting at least one of the manipulated 71000 items is approximately 80%. Though it is also interesting, if the design copes so well with other numbers of manipulated votes or credential requests. Figure 6.2 compares the results (≈ 70.000) from the election example above with three more numbers of modified votes (20.000, 40.000 and 100.000) to show the resilience of the system.

Even for values three times lower than the election example, about 10.000 verifiers (≈ 0,023% of all voters) are enough to reach a probability of nearly 100%. This shows, that the system is well designed with respect to detecting manipulations in the voting procedure.

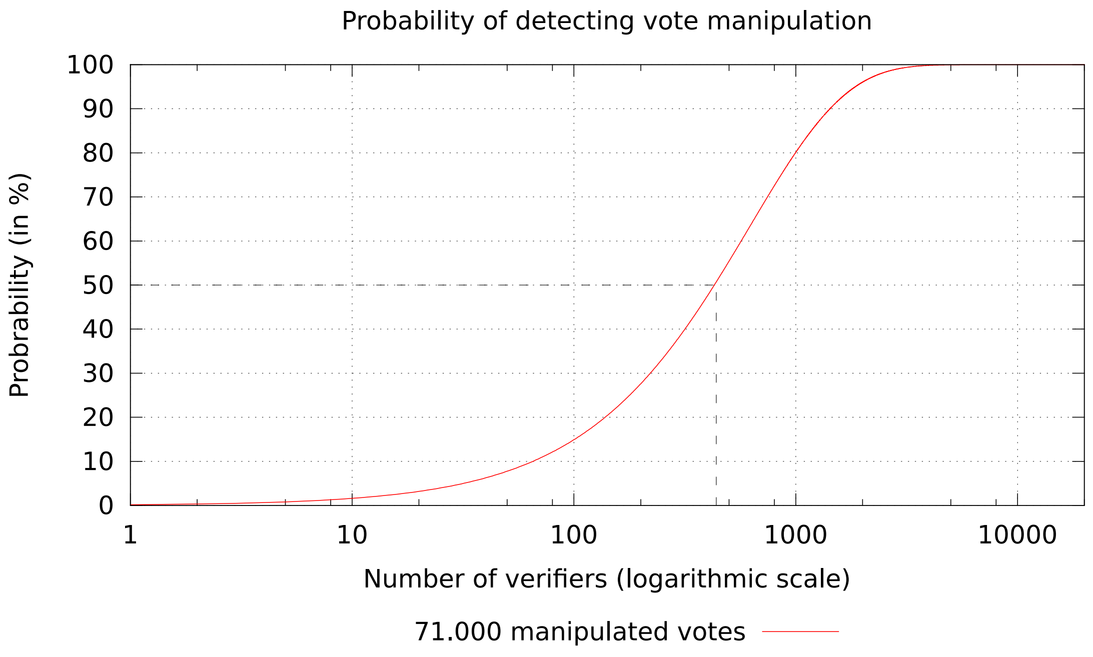


Figure 6.1 Manipulation detection probability for a constant number of manipulated votes (71.000) and variable number of result verifiers.

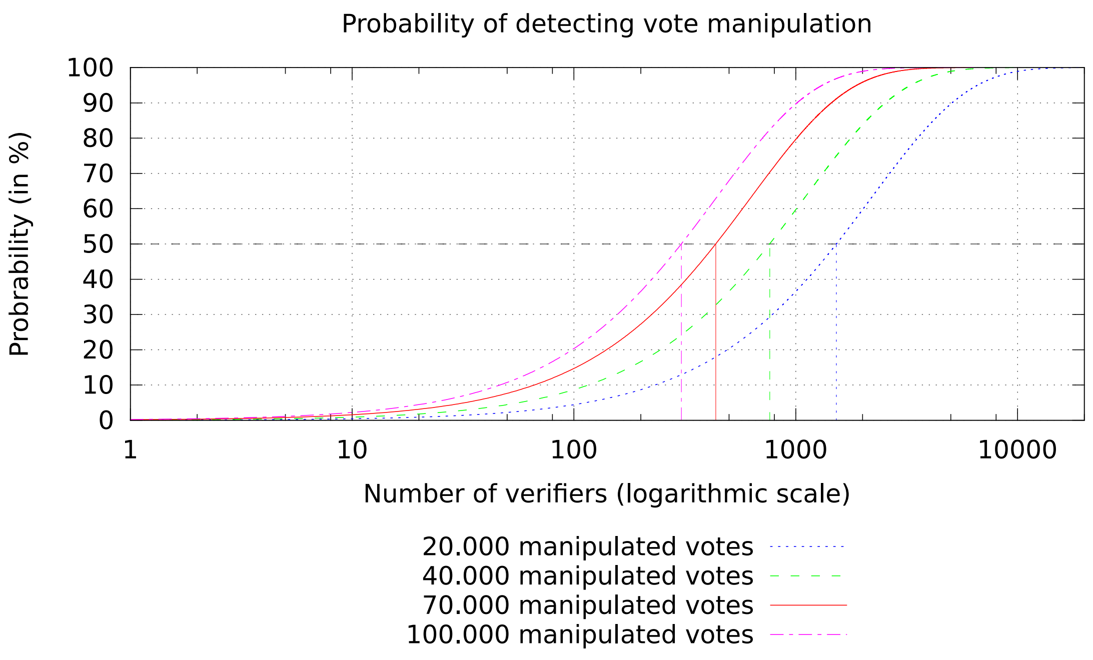


Figure 6.2 Manipulation detection probability for 4 different constant numbers of manipulated votes (20.000, 40.000, 70.000 and 100.000) and variable number of result verifiers.

6.2 Ballot Security/Reliability

The VR and VC instances could be run on the same server, but then the following problems occur:

1. A single point of failure (e.g. DoS, hardware problems).
2. One successful attack on the shared server would compromise the complete voting process.
3. As only one system operator has access to all information, statistical methods could be used to analyse voters' behaviour on-the-fly.

Running the two instances on separate systems makes the system reliable against DoS and hardware problems. Furthermore a distribution of the VR and VC instances is possible, as it reflects the way elections are organised nowadays. Having a VR responsible only for a small part of the citizens (for example a city or an urban district) allows a much more reliable system to be created. The same is possible for VC instances, as long these are time synchronized to avoid cheating.

RSA Key Security

As attacks on the user's system make it possible to steal the RSA private key, a smartcard can be used for the purpose of assuring private key security. There is no possibility to extract data from the internal memory of such a card without destroying it. Hence copying or stealing the private key will be detected by the card owner. As a result the public key saved in the register list can be revoked, so nobody is able to vote with this `stolen identity`.

The new German ID card offers a functionality to generate digital signatures on behalf of its owner and is a smartcard. This functionality can be used instead of the VR identification procedure based on RSA encryption and would not require utilising additional asymmetric keys. In this case the card reader in use has to be a trusted one. It has to be taken into consideration, that the German ID card already had a couple of major security problems, which are discussed in [14] and [8]. Therefore, before either a stand-alone smartcard or the new German ID card can be adopted to protect the security of the authentication process, it should be thoroughly tested.

6.3 Discussion on Merkle Trees

Section 4.4 presented an improvement of the basic design through the use of a Merkle tree structure, which allows time or topic delegation to be carried out. Nevertheless the use of this data structure exhibits also drawbacks. Using a single hash chain for delegation requires only two tokens (voting hash and signed anchor) to be transfered to the delegatee. In the case where a Merkle tree is combined with hash chains to allow multiple delegation features, a larger amount of data has to be transferred to the vote recipient. Furthermore, the validity verification of voting credentials and the decision, if these are used to outvote an already submitted ballot, requires more calculations by the VC. For the number of $D = 2^n - 1$ tree nodes exactly $n + 2$ values – 2 for the selected hash chain and n for every level in the tree – are needed to generate the tree root and compare it with the by VR signed value. If more than one hash chain (topic or time signifier) is to be delegated or verified, more nodes and respectively more processing power are required. The exact number depends on the position of the hash chains in the tree. Although common nodes (covered by more than one generation path) don't have to be transferred multiple times, the worst case scenario requires $n + 2$ data sets for every hash chain delegated. For this reason, compared to the hash chain delegation only design, a higher performance system for the VC is needed. A detailed discussion on the system performance follows in Section 6.5.

6.4 Possible Weaknesses

Even with the improvements described in Chapter 4, Section 4.4 the system design still contains possible weaknesses, which will be discussed in this section.

6.4.1 Vote Selling or Coercion

The improvement in Chapter 4, Section 4.4 addresses the possibility of vote selling and coercion prevention. Although the use of a trusted third party complicates both of the above mentioned actions, it does not prevent them completely. If an insider from one trusted third party cooperates with an extortioner or vote buyer, the system suffers the same attack types as if no trusted third party was used. In addition the voters don't get direct feedback, if the vote is counted or not. VC has access to a large number of (h_S, h_A) pairs, which it receives in the voting procedure. If it is in possession of a hash, that can be used to generate the hash used in one submitted vote block, VC is able to manipulate the voting by outvoting. As the trusted third party is not the real owner of the votes, it does not know how long any of the used hash chains really is. As a result the outvoting procedure is not recognised as cheating.

6.4.2 Hash Chain Delegation Priority

The way delegation priority is defined gives rise to two problems in the design. First of all, collisions in the hash function are possible albeit unlikely, so it is imaginable, that two different participants might have the same value for their anchor elements of the hash chain or even the same hash chain. This can be circumvented by adoption of a hash function with a low collision probability or a large co-domain. The second problem is that over proxy voting the same hash chain can be generated for two delegatees. A simple example is the case when Alice uses the same depth for both delegations in the example described in Chapter 4, Section 4.3.3. This can of course happen in different branches of the delegation graph making this phenomenon hard to foresee. As it is an unwanted side effect of the proxy voting and happens only in cases with complex proxy graphs (a high number of branches raises the possibility of collisions) the problem is handled on VC's side - only the submission, that happened earlier in time, of duplicate voting credentials is counted as valid. Furthermore as a future work a delegation scheme can be developed, which minimises the probability of collisions.

6.5 Performance

In this Section the performance of individual building blocks will be described. For all the tests the same system will be used. It has 3 GB of RAM and an Intel Core 2 Duo CPU with 3MB layer 3 cache. The operating system is a Gentoo[3] linux with the kernel version `3.1.1` and uses the python `2.7.2` interpreter.

For measurement of the execution time, the `clock()` function from the `time` python module will be used. On Unix systems it returns "the current processor time as a floating point number expressed in seconds. The precision, and in fact the very definition of the meaning of "processor time", depends on that of the C function of the same name, but in any case, this is the function to use for benchmarking Python or timing algorithms."[2].

For procedures involving more than one party, additional machines on a LAN with low latency will be used, though all the measurements are taken on the testing machine described above.

[3]http://www.gentoo.org

6.5.1 Hash Chain as Voting Credentials

As described in Chapter 2, Section 2.4.2, to generate the next element in a hash chain, the current element is hashed. Hence it is clear, that there is no way to parallelise this process. To calculate the execution time of a single operation of the hash function `sha512`, I used the following routine:

```
def perf_hash_chain(sval, depth):
    ts = time.clock()
    shash = hashlib.sha512()
    thash=hashlib.sha512(sval)
    for x in xrange(depth):
        thash = hashlib.sha512(thash.hexdigest())

    thash.hexdigest()
    return time.clock()-ts
```

For the calculation of the execution time I used a `depth` value of 1.000.000 and a repeated the measurement 100 times. The resulting average value for a single hashing operation on my testing system amounts to $4\mu s$ with a deviation of $\pm 0,1\mu s$. Hence even for a hash chain consisting of 1.000 elements, 500 (2x250 because of the dual core) hash chains can be calculated in one second. As the clients need to calculate only one hash chain, and the VC will be a dedicated server with enough resources, neither the generation nor the verification are a performance bottleneck.

6.5.2 Merkle Tree as Voting Credentials

Using a Merkle tree for the vote delegation is the second interesting aspect in respect of performance. My implementation uses again the `sha512` hash function to generate the tree nodes, so the same time is needed for a single operation. Though compared to a hash chain with a flat structure, a Merkle tree with height of 10 has exactly 1.024 leaf nodes and a total of 2.047 elements. As every node is generated with the help of the hash function, the lower limit for the execution time of a routine creating a Merkle tree is:

$$2.047 * 4\mu s \approx 8,2ms(deviation : \pm 0,2ms) \tag{6.4}$$

The actual testing results show that the average execution time of a function generating the tree is $\approx 10ms$ with a (deviation of $0,2ms$), as there are also concatenation and memory allocation operations, which weren't considered in the above estimation.

This amount of time is needed for the Merkle tree generation exclusively. Assuming that the user is to generate the complete vote delegation structure consisting of a tree with the height 10 (1.024 leaf nodes) and the corresponding hash chains with the depth of `1.000`, the execution time needed is roughly:

$$10ms + 1024 * 4ms \approx 4,1s(deviation : \pm 0,1ms) \tag{6.5}$$

To verify the correctness of the above calculations, I used the following python code:

```
def gen_tree_row(row):
    nextrow = []
    for i in range(0, len(row), 2):
        nextrow.append(hash_calc(row[i]+row[i+1]))

    return nextrow

def perf_merkle_tree(leaves):
    while (len(leaves) > 1):
        leaves = gen_tree_row(leaves[len(leaves)-1])


pref_merkle_tree([perf_hash_chain('',1000) for i in range(1024)])
```

It uses the previously defined function `perf_hash_chain()` and generates the Merkle tree recursively row by row until the root is calculated. The measurement is again repeated 100 times in a row and the average is taken. On my testing setup the approximate of $3,8ms$ was required for the code to execute. In spite of this high number, this procedure is performed only by the client. The VC has to only verify that the credentials supplied to it are valid. For this only the hash chain used for voting has to be verified, meaning that the maximal amount of operations is `1.000` (in case of no delegation) or lower. Furthermore the tree verification has to be proceeded, but it requires an amount of operations proportional to its height, thus `10` in this example. Assuming both topic and time delegation was used (two hash chains delegated) the worst-case execution time is:

$$2(10 * 4\mu s + 4ms) \approx 8ms \tag{6.6}$$

This shows, that even a slow desktop computer like my dual core testing machine is able to perform more than 250 voting credential verifications in one second. This performance measurements and calculations show, that not even the deployment of the Merkle tree based structure should present a bottleneck in my design.

6.5.3 Blinding and Signing Procedures

The RSA public key routines for blinding, signing and signature verification supplied by the python library `gdata` play an important role in the my system implementation. Therefore it is important to know how they perform to be able to estimate the resources needed for the server instances VR and VC. As the implementation employs keys with the length of 2048 bit, this length will be used for the tests.

The blinding and unblinding procedures are the basis of the anonymity in my system. Although they are both performed only on the client side, it is important to know their execution time. RSA signatures allow on the one hand verification, if every vote issued by the VR is legitimate and on the other hand, if credentials supplied to the VC are really issued by the VR. Thus both VR and VC have to issue verification procedures, whereby VR needs to also sign a blinded hash for every valid client.

To check the runtime of both (blinding,unblinding) and (signature,verification) operation pairs, the following scenario is used:

1. generate a random value BSEC (blinding secret)
2. blind the HASH with the help of an RSA public key and BSEC
3. sign the blinded value with the RSA private key
4. strip the blinding to receive a valid signature on HASH
5. verify the signature on HASH

This way all relevant execution times can be measured at one. The python code used for this performance check is available in the appendix. Every single function is executed repeatedly and only the averaged execution time and the deviation are presented here, so that the results are more accurate.

The blinding, unblinding and signature verification operations requires 265, 70 and $185\mu s$ respectively with a deviation in the interval $\pm 1\mu s$, which is fast for functions operating on asymmetric keys. The signing operation is the slowest of them and has an execution time of $3,645ms$, having a deviation of $\pm 0,02ms$. In spite of this fact, my system was able to generate more than 400 signatures for a second. So even this comparably expensive function doesn't represent a bottleneck in my implementation.

Next I will examine the interaction procedures with VR and VC as a whole to give you a better idea of their execution time and the load they generate on my testing machine.

6.5.4 VR and VC Interaction

The previous three sections discuss the performance of the separate functions used in the system design presented in Chapter 4. Though their analysis doesn't include the execution time needed for memory management and IO operations needed by the program. So in this section the execution time of the PoC code and the load on the test system will be measured.

First the execution time of the credentials acquirement procedure is measured. Generating a hash chain with the depth of 1024 as a voting credential occupies the processor on the client side for 0.14 seconds on average. In contrast, the execution on the server side requires half the time and is completed in approximately 0.07 seconds. The memory occupied by both of the processes is under 0.5% and the load temporarily jumps to 5% during the interaction. A simulation with 20 parallel clients using the server, the load for the server jumps to 15% not affecting the amount of RAM needed. The execution time has only a slight variation of 0.01 seconds more compared to the case of a single client connection. In the case of a Merkle tree used as voting credentials no difference can be determined, as the tree is generated entirely on the client side and the server sign only one element, which is comparable to the signing of the chain anchor.

Next the voting procedure is timed, where the hash chain is used as a whole – no delegation, use of the starting hash for voting. On the VC an execution time of 0.15 seconds in average and around 0.5% memory usage could be measured. On the client side, the python script is running under 0.1 seconds and uses about the same amount of memory as the server process. For 20 simultaneous voting connections, the peak load reaches 30%, but again no impact on the used RAM can be determined. As the complete depth of the hash chain is used for the tests, this is the scenario with worst-case load and execution time. In the case of a Merkle tree as voting credentials, the length of a path from a tree leaf to the root node is negligible compared to the depth of the hash chain(s) used for delegation. Because of this, no extensive impact on the performance is to be expected. Furthermore, the delegation procedure arranges for reducing the depth of the hash chain and the amount of hash operations needed to generate the from VR signed significant element. With a well-thought division scheme for the hash chain both more flexibility for the user and higher system performance can be achieved.

6.5.5 Input/Output Costs

IO operations can easily cause a bottleneck in a system, where much information is to be read or written to disk. The PoC implementation currently reads the RSA keys from disk and stores the data in local SQLite[4] databases. The former is done only once by starting the corresponding server and should therefore have no relevant impact on the performance of the system. The latter can easily be diminished by replacing them with relational databases (SQL, e.g. MySQL[5]) and storing the tables in RAM or deploying structured storages (NoSQL, e.g. BigTable[6], Memcached[7]) allowing rapid read/write operations. Currently the amount of time for IO operations is around 40% of the wall clock execution time, though this is not directly reflected in the measurements presented above. This is due to the fact IO is handled with the help of DMA[8]. As the purpose of the implementation is to show, that the system concept is usable, I intentionally decided against a complex data storage system.

As we have seen, the voting acquirement procedure does Using the calculations and measurements from above, my testing system should be able to handle approximately 500 simultaneous voter connections on full capacity. For an election duration of a 12 hours and assumed that the ballot procedure needs 6 seconds[9] to complete, the maximum amount of 3.600.000 submitted votes can be reached. A realistic number will be in the range 400.000 to 600.000. With a high-performance CPU, more cache and RAM the performance of the system can at least be doubled. So it should be feasible, that roughly 1.000.000 voters can submit their votes using one such VC instance.

6.5.6 Voting Lists Verification

As the resilience of the system against manipulation relies on the fact, that the voters verify their own vote and the consistency of the overall voting results, it is important to know how much time such an verification procedure requires on a

[4]http://www.sqlite.org
[5]http://www.mysql.com
[6]http://research.google.com/archive/bigtable.html
[7]http://memcached.org/
[8]http://en.wikipedia.org/wiki/Direct_memory_access
[9]This assumption includes network delay, possible thread waiting, RAM swapping and so on.

voter's computer. To be able to calculate the execution time, the following relevant actions need to be timed:

1. Read in the file containing the tuples `(signed_nonce,public_key)` for every user, who has acquired voting credentials, and verify the signatures. (generated by VR)
2. Read in the file containing the quadruples `(vhash,anchor,signature,depth)` (voting credentials information), check the signature on `anchor` and verify that `vhash` generates `anchor` in `depth` steps. (generated by VC)
3. Read in the voting results file containing the tuples `(nonce,vote)` and verify that your own pair appears in the list. (generated by VC)

The operations employed are reading a flat file of data, checking an RSA signature and generating a hash chain with a specified depth. The execution time of the signature verification operation is $185\mu s$ and of the hash chain verification with a depth of 1.000 is roughly $4ms$, as we've already seen in Section 6.5. The only execution time currently unknown is the file read-in operation. I used a test scenario, in which a file with 100.000 lines having the form

```
VHASH,ANCHOR,SIGNATURE,DEPTH
```

is read line by line and the four elements are extracted to a python `list`. Next, two comparison operations between `VHASH` and `ANCHOR` are used to approximate the tests needed in the PoC implementation, where both the validity of the signature and the equivalence of the generated and supplied anchor need to be checked. The execution time of this whole routine is $500ms$, resulting in an execution time of $5\mu s$ per line. The following listing presents the summed up execution time per input line for each of the verification actions presented above:

- action 1) — $5\mu s + 185\mu s = 190\mu s$ (line read and signature verification)
- action 2) — $5\mu s + 4ms + 185\mu s \approx 4,2ms$(line read, signature verification on anchor, vhash to anchor generation)
- action 3) — $5\mu s$ (line read, verify if the own nonce and vote are found)

These times reflect the worst-case execution of the operation 2), as not all the votes will use the complete length of the hash chain in the actual voting process.

Let's recall the values of the last German parliamental elections, as used in the Section 6.1.1, to get a realistic usage example. The number of participants was 44.000.000 and this is the number used to calculate the time needed for the verification procedure. All three files have 44 million lines, so the following listing summarises the runtime of the actions 1) to 3) on a single CPU machine:

- action 1) — $190\mu s * 44.000.000 \approx 2,33h$
- action 2) — $4,2ms * 44.000.000 \approx 51,34h$

- action 3) — $5\mu s * 44.000.000 \approx 3,34m$
- altogether — $\approx 54,11h$

The procedure consists of many checks, which are line-independent and therefore allows its parallelisation to minimise the execution time. Even the simple approach of splitting the second list (the long running verification) in equivalent parts and starting so many checker scripts, as the number of CPUs of the verification system, scales good and would allow my dual core testing machine to verify the overall voting results in approximately **27 hours**.

6.6 Distribution and Scalability

The structure of my system allows the straightforward separation of the VR and VC instances, as these do not share any data by design. Furthermore, the VR instance can be deployed as separate VR instances, each responsible for only a disjunct part of the voting register. Removing the disjunction requirement would lead to a necessity of data synchronisation – not considering this would allow users to acquire multiple voting credentials from different VR instances (cheating). With this requirement in place, the large amount of computational power and data exchange overhead can be eliminated.

Unfortunately there is currently no way to distribute the VC into separate instances without the use of synchronisation. If this is done, there is no way to determine the temporal order of incoming votes, which is required because of the collision issues by the delegation, as described in Section 6.4.2. Nevertheless, only time synchronisation of all the VC instances is required and absolutely no data has to be exchanged between the different instances. After the closing of a voting procedure, all possible collisions can be solved off-line.

The clock synchronisation with an external source can be done with the help of different protocols (e.g. NTP[10], PTP[11], GPS[12]). To decide which one suits best, the required accuracy has to be defined, which on the one hand depends on the voting procedure execution time, on the other hand has involve considerations of the network latency and architecture.

Fulfilling the requirements for distribution of both servers allows a straightforward horizontal scaling[13] of the voting infrastructure. Using the summarising results from the previous section as a point of reference of the performance of a single VC instance, we can easily calculate the number of instances to be added, for a given number of participants.

[10]http://tools.ietf.org/html/rfc5905
[11]http://www.nist.gov/el/isd/ieee/ieee1588.cfm
[12]http://www.gps.gov/systems/gps/
[13]"To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application. An example might be scaling out from one Web server system to three."[4]

6.7 Comparison with other Implementations

In Chapter 3 common implementations of the LD concept were described. In the following I will compare them with the design and implementation described in this thesis. The features, both common and different, of these systems are summarised in Table 6.1.

Let's first concentrate on the LD aspects. All of the systems support delegated voting, outvoting and publish their results online. Apart from Votorola, they additionally allow multiple, topic and global voting delegations. A negative aspect of my work is, that it doesn't present any possibility of consolidated decision making, as no forums or other discussion media are presented. No proposals, initiatives or drafts can be used as a tool in the implementation, which is a standard for all the remaining systems.

Looking at the voting procedure, none of the systems besides mine supports neither receipts in any form nor real anonymous voting, which makes it impossible to definitely prove system manipulation. Albeit they provide anonymity from the voter's view, everyone with access to the database can find out, what a participant voted for (no real secrecy and anonymity). In addition, no cryptographic security in form of connection encryption or integrity protection is implemented in any of the other systems by default. Adhocracy and LiquidFeedback tough allow the deployment of TLS on the web server, where the software is running. This would at least complicate MitM attacks. If the verification behaviour of voters is analysed, which is possible because of the missing anonymity, selective vote manipulation by users generally not verifying is possible. As my system features anonymity and high probability of manipulation detection based on probability, such attacks are impossible.

Considering the features resulting from the system design, used programming language and possible additional software, all the systems perform relatively well. Both Adhocracy and Votorola use a modular process structure, allowing simple reconfiguration of their internal workings. In contrast both LiquidFeedback and my implementation have a fixed structure and event order, which has to be followed for a voting procedure to complete successfully. Based on the design, Votorola and my own implementation directly allow the voting infrastructure to be divided into a large amount of autonomous systems (distribution), hence a more resilient and high-performance infrastructure is to be expected. Nevertheless, my current implementation uses SQLite databases to store the results, which should cause a major performance bottleneck.

To bring this chapter to a conclusion, my implementation facilitates a large number of features and focuses on the secrecy, anonymity and security of the voting process. Although the overall design performs well, there are some bottlenecks in the PoC code. These can be diminished by replacing the SQLite databases with a performance oriented database system.

6.8 Legal Compliance

The system design as well as the implementation presented in this thesis fulfill the requirements 1) to 10) specified in Section 4.1. Although the possibility of vote

	Adhocracy	LiquidFeedback	Votorola	My System
LD concept:				
multiple delegation	yes	yes	no	yes
topic delegation	yes	yes	no	yes
time delegation	no	yes	no	yes
global delegation	yes	yes	no	yes
outvoting option	yes	yes	yes	yes
discussion possibilities	yes	yes	yes	no
support drafts/initiatives	yes	yes	yes	no
Voting Procedure:				
public results	yes	yes	yes	yes
voting receipt	no	no	no	yes
modification detection	low	low	low	very high
anonymous/secret ballot	no	no	no	yes
cryptographically secured	no	no	no	yes
System Features				
programming language	python	lua	java	python
performance	good	medium	good	medium
modular structure	yes	no	yes	no
distributed setup	no	no	yes	yes

Table 6.1 Comparison between Adhocracy, LiquidFeedback, Votorola and the System presented in this Thesis Implementation

selling/coercion is not completely diminished by the improvements presented in Section 4.4, the same problem is presented in the postal voting mechanism conforming to German law regulations. Therefore it is to be assumed, that the system presented in this thesis should satisfy the regulations in the German Constitution with respect to voting.

7 Conclusion

In this thesis presented an architecture and a proof-of-concept implementation of a system, which mainly focuses on the anonymity and security of a liquid democracy based voting mechanism. It can be used over an insecure network like the Internet, issues credentials to the voters, allows vote delegation to one or more parties (liquid democracy requirement), issues receipts for the votes and allows public verification of the voting results (see Chapter 4, Section 4.1).

The security of the system is provided through the use of encryption/decryption and signing/verification mechanisms based on RSA public-key cryptography. RSA blinding functions are used to provide anonymity in the credentials acquirement process from the Voting Register (`VR`) and anonymous proxy servers can be applied to disguise the voter's identity to the Voting Computer (`VC`). Cheating by VR or VC is further countered by adding an anonymous message board and commitment protocols to the system. This way it easily can be detected and proven by voters verifying their votes, that one of the server instances tried to manipulate the voting results.

To facilitate the vote delegation features – simple and multiple proxying, time and topic based delegation and outvoting – I deployed hash chains and Merkle trees. Both of these constructs are generated with the help of a cryptographic hash function, which is further used to link the user's ballot with the receipt of the chronologically previous ballot, so no retrospective manipulation in the ballot content or order is possible. The system provides publicly available voting results, where every participant is able to verify the own vote, meaning that the detection of voting manipulation bases on statistical probability. As the evaluation shows, the system is highly resilient against cheating. Even for a small percentage (around 0,1%) of verifiers the probability of detection is very high (around 99%). The design furthermore allows distribution of the VR and VC instances, allowing a good horizontal scalability of the system. Nevertheless, based on test cases and calculations from Chapter 6, the system performs reasonably well even on a normal desktop system not specifically designed for high performance.

The comparison with related implementations showed, that my system is superior to the remaining ones in behalf of security, anonymity, secrecy, verifiability and cheating detection.

Despite the high security of the system, the design can't completely neutralise the vote selling and coercion threats, which result from the voting delegation feature paired with the publicly available voting results. However, the system should be conforming to the German constitutional regulations on voting procedures, as described in Section 4.1.

7.1 Future Work

The system design focuses on security and anonymity of the voting process, though still has weaknesses in respect to vote selling/coercion. The following list presents possible future research direction, which could help solving these problems:

- Although a possible solution of the vote selling/coercion problem was presented in Section 4.4, it is not feasible if the voting results are publicly available. Using a third party to verify the validity reduces this problem, but requires the voters to fully trust this entity. Therefore, to enhance the system design, ways of mitigating this weakness/problem have to be examined.
- As there is no guarantee of security, formal methods for proving the security of the used protocols should be examined.
- A performance oriented database system should be chosen to replace the SQLite databases used in the proof-of-concept implementation and an interface for it should be implemented.
- A delegation scheme should be developed, which minimises the risk of collisions in the delegation process, so no priority problems arise on the VC side.
- A more realistic performance test with a real network setup and a large number of participants should be performed. This way the server performance needed to handle the loads resulting from a voting procedure can be determined.
- Tools for consolidated decision making should be incorporated in the design of the system and implemented as a modular part of it. This surely would make the system more attractive, as the comparison in Section 6.7 suggests.

Bibliography

[1] Federal electoral law. http://www.iuscomp.org/gla/statutes/BWG.htm.

[2] Python documentation. http://docs.python.org/.

[3] Python website. http://python.org/.

[4] Scalability - scale out. http://en.wikipedia.org/wiki/Scalability.

[5] What is liquid democracy? http://adhocracy.de/static/about.html.

[6] Win-win game. http://en.wikipedia.org/wiki/Positive_sum.

[7] Federal constitutional court website. http://www.bundesverfassungsgericht.de/pressemitteilungen/bvg09-019.html, 03.03.2009.

[8] eperso kann remote missbraucht werden. http://janschejbal.wordpress.com/2011/08/08/eperso-kann-remote-missbraucht-werden/, 08.08.2011.

[9] *Password Authentication with Insecure Communication* (1981), Communications of the ACM.

[10] Voting manipulation in dachau. http://www.spiegel.de/politik/deutschland/0,1518,232662,00.html, 2002.

[11] *A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL* (2006), University of Maryland, Department of Mathematics.

[12] Electoral management body website. http://www.bundeswahlleiter.de/de/glossar/texte/Briefwahl.html, 2009.

[13] *Renegotiating TLS* (2009), PhoneFactor, Inc.

[14] Praktische demonstration erheblicher sicherheitsprobleme bei schweizer suisseid und deutschem elektronischen personalausweis. http://www.ccc.de/de/updates/2010/sicherheitsprobleme-bei-suisseid-und-epa, 2010.

[15] Adhocracy api documentation. http://api.adhocracy.de, 2012.

[16] Adhocracy wiki - secret ballots. http://trac.adhocracy.de/wiki/SecretBallots, 2012.

[17] Hand counts of votes may cause errors. http://www.sciencedaily.com/releases/2012/02/120202151713.htm, 2012.

[18] Public software group e.v. - liquidfeedback. http://www.public-software-group.org/liquid_feedback, 2012.

[19] ANONYMOUS. Liquid democracy: When, not if. http://www.kuro5hin.org/story/2003/7/16/201556/896, 2003.

[20] APPEL, A. W., GINSBURG, M., HURSTI, H., KERNIGHAN, B. W., RICHARDS, C. D., TAN, G., AND VENETIS, P. The new jersey voting-machine lawsuit and the avc advantage dre voting machine. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections* (Berkeley, CA, USA, 2009), EVT/WOTE'09, USENIX Association, pp. 5–5.

[21] BALZAROTTI, D., BANKS, G., COVA, M., FELMETSGER, V., KEMMERER, R., ROBERTSON, W., VALEUR, F., AND VIGNA, G. Are your votes really counted?: testing the security of real-world electronic voting systems. In *Proceedings of the 2008 international symposium on Software testing and analysis* (New York, NY, USA, 2008), ISSTA '08, ACM, pp. 237–248.

[22] BUNDESTAG, G. Constitution of the federal republic of germany. http://www.iuscomp.org/gla/statutes/GG.htm.

[23] CHAUM, D. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM 28*, 10 (Oct. 1985), 1030–1044.

[24] ECKENFELS, M. Alle macht dem volke, 2012.

[25] FRIEDMAN, B. Diebold voting machines can be hacked by remote control. http://www.salon.com/2011/09/27/votinghack/, 2011.

[26] GERMANY, C. C. C. Schwerwiegende wahlcomputer-probleme bei der hessenwahl -– wahleinsprüche und nachwahlen erwartet. http://www.ccc.de/de/updates/2008/wahlbeobachtungen-hessen, 2008.

[27] GUTMANN, P. *Cryptographic security architecture design and verification.* SpringerVerlag, 2003.

[28] JEFFERSON, D. If i can shop and bank online, why can't i vote online?, Nov. 2011.

[29] MAO, W. *Modern Cryptography: Theory and Practice.* Prentice Hall Professional Technical Reference, 2003.

[30] MENEZES, A. J., VANSTONE, S. A., AND OORSCHOT, P. C. V. *Handbook of Applied Cryptography*, 1st ed. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[31] SCHNEIER, B. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C.* John Wiley & Sons, Inc., New York, NY, USA, 1995.

[32] Wolchok, S., Wustrow, E., Halderman, J. A., Prasad, H. K., Kankipati, A., Sakhamuri, S. K., Yagati, V., and Gonggrijp, R. Security analysis of india's electronic voting machines. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 1–14.

A

List of Abbreviations

- AP/APS - Anonymous Proxy/Anonymous Proxy Server
- BVerfG - Federal Constitutional Court (BundesVerfassungsGericht)
- BWG - Federal Electoral Law (BundesWahlGesetz)
- CPU - Central Processing Unit
- D - hash chain depth
- DH - Diffie-Hellman
- DHKE - DH Key Exchange
- DMA - Direct Memory Access
- DoS - Denial of Service
- e-voting - electronic voting
- HC - Hash Chain
- h_S - HC starting element
- h_A - HC anchor element
- h_I - HC inner element
- ID - Identity Document (Identity Card)
- IO - Input/Output
- i-voting - internet voting
- ISO - International Organization for Standardization

- LD - Liquid Democracy
- MAC - Message Authentication Code
- MitM - Man in the Middle
- N - number of HCs/leaves in a Merkle tree
- NTP - Network Time Protocol
- OSI - Open Systems Interconnection
- PoC - Proof of Concept
- PTP - Precision Time Protocol
- RAM - Random Access Memory
- RSA - Rivest Shamir Adleman
- SQL - Structured Query Language
- SSL - Secure Socket Protocol
- TLS - Transport Layer Security
- VC - Voting Computer
- VR - Voting Register
- UI - User Interface