Aalborg university

Department of Mathematical Sciences

Master of IT, Software Development

Master thesis

Electronic voting application based on public verifiable secret sharing

Author:
Sune Chung Jepsen
Kasper Jensen Lemming

Supervisor:
Ignacio Cascudo

June 2017

AALBORG UNIVERSITET

Acknowledgements

It has been a learning period for the past six months, where we have studied the public verifiable secret sharing protocol. It has been a steep learning curve, but as we have worked with the topics on a theoretical and practical level, we have gained an understanding of the protocol and its application.

We would like to express our gratitude to our supervisor Ignacio Cascudo for his guidance and expert knowledge on this area throughout our work with this Master thesis.

Summary

In this Master thesis we will study the public verifiable secret sharing protocol and how it can be used in an electronic voting application based on the work from [Sch99]. Based on this knowledge we will design and implement a web based electronic voting application.

Our work with this protocol leads to the following main topics which should cover our objective about Shamirs secret sharing, multiparty computation, public verifiable secret sharing protocol and our implementation of an electronic voting application.

1. Voting
2. Mathematical understanding
3. Multiparty computation
4. Electronic voting protocol
5. Designing the application
6. The application
7. Reflection

We start with describing the concepts of electronic voting and the challenges with the different types of electronic voting applications. We will use other studies and their demands for concrete security requirements, which we can include in our consideration for our electronic voting application.

To understand the public verifiable secret sharing protocol one need some basic mathematical understanding and some knowledge about cryptographic tools. Modular arithmetic and group theory will be key elements in understanding how the protocol works. Regarding to the cryptographic tools we will present the discrete logarithm problem which is the security primitive for this protocol.

Multiparty computation is basically about allowing parties to compute some function on some private inputs, in such a way that they learn the result but not the inputs from the other parties. Secret sharing is about hiding information in a random polynomial. By using this polynomial, parties can create shares based on evaluations in the polynomial. If enough parties then collect their shares together they will be able to recover the secret. We will present a simple

secret sharing example which illustrate how a secret can be distributed and reconstructed. In addition to these properties the public verifiable secret sharing protocol gives us the ability to publicly verify the validity of the shares among the parties involved in this protocol. This means that the protocol is secure against malicious parties which try to send votes which they are not supposed to do.

The part describing the electronic voting protocol is divided into two parts, a basic and a more-in-depth description of the protocol. The first part is intended to supply enough basic knowledge for a software developer to implement a simple voting application based on the protocol. The second part is intended to give a more thorough insight of the protocol, here we describe the mathematical justifications behind the protocol as well as the proofs to verify the correctness and the consistency of the protocol.

Designing the application is about architectural strategies for our application based on the knowledge from literature of [BCK12] and [Chr10]. We took the security requirements of electronic voting in general as described in [Cet09] as the functional demands for our application. To extract the architectural demands, such as *Interoperability*, *Modifiability* and *Testability*, we used Quality attribute scenarios which is a way of defining a clear architectural measurable demands. In order to illustrate the impact these demands have on the architecture we will use several documentation methods here among diagrams. The process of deriving these demands is done through an Quality attribute workshop [BEL$^+$03]. Since we have limited time we only used the structure of the Quality attribute workshop for deriving the Quality attributes scenarios without actually holding a workshop. Furthermore the structure of the workshop helped us prioritize among a long list of demands, and helped us derive the most important scenarios which we then proceeded on implementing.

In the application part, we will elaborate on how we have implemented the final design of the architecture on a proof-of-concept application.

Lastly the reflection part, summarizes the most important reflections on our results from the theoretical and the practical parts of this thesis.

Contents

Listings

Part I

Introduction

Chapter 1

Introduction

1.1 Introduction

In this master thesis we will study how to implement an electronic voting scheme application based on cryptography tools, most importantly the Publicly Verifiable Secret Sharing (PVSS) protocol and the multiparty computation protocol (MPC). The main paper for this protocol will be Berry Schoenmakers paper "A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting" [Sch99].

The PVSS protocol is based on Shamirs secret sharing. Secret sharing is way of distributing a secret among multiple parties, where each party gets a subset of the secret. The individuel party member cannot extract any information about the secret from his subset alone, but if enough parties pulls there shares together, they will be able to recontruct the secret. More technically the idea is to hide a secret inside of a polynomial so that given certain partial information of the polynomial we can recover the secret that was hidden in it.

The MPC protocol is a protocol which uses secret sharing and allows several parties to compute some function on some private inputs, in such a way that they learn the result but not the inputs from the other players. MPC is not using conventional methods, where some commonly trusted party, could gather sensitive information. In the left image of figure 1.1 it shows a judge who all participants trust to give their secret inputs. The trusted party can then compute the outcome of the process, for instance the sum of all inputs, and reveal the output. The right figure is how an equivalent MPC would work. Here there is no trusted party, but by using multiparty computation they can still achieve the same level of secrecy, but without having to trust someone.

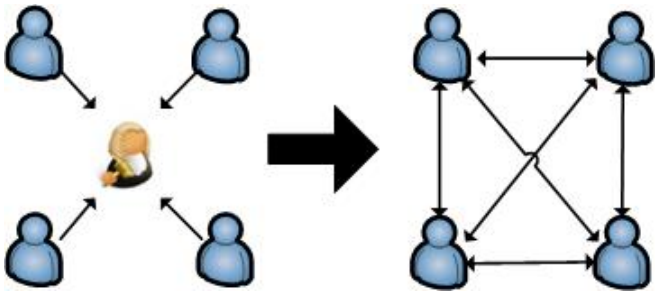


Figure 1.1: Multiparty computation overview

This leads to a core observation namely that without a trusted party the ability to validate the input the MPC protocol relies on the participants honesty. The PVSS protocol used in this thesis proposes a solution to this problem. The idea is that not only can the participants verify their own shares, but that anybody can verify the correctness of the transmitted data.

1.2 Motivation

There are theoretical papers about how participants can communicate secure. For different reasons there are fewer papers which propose how to implement their theoretical results. It could be interesting to combine the knowledge we gained from our lectures in it-security and software architecture, to describe, design and implement a MPC protocol. In collaboration with our mentor we decided to study and implement the electronic voting solution based on the PVSS protocol describe in [Sch99]. In this solution we will have focus on designing a secure and scalable distributed architecture. Using known software design principle from [BCK12] and [Chr10], we discuss and reflect on different solutions.

1.3 Objectives

This master thesis consist of two main parts. The first a theoretical background where we study the electronic voting protocol described in [Sch99]. In order to understand the protocol, some study in the basic field of cryptography is required. In the second part of the project, we will try to design and implement a web based electronic voting application based on the protocol. Basically this master thesis have the following objectives.

1. Describe the theory behind Shamirs secret sharing and multiparty computation as well as the cryptographical concepts needed to understand it. The aim is to describe the theories clearly and with examples such that one really understand how it works.

2. Describe the electronic voting protocol [Sch99] and the mathematically justification behind it, again aiming to describe the protocol clearly and with examples to really understands how it works.

3. Design and implement a secure and scalable web based electronic voting application based on the protocol. Known software design principles will be used to acquire high and reliable software quality. The aim is to make the application web based, such that it is easily accessible to a broad audience.

1.4 Limitations

The following limitations have be identified.

- The field of multiparty computation is broad and there exist numerous of scientific papers describing different protocols. This thesis will only focus on the main concepts in order to understand the electronic voting protocol.
- Even though the aim is for high and reliable software quality, the application designed and developed in this thesis is to be considered proof-of-concept.
- As this project involves three large objectives, we cant document every details due to time constrain. Regards to the third objective we will only document our theoretically approach and the final implementation of our electronic voting application. Leaving out documenting the agile development process that normally involves implementing the design iterative and reevaluating the design through out the process.

Part II

Theoretical

Chapter 2

Voting

In this chapter we will briefly describe electronic voting in general. In order to develop an electronic voting application, its important to understand the challenges and secure parameters for an electronic voting. This knowledge will help us define system requirements for the design of the electronic voting systems. And it will ultimately serve as parameters which we can hold our implementation accountable.

2.1 Introduction

In many aspects of our life's we encounter the act of voting, from simple things as voting whats for dinner, to the more complex things as a government election. The later involving a large number of people from different geographical locations. These election is normally handled by dividing the people into sections based on location, each section handling it's own sub-election and submitting the result to a overall tally. Though many countries still mostly rely on the old fashioned paper based ballots, we have in the recent years seen an increase in electronic voting. Electronic voting refers to the act of voting through an electronic devices and depending on the voting equipment and location, this can be divide into five categories [Cet09].

2.2 Classification

DRE voting Direct Recording Electronic is a specialized standalone electronic voting machine, which have the attributes of being physically hardened and have software specifically for voting installed. Votes casted on a DRE is done within a voting booth located on a polling site, and the votes is then recorded into a electronic ballot box.

Poll-site voting The voting is located on a polling site, votes are cast on public computers located on the site, in favour for voting booths. The computers on site are connected to a counting authority server though a closed and controlled network. Authentication can be done prior to the voting period or at the polling site.

Poll-site kiosk voting Votes are casted inside a voting booth located at a polling site, using a terminal. The terminal is connected to a counting authority server through a closed and controlled network. Authentication is done at the polling site before allowed access to the voting booth.

Poll-site Internet voting Votes are casted on public computers located on a polling site. The public computers on site is online and connected to a counting authority server over uncontrolled network. Authentication can be done prior to the voting period or at the polling site

Remote Internet voting Simply requires internet access and can be done from a home computer. Authentication is done prior to the voting an typical involves password or some type of authentication token.

The electronic voting application design and implemented in this thesis will be working within the **Remote internet voting** classification.

2.3 The Voting Process

Similar to the classification, there exist several of different systems and protocols for electronic voting. But they typically all follows the same process and includes the same actors as illustrated and describe below [Cet09].

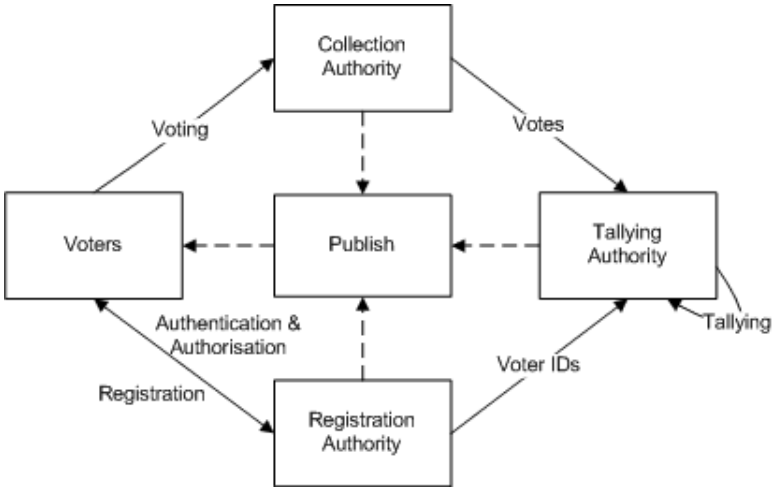


Figure 2.1: Voting process

Voter Voter refers to a person who is eligible and who has registered for the privileges of voting in a given election.

Registration authority Registration authority refers to the authority that is responsible for registering eligible voters, and ensuring that only these voters are allowed to vote. Furthermore they ensures that voters only vote ones.

Collection authority The collection authority refers to the authority that is responsible for properly collecting all the votes. This authority can be represent as a simple electronic ballot box.

Tallying authority Tallying authority are responsible for counting the votes of the election and publishing the result.

The voting process as illustrated on figure 2.1 holds true for any voting system, and includes four stages.

Registration Prior to an election, people who are eligible for voting, signs up to vote in the election. These voters are then registered, thus hereby ensuring no double voting acquires.

Authentication and Authorisation Registered voters are authenticated, if they are found eligible and have not voted yet. Ones authenticated, the voter is giving access to cast his vote in the election.

Voting The voters cast there votes.

Tallying In this final stage the votes are counted, and of course only valid votes are being included. Ones the counting process is finished the final count is published.

The terminologies introduced in this section will be used throughout the rest of the thesis.

2.4 Challenges

Though the process of electronic voting have many similarities with paper based voting systems, there is still concern of the security. In paper based voting systems, the security is easily noticeable as this is represented by officials observing every stages of the process. The process typically goes along the lines of: When arriving to the polling site a voter typically already have proofs that he is eligible and is simply registered and handed the paper ballots. The paper ballot is fulfilled in a voting booth where only the voter himself is present. When the ballot is fulfilled, it is folded or put in an envelope, in order to hide the vote, and then put in a ballot box located on the polling site. Note that ballot is anonymous and typical only requires a simple mark. Ones all the votes have been cast or a deadline is reached then the votes is counted. Throughout this entire process there are neutral officials physical present typically along with represents from each party in the election and passive observers.

In electronic voting systems the security is not this visible and if the system falls into the classification of *Poll-site Internet voting* or *Remote Internet voting* then uncontrolled network is used and the group of potential adversary is significantly increased. To ensure security and public trust an electronic vote system should aim to fulfill the goals listed below [DGS02].

Privacy Throughout the process no information should be leaked, only the final counting should be made public. At no time should a vote could be linked to the voter.

Robustness The system should be tolerant against cheating. Only valid and correctly fulfilled ballots should be counted. Nobody should be able to manipulating the final count.

Universal verifiability The final count should be public verifiable. Anyone should be able to convince himself of the fairness of the election. This should be done without gain any other information.

2.5 Security Requirements

The previous three goals are fairly board and can with benefit be specified into concrete security requirements as done in [Cet08]

- *Voter Privacy* No one should be able to link a vote back to the specific voter, and only the voter should know his vote. These requirements shall hold during and after the election.
- *Eligibility* Only Eligible and registered voters can vote.
- *Uniqueness* Only one vote per registered voter should be counted.
- *Fairness* None should be able to gain any knowledge of the outcome of the election, before the ending. This is to prevent voters of voting accordingly to any leaked information.
- *Uncoercibility* Nobody should be able to extract the value of a vote. This is to prevent anybody from compelling a voter by force, intimidation, or authority to cast a vote in a specific way.
- *Receipt-freeness* The voting system should not produce a receipt that reveals any information about the casted vote. This is to prevent a voter from trading his vote.
- *Accuracy* The final tally should be correctly computed from valid casted votes. It should not be possible to manipulate the final tally without being detected.
- *Universal Verifiability* It should be possible for any participants and observers to validate individual votes as well as the final tally of the election.
- *Individual Verifiability* Every registered voter should be able to verify that his vote is counted correctly.

2.6 Cryptography

Fulfilling all of the the above security requirements, working within the classification of *Remote Internet voting* is not a simple task. However the field of cryptography provides us with the tooling and the protocols to construct such an application. As the field of cryptography is comprehensive, we will only, in this thesis, be looking at the concepts used in the protocol to which this thesis is based upon. Building on the figure 2.1 from section 2.3 and the concepts described in [Cet09] we show in figure 2.2 below how the cryptography interacts with the voting process.

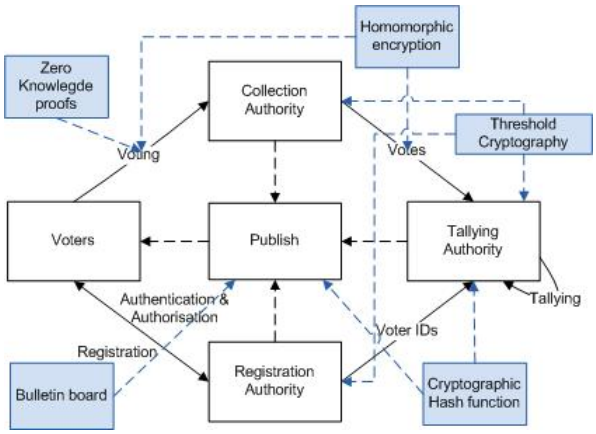


Figure 2.2: Voting process and cryptography

Marked with the blue boxes we see the cryptography concepts that will be used in this thesis. Most of the concepts will be described in depth in the following chapters, in fact the only concept we will elaborate on here is the bulletin board the rest we will briefly present.

Zero Knowlegde proofs refers to proofs, that have the notion of a prover and a verifier. The proofs promises that if the prover is honest then the verifer will always accept and if the prover is dishonest the verifier would reject with overwhelming probability. It also promises to not leak any information besides the outcome of the verdict.

Homomorphic Encryption is a concept that allows one to compute on the combination of data entities without having to retrieve the individual entity. Thereby preserving the confidentiality of the individual entity. This also allows one to compute the calculations on encrypted data without decrypting it first.

Threshold Cryptography is in this thesis represented using Shamirs secret sharing as described in the introduction.

Cryptographic Hash function is a function that takes a arbitrary size input and outputs a fixed length output. Its required that the output is easily computed given any input, and that it is hard to invert the output into the given input.

Bulletin board as described in [Cet09], is a public broadcasting channel where parties may publish there information. All communication to the bulletin board is public an can thereby be monitored. Generally it is only allowed to publish and read information on the bulletin board, no parties is allowed to delete or alter the published information. The public nature of the bulletin board prevents good support in order to fulfill the requirements of *Universal Verifiability* and *Accuracy*.

Chapter 3

Mathematical understanding

In this chapter we describe the mathematical concepts and theories behind the electronic voting protocol. As we do not except the reader to have a fully comprehensive mathematical background, we will be describing the required concepts and theories here.

3.1 Method

To ensure structure the sections will be constructed based on one or more of the following parts.

1. General structure
   (a) Informal description
   (b) Definition
   (c) Example
   (d) Mathematical justification
2. Pseudo code

General structure The general rule will be that every section will start informal description of the subject. Here we will give a informal description of why this it is relevant to our rapport. After that a more formal description will come. Their will be parts where the formality will require a more in depth explanation. To avoid disruption we then put this formality last in the section.

Pseudo code We will use pseudocode as an informal technique to outline the structure of our algorithms. This technique aims to describe a solution so that it is easy to read for humans.

3.2 Modular arithmetic

"Modular arithmetic is a system of arithmetic for integers, where numbers 'wrap around' upon reaching a certain value—the modulus" - Wikipedia

An intuitive example of modular arithmetic usages is given a 12-hour clock. Let the clock be 10:00 now, then in 5 hours the clock will show 3:00 and not 15:00. 3 is the remainder of 15 modulus 12. We can define this as.

Definition 3.2.0.1: Modulo Operation

Let $a, r, m \in \mathbb{Z}$ (where $\mathbb{Z}$ is a set of all integers) and $m > 0$ and we write

$$a = r \ mod \ m$$

if m divides $a - r$.
m is called the modulus and r is called the remainder.

Computing the remainder By example we can compute the remainder according to the definition. Given: $a, m \in \mathbb{Z}$ we compute the remainder by the following fomular: $a = qm + r$. This example illustrate that the remainder is not unique. Though the remainder will be unique in set [0, m).

$$\begin{aligned}
42 = 4 \cdot 9 + 6 &\implies r = 6, \quad \text{by definition} : (42 - 6) &&= 36, 9|36 \\
42 = 3 \cdot 9 + 15 &\implies r = 15, \text{ by definition} : (42 - 15) &&= 27, 9|27 \\
42 = 5 \cdot 9 + (-3) &\implies r = -3, \text{ by definition} : (42 - (-3)) &&= 45, 9|45
\end{aligned}$$

Equivalence classes Above can also be written with the modulo operator. Here we show that all have different remainder but are in the same equivalence class modulo *9*. This means that all members of a given equivalence class behave equivalently. Note also that one can compute with negative integers.

$$\begin{aligned}
42 &= 6 \ mod \ 9 \\
42 &= 15 \ mod \ 9 \\
42 &= -3 \ mod \ 9
\end{aligned}$$

Computing the inverse As we will see later, computing the inverse becomes an important part in this protocol for how to divide modulo an integer arithmetically. For this we have the Extended Euclidean algorithm which allows us to compute the inverse modular some integers. This computation results in computing the inverse.

Extended Euclidean algorithm To compute the inverse means that we want to compute the following $a \ \cdot \ x \ mod \ q = 1$ where $a, \ x, \ q$ are integers and x is the inverse of a. The condition for the existence of the inverse is that the $gcd(q, \ a) = 1$. This means that the only number which divides both q and a is 1. So if $gcd(q, \ a) = 1$ the Extended Euclidean Algorithm computes the inverse of a.

Explanation of the Extended Euclidean algorithm (EEA) Since we will use an implementation of the EEA in our electronic voting application we will explain the EEA by example. In order to explain the EEA we will explain the regular Euclidean algorithm (EA). The first two lines (6-7) in the algorithm is the EA, which computes the *gcd*. It turns out that if the $gcd(n, a) = 1$ and

we give the EEA $gcd(n, a)$, where n is the modulo integer and a is an integer, then the t parameter will be the inverse of a. [PPP09]

Algorithm 1: Extended Euclidean Algorithm (EEA)

Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: gcd(r_0, r_1), as well as s and t such that gcd(r_0, r_1) = $s \cdot r_0 + t \cdot r_1$.

Initialization:
  $s_0 = 1 \quad t_0 = 0$
  $s_1 = 0 \quad t_1 = 1$
  $i = 1$
begin
  do
    $i = i + 1$
    $r_i = r_{i-2}\ mod\ r_{i-1}$
    $\mathbf{q_{i-1} = (r_{i-2} - r_i)/r_{i-1}}$
    $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$
    $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$
  while $r_i \neq 0$;
  return
  gcd(r_0, r_1) = r_{i-1}
  $s = s_{i-1}$
  $t = t_{i-1}$

The EA works that given to integers $r_0 = 911$ and $r_1 = 301$. The gcd is computed by reducing the problem of finding the gcd of two given numbers to that of the gcd of two smaller numbers. The example shows that the *gcd* between 911 and 301 is 1.

911	$= 3 \cdot 301 + 8$	$gcd(911, 301)$	$= gcd(301, 8)$	
301	$= 37 \cdot 8 + 5$	$gcd(301, 8)$	$= gcd(8, 5)$	
8	$= 1 \cdot 5 + 3$	$gcd(8, 5)$	$= gcd(5, 3)$	
5	$= 1 \cdot 3 + 2$	$gcd(5, 3)$	$= gcd(3, 2)$	
3	$= 1 \cdot 2 + 1$	$gcd(3, 2)$	$= gcd(2, 1)$	
2	$= 1 \cdot 1 + 1$	$gcd(2, 1)$	$= gcd(1, 1)$	
1	$= 1 \cdot 1 + 0$	$gcd(1, 1)$	$= gcd(1, 0)$	$= 1$

Example Extended Euclidean algorithm We will now extend the EA example with EEA with the same values $r_0 = 911$ and $r_1 = 301$. Here the main point is that the last iteration we compute the parameter t, from two previous iterations. This t parameter is the inverse of r_1. On the left-hand side, we compute the standard Euclidean algorithm, i.e., we compute new remainders r_2, r_3, ... Also, we have to compute the integer quotient q_{i-1} in every iteration. On the right-hand side we compute the coefficients s_i and t_i such that $r_i = s_i r_0 + t_i r_1$.

i	$r_{i-2} = q_{i-1} \cdot r_{i-1} + r_i$	$r_i = [s_i]r_0 + [t_i]r_1$
2	$911 = 3 \cdot 301 + 8$	$r_2 = 8 = [1]911 + [-3]301$
3	$301 = 37 \cdot 8 + 5$	$r_3 = 5 = 301 - 37 \cdot 8$ $r_3 = 301 - 37(911 - 3 \cdot 301)$ $r_3 = -[37]911 + [112]301$
4	$8 = 1 \cdot 5 + 3$	$r_4 = 3 = 8 - 5$ $r_4 = 1 \cdot 911 + (-3) \cdot 301 - (-37 \cdot 973 + 112 \cdot 301)$ $r_4 = [38]911 - [115]301$
5	$5 = 1 \cdot 3 + 2$	$r_5 = 2 = 5 - 3$ $r_5 = -37 \cdot 911 + 112 \cdot 301 - (38 \cdot 911 - 115 \cdot 301)$ $r_5 = -[75]911 + [227]301$
6	$3 = 1 \cdot 2 + 1$	$r_6 = 1 = 3 - 2$ $r_6 = 38 \cdot 911 - 115 \cdot 301 - (-75 \cdot 911 + 227 \cdot 301)$ $r_4 = [113]911 - [342]301$
7	$2 = 1 \cdot 1 + 1$	$r_7 = 1 = 2 - 1$ $r_7 = -75 \cdot 911 + 227 \cdot 301 - (113 \cdot 911 - 342 \cdot 301)$ $r_7 = -[188]911 + [569]301$

From the EEA we have now computed the inverse of 301 modulo 911 as 569 from the following equation $1 = -188 \cdot 911 + 569 \cdot 301$.

To understand how the EEA works we observe that the righthand side is always constructed with the help of the previous linear combinations. We will now derive recursive formulae for computing s_i and r_i in every iteration. Assume we are in iteration with index i.The two previous iterations we computed the values.

$$r_{i-2} = [s_{i-2}]r_0 + [t_{i-2}]r_1$$
$$r_{i-1} = [s_{i-1}]r_0 + [t_{i-1}]r_1$$

In the current iteration i we first compute the quotient q_{i-1} and the new remainder r_i from r_{i-1} and r_{i-2}:

$$r_{i-2} = q_{i-1} \cdot r_{i-1} + r_i$$

This equation can be rewritten as:

$$r_i = r_{i-2} - q_{i-1} \cdot r_{i-1}$$

The goal is to represent the new remainder r_i as a linear combination of r_0 and r_1 as $r_i = [s_i]r_0 + [t_i]r_1$. The core step for achieving this is by substitute r_{i-2} and r_{i-1} by the following. The general formula is derived by substitute r_{i-2} and r_{i-1} as:

$$r_i = (s_{i-2}r_0 + t_{i-2}r_1) - q_{i-1}(s_{i-1}r_0 + t_{i-1}r_1)$$

If we rearrange the terms we obtain the desired result:

$$r_i = [s_{i-2} - q_{i-1}s_{i-1}]r_0 + [t_{i-2} - q_{i-1}t_{i-1}]r_1$$
$$r_i = [s_i]r_0 + [t_i]r_1$$

3.3 Group theory

A computer cant work with infinite set. If we look at the set of reel numbers we have infinite numbers like $\frac{1}{3} = 0.33333...3$, we can therefor not work with reel numbers. We therefor turn to integers which we know dont have these types infinite representation. However the set is also infinite. We therefor need find a subset of integers which can form a finite group. We need this subset of integers to uphold certain properties. We start by looking at the definition of a general group [PPP09].

Definition 3.3.0.1: Group

A group is a set $\mathbb{G}$ along with a binary operation $\circ$ for which the following conditions hold:.

- **(Closure:)** *For all* $g,\ h \in \mathbb{G},\ g \circ h \in \mathbb{G}$.
- **(Existence of an identity:)** *There exists an* identity $e \in \mathbb{G}$ such that for all $g \in \mathbb{G}, e \circ g = g = g \circ e$
- **(Existence of Inverses:)** For all $g \in \mathbb{G}$ there exists an element $h \in \mathbb{G}$ such that $g \circ h = e = h \circ g$ such that an h is called an inverse of g and e is an identity.
- **(Associativity:)** For all $g_1, g_2, g_3 \in \mathbb{G}, (g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$

When $\mathbb{G}$ has a finite number of elements, we say $\mathbb{G}$ is a finite group and let $|\mathbb{G}|$ denote the order of the group; that is, the number of elements in $\mathbb{G}$.
A group $\mathbb{G}$ with operation $\circ$ is abelian if the following holds:

- **(Commutativity:)** For all $g, h \in \mathbb{G}, g \circ h = h \circ g$

As we described above we should find a subset of integers which satisfies the group definition. By using modulo we define a subset of integers, as $G = \mathbb{Z}_q$ where q is an modulo integer. By example we have the following $\mathbb{Z}_4 = \{0, 1, 2, 3\}$

Closure Closure means when we do operations in the set we always end up with an element from the set. The modulo operation ensures that we always reduce computations into a closed set.

Existence of Inverses The inverse means that we want to compute a number such that $a \cdot x \ mod\ q = 1$. For the integers we have to ensure that the greatest commend divisor is 1 to ensure the inverse property. By example we see the following $\mathbb{Z}_4 = \{0, 1, 2, 3\}$. Removing all the numbers that does have a gcd with 4 larger then 1 gives set of $\mathbb{Z}_4^* = \{1, 3\}$. Note the star in the notation. This refers to a set with none greatest common divisor larger then 1. If we take a prime we know that the gcd holds for every number up to the prime. So $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$ becomes $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$. To compute the inverse

we will use Extended Euclidean algorithm which is described in section 3.2.

Existence of an identity There should always be a neutral element. When the operation is mulitplication the neutral element is 1 and when the operation is addition the neutral element is 0.

Associativity By example one can show that associativity holds. If we take $\mathbb{Z}_{10}^*$ with the following two expressions $(3 \cdot 7 \ mod \ 10) \cdot 9 \ mod \ 10$ and $(3 \cdot (7 \cdot 9 \ mod \ 10) \ mod \ 10)$. This can be reduced to $(3 \cdot 7 \ mod \ 10) \cdot 9 \ mod \ 10 = 1 \cdot 9 \ mod \ 10 = 9 \ mod \ 10$. This can be reduced to $(3 \cdot (7 \cdot 9 \ mod \ 10) \ mod \ 10) = 3 \cdot 3 \ mod \ 10 = 9 \ mod \ 10$.

Cyclic group A cyclic group is if the group contains at least one element (generator) with the order of the cardinality (number of elements) of the group. This can also be formulated as the maximum cycle length which is $p-1$, which means when the generator begins to repeat the elements it is said to be cyclic.

Generators An example of a generator could be the following. Let $q = 5$ and $\mathbb{Z}_q^*$ be a group with $\mathbb{Z}_q^* = \{1, 2, 3, ..., q-1\}$. It can be seen that $g = 2$ is a generator because, $2^1 = 2,\ 2^2 = 4,\ 2^3 = 8 \ (mod \ 5) = 3,\ 2^4 = 16 \ (mod \ 5) = 1$, generates every element in the group. Here we see that 2 have the maximum order $ord(2) = 4$ of the group which i 4 and therefor this group is said to be cyclic.

Definition 3.3.0.2: Cyclic Group

A group G which contains an element α with maximum order $ord(\alpha) = |G|$ is said to be cyclic. Elements with maximum order are called generator.

Field The protocol uses finite field because the protocol uses Shamir secret sharing. Particular we are using field in the exponents where we are summing and multiplying. In the base we multiplying. But before we can explain a finite field we should know a field. A field is a algebraic structure which forms an additive group and multiplicative group respectively with group operation addition and multiplication. Remember that a field also contains subtraction and division operator. Subtraction can be formulated through addition of a negative number. Division can be formulated as a multiplication between number an inverse.

Definition 3.3.0.3: Field

A field F is a set of elements with the following properties:

- All elements of F form an additive group with the group operation + and the neutral element 0.
- All elements of F except 0 form a multiplicative group with the group operation $\cdot$ and the neutral element 1.
- When the two group operations are mixed, the distributivity law holds, i.e., for all $a, b, c \in F : a(b + c) = (ab) + (ac)$.

Finite field A finite field is when one does operation in the set the result

stays in the set. When we do operation e.g. do multiplication and uses modulo we will stay in the set.

Definition 3.3.0.4: Finite fields

A finite field is a field F which contains a finite number of elements. The order of F is the number of elements in F.

3.4 Cryptographic tools

The following section will be about the cryptographic assumptions which are used in the PVSS protocol. We will also describe some techniques which will be used in practical part of coding the protocol.

3.4.1 Zero knowledge proof

In cryptography, a zero-knowledge proof is a protocol by which one party (the prover) can prove to another party (the verifier) that a given statement is true, without revealing any information apart from the fact that the statement is indeed true.

Zero knowledge proof is a important part of the PVSS protocol. It is used for verifying the correctness of the data. Intuitively Zero knowledge proof is a protocol between two parties a prover and a verifier. The prover tries to convince the verifier about some statement, for which he uses additional knowledge. In the last step in the protocol, the verifier either accepts or rejects the proof. A zero-knowledge proof must satisfy following three properties:

- **(Completeness:)** If the prover is honest and the statement is true, then the honest verifier always accept.
- **(Soundness:)** If the statement is false then it should fail with overwhelming probability.
- **(Zero-knowledge:)** If the statement is true, no cheating verifier learns anything other than the fact that the statement is true.

3.4.2 Discrete Logarithm problem

One-way function One-way functions are easy to compute but it is very difficult to compute their inverse functions. Thus, having data x it is easy to calculate $f(x)$ but, knowing only the result of $f(x)$ it is hard to calculate the value of x. We say a function $f(x)$ is easy to compute if this can be done in polynomial running time. In order to be useful in practical crypto schemes, the computation of $f(x)$ should be fast enough that it does not lead to unacceptably slow execution times in an application. The inverse computation of $f(x)$ should be so computationally intensive that it is not feasible to evaluate it in any reasonable time period. We define a One-way function as

Definition 3.4.2.1: One-way function

A function $f()$ is a one-way function if:
1. $y = f(x)$ is computationally easy
2. $x = f^{-1}(y)$ is computationally infeasible

An example of a one-way function is $n = pq$ where p and q are primes, it is easy to compute n given p and q but hard to find p and q given only n. Inverting this function requires finding the factors of n. Another example of a one-way function is Discrete logarithm.

Discrete logarithm (DL) A discrete logarithm is a integer a exponent that solves $g^a = c$, where g is a generator and c is a element of a cyclic group. Given a its easy to compute, but given only g and c its very hard to find a. We define discrete logarithm as

Definition 3.4.2.2: Discrete logarithm (DL) problem

Given a group G, generator g and $c \in G$, find integer a, such that $g^a = c$

An example of the DL problem could be the following. Given a group $G = \mathbb{Z}_{47}^*$, an generator $g = 5$, and an element $c = 41$ find a integer a to solve:

$$5^a \stackrel{?}{=} 41 \ mod \ 47$$

To solve this DL problem we need to find
$a = 15 \ as \ 5^{15} = 41 \ mod \ 47$

To solve this DL problem we could just tried all possible solution of $\{g^0, g^1, g^2, ..., g^{46}\} \ mod \ 47$ until we find the correct answer. However it is easy to see that given a large enough group this would be ineffective, actually the DL problem is believed to be notoriously hard, for instance in $\mathbb{Z}_p^*$ for large prime p.

We use the DL problem in the following

Diffie-Hellman problem (DHP) One of the best known application of the DL problem is in the Diffie-Hellman problem an in particular in the Diffie-Hellman key exchange. Though the Diffie-Hellman key exchange is not used in the PVSS protocol we will use it to illustrate the discrete logarithm problem.

Diffie-Hellman Key Exchange

Public: a group G and a generator g

	Alice	$\text{Eve}_{Adversery}$	Bob
Step 1	$Choses: \ a \in_R G$ $Compute: \ A = g^a$	$\xrightarrow{A}$	
Step 2		$\xleftarrow{B}$	$Choses: \ b \in_R G$ $Compute: \ B = g^b$
Step 3	$C = (B)^a$ $= (g^b)^a$ $= g^{ab}$	$\xleftrightarrow{encrypt_C(message)}$	$C = (A)^b$ $= (g^a)^b$ $= g^{ab}$

Figure 3.1: Diffie-Hellman Key Exchange

As shown in step 1 Alice is independently from Bob choosing an random element from the group G and using this to create a DL problem A which is sent to Bob. In step 2 Bob is doing the same procedure as Alice and sends a DL problem B to Bob. In step 3 they both independently from each other using the received DL problems to create a key C. Both Alice and Bob will compute the same value C which can be used as a key in a cryptographic scheme.

If we look at the exchange from Eve's point of view, we can see that Eve knows the public elements G and g aswell as A and B from step 1 and 2. If Eve can compute $C = g^{ab}$ then Eve would be able to decrypt any message sent between Alice and Bob. We define this problem as.

Definition 3.4.2.3: Computational Diffie-Hellman (CDH) problem

Given a group G, generator g and $A = g^a$, $B = g^b$, where a, b is are randomly independently chosen from $\mathbb{Z}_q$, compute $C = g^{ab}$

If Eve knows an efficient algorithm to solve the DL problem, then Eve would also be able to solve the CDH problem. Finding a from $A = g^a$ or b from $B = g^b$ then Eve can easily compute C the same way that Alice and Bob was able to. which leads us to

Lemma 3.4.2.1. *The CDH problem is no harder then the DL problem*

It is not known if the opposite direction is true in general, but in some groups, the problems are equivalent.

Decisional Diffie-Hellman (DDH) problem The CDH problem have another property namely if given a group element and the claim that this solves a CDH instance, then is not easy to verify that the solution is correct unless we can solve the CDH problem. We would need to decide if, given g^a, g^b, g^c, if it holds that $c = ab \ mod \ q$. We define this as.

Definition 3.4.2.4: Decisional Diffie-Hellman (DDH) problem

Given a group G, a generator g and $A = g^a$, $B = g^b$ and $C = g^c$, where a and b are randomly and independently chosen from $\mathbb{Z}_q$ and where c is chosen either as $c = ab$ or uniformly random from $\mathbb{Z}_q$. Now guess if c is the product of ab or randomly chosen.

Looking at the key-exchange protocol again, then if Eve calculate a value C and present this to Alice with the claim that this is a valid C, then Alice could easily tell if this is true or false as Alice is able to compute $C = g^{ab}$ and could then just simply compare the two values. However if reversed as in Alice gives Eve a value C and the claim that this is valid, then Eve cannot verify this unless Eve is able to solve the CDH problem, this lead us to.

Lemma 3.4.2.2. *The DDH problem is no harder then CDH problem*

In the PVSS protocol The security of the PVSS protocol can be reduced down to the hardness of the DL problem. Which means that if there can be found a efficient algorithm to solve the DL problem with large expoents then the entire PVSS protocol is insecure. In the next subsection we will look at known algorithms for solving the DL problem.

3.4.3 Solving the Discrete Logarithm Problem

We will here look at some of the known algorithms to solve the DL problem. It is known that the DL problem can be solved given enough time and compute power, so we only ask that this cannot be done within polynomial time. In order to increase the computation time needed to solve the DL problem, we increase the size of the groups we operate within. However the consequence of increasing the group size is an increased execution time of our protocol.

Below we have listed some of the known algorithms. These algorithms all takes the same input and gives the same output. The input is a cyclic group G, an generator g and a element $c \in G$, and we can calculate the $t = ord(G)$. The output is an integer a that solves $g^a = c$

1. **Brute-force / Exhausted search**
   This algorithms tries every solution until it finds the correct one, by simply calculating $g^0, g^1, g^2, g^3 .... g^{t-1}$ and then testing if $g^i \stackrel{?}{=} c$.

   This algorithm have a running time of $\mathcal{O}(n)$, as it have to calculate every single potential solution.

2. **Square-root attacks**
   The main idea with square-root attacks is to divide the DL problem into small problems, which can be calculate more efficiently and faster.

   As the name square-root attacks implies, the running time is around $\mathcal{O}(\sqrt{n})$

   There are several known square-root attacks, one of them is the **[Baby-steps Giant-steps algorithm]** which we will look at in details below.

3. **Index-Calculus attacks**
   Index-Calculus attacks are the best known attack against the DL problem, but it only works in certain groups, here amongst $\mathbb{Z}_q^*$. The topic of this algorithm is beyond the scope of this thesis. As such we will not elaborate on this any further other then stating, that according to [PPP09] one would need a set q of bit length at least 1024 to achieve an 80 bit security

Baby-steps Giant-steps algorithm We will here look in details how the Baby-steps Giant-steps algorithm works. For simplicity and as our implantation of the PVSS protocol operates in $\mathbb{Z}_q^*$ we will only look at the algorithm in this group.

The idea of the algorithm is to divide the group into smaller sub-groups. Given a cyclic group G, a generator g and an element $c \in g$, we can then think of all the elements in G as points in a circle

$$1 = g^0, g^1, g^2, ..., g^{n-2}, g^{n-1}, g^n = 1$$

We then divide the circle into intervals of $u = \lceil \sqrt{q} \rceil$ sizes, each interval is the Giant step whereas the elements in the intervals is the baby steps which there are at most u of. We can now look at the exponent a as the product of $a = ui + j$ where $0 \leq i, j \leq u$. This allows us to restate the problem as such

$$\begin{aligned} c &= g^a \\ c &= g^{ui+j} \\ c &= g^{ui} \cdot g^j \\ c(g^{-u})^i &= g^j \end{aligned}$$

The goal now, is to find an integer j and i such that $c(g^{-u})^i = g^j$. this can be done by computing g^j for $j = 0, 1, ..., u-1$ and $c(g^{-u})^i = g^j$ for $i = 0, 1, ..., u-1$ and then finding a match between the two lists. The details are given in algorithm 2

Algorithm 2: Baby-steps Giant-steps algorithm

Input: Group G of order p, a generator g, an element $c \in G$
Output: An integer a such that $g^a = c$

```
1  begin
2      u = ⌈√q⌉
3      for j = 0 to u − 1 do
4          Compute g^j mod q and store the pair (j,g^j) with g^j as key, in a
           table t
5      Compute g^{−u} mod q
6      for i = 0 to u − 1 do
7          Compute x = c(g^{−u})^i
8          if x is in table t then
9              return a = ui + j
```

By example we valid the algorithm. Let $p = 31$, $g = 3$ and $c = 6$.

1. $u = \lceil \sqrt{q} \rceil = 6$
2. Computing $1, g, g^1, ..., g^5$ gives us

$0 \leq j \leq u-1$	g^0	g^1	g^2	g^3	g^4	g^5
$g^j \ mod \ q$	1	3	9	27	17	26

3. Next we compute $g^{-u} = 3^{-6}$, we can use the euclidean algorithm to find $g^{-1} = 3^{-1} = 21$. Using this result we get $3^{-6} = 21^6 = 2 \ mod \ 31$.
4. Using the result for the previous step we can efficiently compute $c(g^{-u})^i$ for an increasing i until we find a match with the result from step 2.

$0 \leq i \leq u-1$	$6 \cdot 2^0$	$6 \cdot 2^1$	$6 \cdot 2^2$	$6 \cdot 2^3$	$6 \cdot 2^4$
$c(g^{-u})^2 = 6 \cdot 2^i \ mod \ 31$	6	12	24	17	3

   We found a match as $c(g^{-u})^4 = g^1 \ mod \ q$.
5. We can then compute $a = g^{iu+j} = g^{4u+1} = g^{4 \cdot 6+1} = g^{25} = 3^{25} = \underline{6} \ mod \ 31$

3.4.4 Hash function

Hash functions is a function that takes a message of arbitrary size and outputs a digest hash value of a fixed size. The hash functions used in this project are collision resistant hash function, which means that they are considered as one-way function. we define collision resistant hash function, denote just as hash functions onward, as.

Definition 3.4.4.1: Hash function

1. Takes a message of arbitrary size and outputs a value of fixed size
2. Is deterministic so the same message always results in the same hash
3. Is quick to compute the hash value for any given message
4. Is collision resistant, mean it is infeasible to find two inputs x and x' such that $H(x) = H(x')$, and $x \neq x'$
5. A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value

A function hash function: $\{0,1\}^{\leq L} \rightarrow \{0,1\}^{\ell}$ is called collision resistant if it is hard to find $x \in \{0,1\}^{\leq L}$ and $x' \in \{0,1\}^{\leq L}$ such that $x \neq x'$ and $H(x) = H(x')$ - the value (x, x') is called a collision. Here $\{0,1\}^{\leq L}$ denotes the set of bitstrings of length at most L. If $L \geq \ell$, then of course collisions exist, so they can be found given enough time, which is fine as we only ask that they are computationally hard to find.

3.4.5 Fiat Shamir

The Fiat–Shamir heuristic is a technique in cryptography for taking an interactive proof of knowledge into a non interactive proof. This way, some fact (for

example, knowledge of a certain number secret to the public) can be proven without revealing underlying information. This means that transforming a interactive proof into a non-interactive proof. Instead of the verifier creates a challenge the prover creates a challenge, on some previous data, based on a hash function.

Chapter 4

Multiparty Computation

4.1 Multiparty Computation

Multiparty computation can be defined as the problem of n-parties computing the same function on their input in a secure way. This problem was first introduced by Yao in 1982 and exemplified through what is known as the "millionaire problem" [Yao82]:

"Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?"

There are two types of MPC protocols. First there are MPC protocols which are secure against passive corruption which assumes that everyone is honest and sends correct data. Then there are MPC protocols which are secure against active corruption where adversary is able to send incorrect data. Here we need more advanced protocols like the Verifiable secret sharing protocol (VSS) and Zero knowledge proofs. Here the protocol allows the involved participants to verify their shares as consistent. This means that the VSS will be able to detect incorrect shares.

An extension to the VSS protocol is PVSS protocol where the goal is not just that the participants can verify their own shares, but that anybody can verify the correctness of the transmitted data.

4.1.1 Adversaries

The reason for this is that we cannot rely on every participants in a MPC protocol to behave as intended. Some participants could be corrupted, we divide these adversaries into two main groups [KL14].

- **Passive corruption** is that a participant (semihonest) gets access to information which the participant is not entitled to, e.g. if a voter ask another voter about his information and tries to compare the information and get some more information in that way. By using secret sharing we can prevent passive corruption, because the scheme guaranties that if t-participants are corrupted then they will not be able to gain anything.

- **Active corruption** happens when the participant (malicious) try to send values that they are not supposed to send - so the voters deviate from the protocol. The PVSS prevent these attacks.

4.2 Secret Sharing

In this section we look at theory behind secret sharing. Secret sharing is a method for splitting a secret into shares and distributing each shares among a group of participants. None of the participants, on there own, knows any information about the secret from there given share, but by pooling a sufficient number of shares together the secret can be reconstructed.

The basic model for secret sharing, as described above can be divide into two parts.

Distribution The dealer divides the secret into shares and distributing these shares among a group of participants

Reconstruction The secret is reconstructed, given enough participants is pooling there induvidiel shares together.

There are several secret sharing schemes, but we will only be looking at the one used in the electric voting protocol, which is Shamir secret sharing.

4.2.1 Shamir Secret Sharing

The PVSS protocol uses secret sharing as tool to distributes pieces of information. Basically it is about hiding information in a polynomial p. For example, a participant chooses a random polynomial of the degree 1, which is a line. The secret is the evaluation of p in 0. Each participant receive a share, the evaluation of p in some other point. Like as participant 1 receives $p(1)$, participant 2 receives $p(2)$ and etc. To construct a line we need at most two point. One can see that if we don't have at least 2 points then the line can be constructed in many ways, which is the same as saying we don't know the evaluation of $p(0)$.

In the general case, the polynomial is chosen to be of degree *t-1*. The scheme requires we need t points to reconstruct the secret. This means that if we have *(t-1)*-participants they would not be able to obtain anything about the secret.

4.2.1.1 Lagrange interpolation

In secret sharing the secret is reconstructed using Lagrange interpolation. The idea is that we know some evaluations points. With Lagrange interpolation, we have a formula, with which we can reconstruct the polynomial. The general formula, Lagrange interpolation, looks like:

Definition 4.2.1.1: Lagrange polynomial interpolation

$p(x) = \sum_{i \in C} p(i)\lambda_i(x)$, where $\lambda_i(x)$ is defined by $\lambda_i(x) = \prod_{j \in C, j \neq i} \frac{x-j}{i-j}$

In secret sharing scheme each participant get some shares ("points") and if there are enough participants then the participants can reconstruct the secret by reconstructing the polynomial. By example we will show how the formula works. We have a polynomial p, where we know the evaluation of some points.

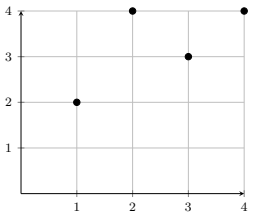
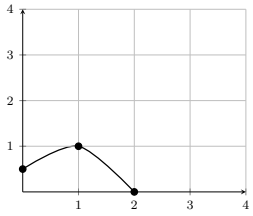
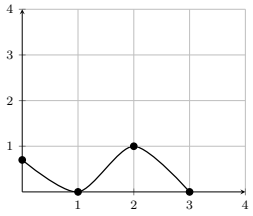


Figure 4.1: polynomial p

Instead of solving the polynomial, we will start to divide the problem into smaller pieces and solve them one by one. We create a polynomial, $\lambda_1, \lambda_2, \lambda_3$ and λ_4, one for each point from polynomial p. These polynomial takes value 1 in one point and 0 in the other points. For example λ_1 takes 1 in 1 and 0 in 2,3 and 4.

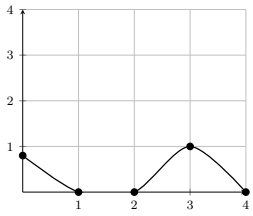


approximately λ_1

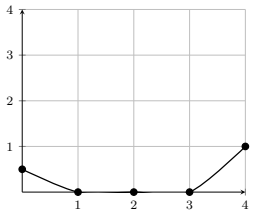


approximately λ_2

Note that the evaluation in 0 will depend on the secret. For the sake of the drawings it is just an approximately of the point.

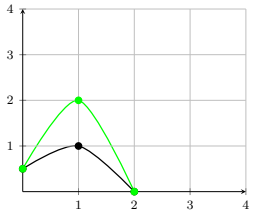


approximately λ_3

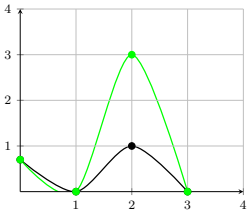


approximately λ_4

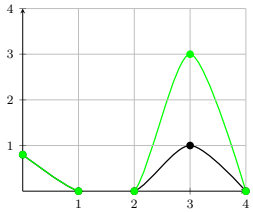
To construct the polynomial p is to take each polynomials and multiply the corresponding coefficient from p and then sum the polynomials together $2 \cdot \lambda_1 + 4 \cdot \lambda_2 + 3 \cdot \lambda_3 + 4 \cdot \lambda_4$. We started with the polynomial p and we ended with four polynomials, which takes value 2,4,3,4 and 0 in the other points. From the sum we see that the value in the first point 2+0+0+0. It is clear this will work in the other points.



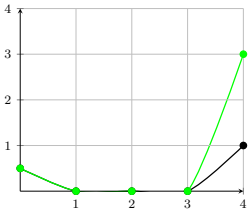
approximately λ_1



approximately λ_2



approximately λ_3



approximately λ_4

We will now construct one of the λ-polynomials by example, by showing it from λ_1 polynomial. We have the following points $\lambda_1(1) = 1, \lambda_1(2) = 0, \lambda_1(3) = 0$ and $\lambda_1(4) = 0$. We take the polynomial $(x-2)(x-3)(x-4)$, and we see that if we get correct evaluation in $\lambda_1(2) = 0, \lambda_1(3) = 0$ or $\lambda_1(4) = 0$. To get correct evaluation in $\lambda_1(1) = 1$ we divide by -6 because we see that $(1-2)(1-3)(1-4) = -6$ and then we end up with a polynomial $\frac{(x-2)(x-3)(x-4)}{(-6)}$. The polynomial still satisfies the conditions because when we divide zero with "something" we get zero. The formula for constructing λ_1

$$\lambda_1(x) = \prod_{j \in C, j \neq 1} \frac{x-j}{1-j} = \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} \cdot \frac{x-4}{1-4} = \frac{(x-2)(x-3)(x-4)}{-6}$$

λ_1 gives 1 in point 1 and 0 in the other points. What this mean is that in $\lambda_1(1) = 1$ and all other points $j(2, 3, 4)$ we have $\lambda_1(j) = 0$. We can construct $\lambda_1, \lambda_2, \lambda_3$ and λ_4 in the same way.

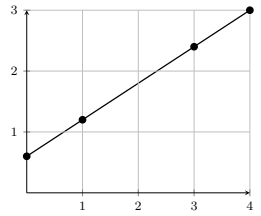
$$\lambda_2(x) = \prod_{j \in C, j \neq 2} \frac{x-j}{2-j} = \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} \cdot \frac{x-4}{2-4} = \frac{(x-1)(x-3)(x-4)}{-2}$$
$$\lambda_3(x) = \prod_{j \in C, j \neq 3} \frac{x-j}{3-j} = \frac{x-1}{3-1} \cdot \frac{x-3}{3-2} \cdot \frac{x-4}{3-4} = \frac{(x-1)(x-2)(x-4)}{-2}$$
$$\lambda_4(x) = \prod_{j \in C, j \neq 4} \frac{x-j}{4-j} = \frac{x-1}{4-1} \cdot \frac{x-3}{4-2} \cdot \frac{x-3}{4-3} = \frac{(x-1)(x-2)(x-3)}{-6}$$

With the knowledge of the evaluation of $p(1), p(2), p(3)$ and $p(4)$ we construct the formula for polynomial evaluation in $p(0)$. We use the Lagrange polynomial interpolation formula $p(x) = \sum_{i \in C} p(i)\lambda_i(x)$ and we get the evaluation on some polynomial in 0.

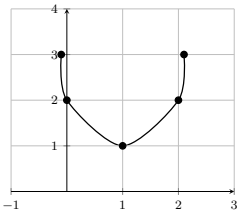
$$p(0) = p(1) \cdot \lambda_1 + p(2) \cdot \lambda_2 + p(3) \cdot \lambda_3 + p(4) \cdot \lambda_4 = 2 \cdot \lambda_1 + 4 \cdot \lambda_2 + 3 \cdot \lambda_3 + 4 \cdot \lambda_4$$

The idea of constructing the smaller polynomials is that we can reuse them for constructing other polynomials. From the smaller pieces and the evaluation

points we can construct the polynomial from Lagrange interpolation. From the secret sharing scheme we know the degree of the polynomial is bounded. Because the voter will share its secret by choosing a random polynomial of the degree at most *t-1*.Then we need t point to reconstruct the $p(x)$. If the degree 1 (line) then we need two points. The parable is where the degree is 2, here we need 3 points to construct the polynomial etc.



polynomial of degree 1



polynomial of degree 2

In the PVSS protocol we will use a simplified version of Lagrange interpolation formula. So instead of recovering the polynomium we just recover the evaluation in zero. The following will happen if take the formula $\lambda_i(x) = \prod_{j \in C, j \neq i} \frac{x-j}{i-j}$ and evaluate in zero $\lambda_i(0) = \prod_{j \in C, j \neq i} \frac{0-j}{i-j} = \prod_{j \in C, j \neq i} \frac{j}{j-i}$. We reduce the formula by evaluating in 0 and then one can multiply the numerator and denominator by -1 and then we get a the above formula. Below we computed the coefficients for 3 static participants. These coefficients will be used in our simplified example of the protocol in appendex B.1.

$$\lambda_1(0) = \prod_{j \in C, j \neq 1} \frac{2}{2-1} \cdot \frac{3}{3-1} = \frac{3}{1} = 3$$

$$\lambda_2(0) = \prod_{j \in C, j \neq 2} \frac{1}{1-2} \cdot \frac{3}{3-2} = \frac{1}{-1} \cdot \frac{3}{1} = -1 \cdot 3 = -3$$

$$\lambda_3(0) = \prod_{j \in C, j \neq 3} \frac{1}{1-3} \cdot \frac{2}{2-3} = \frac{1}{-2} \cdot \frac{2}{-1} = 1$$

4.2.2 Example computation using Shamir Secret Sharing

We have now explained the basic for secret sharing. This section will show a full concrete computational example on distribution and reconstruction of a secret between 5 participant where we want to tolerate $t = 2$ corrupted parties. The computation is computed in $\mathbb{Z}_{11}^*$. The example is that one participant, p_1, has a secret, $s = 7$ and creates shares to the other participants (p_2, p_3, p_4, p_5). Then if 3 participants combines the shares they will be able to reconstruct the secret.

First the participant p_1 creates a random polynomium, $p(x)$ at degree $t = 2$ which is the following polynomium $p(x) = s + a_1 x + a_2 x^2$, where $s = 7$ is the secret and $a_1 = 4$ and $a_2 = 1$ is coefficient uniformly randomly choosen from $\mathbb{Z}_{11}$. The following will show the distribution where p_1 create shares to the other parties. In the reconstruction we show how 3 parties will be able to reconstruct the polynomial and the secret by their shares using Lagrange interpolation.

4.2.2.1 Distribution

The shares s_1, s_2, s_3, s_4, s_5 is computed from $p(x) = 7 + 4x + x^2$ as

$$\begin{aligned}
s = p(0) &= 7 + 0 + 0 \quad (mod\ 11) = 7 \\
s_1 = p(1) &= 7 + 4 + 1 \quad (mod\ 11) = 1 \\
s_2 = p(2) &= 7 + 8 + 4 \quad (mod\ 11) = 8 \\
s_3 = p(3) &= 7 + 12 + 9 \quad (mod\ 11) = 6 \\
s_4 = p(4) &= 7 + 16 + 16 \ (mod\ 11) = 6 \\
s_5 = p(5) &= 7 + 20 + 25 \ (mod\ 11) = 8
\end{aligned}$$

Each party now recieve their shares secure. So $p_2 = s_2, p_3 = s_3, p_4 = s_4, p_5 = s_5$. No that the secret is $p(0) = s = 7$.

4.2.2.2 Reconstruction

A subset, $3 > 2$, of participant p_3, p_4, p_5 wants to reconstruct the secret by their shares using Lagrange interpolation. First every party computes a polynomial. After that the parties will be able to combine their polynomial and their shares to reconstruct $p(x)$.

p_3 computes:

$$\begin{aligned}
\lambda_3(x) &= \prod_{j \in C, j \neq 3} \frac{x-j}{3-j} = \frac{x-4}{3-4} \cdot \frac{x-5}{3-5} = \frac{(x-4)(x-5)}{(3-4)(3-5)} \\
&= (x^2 - 9x + 20)((3-4)(3-5))^{-1} \ (mod\ 11)
\end{aligned}$$

The inverse of $((3-4)(3-5)) = 2$ is 6 since $2 \cdot 6\ mod\ 11 = 1$. We now have the following polynomial

$$\begin{aligned}
\lambda_3(x) = (x^2 - 9x + 20)6 \ (mod\ 11) &= (x^2 + 2x + 9)6 \ \ (mod\ 11) \\
&= 6x^2 + 12x + 54 \ (mod\ 11) \\
&= 6x^2 + x + 10 \quad (mod\ 11)
\end{aligned}$$

We verify the following

$$\begin{aligned}
\lambda_3(3) &= 6 \cdot 3^2 + 3 + 10 \ (mod\ 11) = 67 \quad (mod\ 11) = 1 \\
\lambda_3(4) &= 6 \cdot 4^2 + 4 + 10 \ (mod\ 11) = 110 \ (mod\ 11) = 0 \\
\lambda_3(5) &= 6 \cdot 5^2 + 5 + 10 \ (mod\ 11) = 165 \ (mod\ 11) = 0
\end{aligned}$$

p_4 computes:

$$\begin{aligned}
\lambda_4(x) &= \prod_{j \in C, j \neq 4} \frac{x-j}{4-j} = \frac{x-3}{4-3} \cdot \frac{x-5}{4-5} = \frac{(x-3)(x-5)}{(4-3)(4-5)} \\
&= (x^2 - 8x + 15)((4-3)(4-5))^{-1} \ (mod\ 11)
\end{aligned}$$

The inverse of $((4-3)(4-5)) = -1$ is 10 since $-1 \cdot 10\ (mod\ 11) = 1$. We now have the following polynomial

$$\begin{aligned}\lambda_4(x) = (x^2 - 8x + 15)10 \ (mod\ 11) &= (x^2 + 3x + 4)10 \ (mod\ 11) \\ &= 10x^2 + 8x + 7 \quad (mod\ 11)\end{aligned}$$

We verify the following

$$\begin{aligned}\lambda_4(3) &= 10 \cdot 3^2 + 24 + 7 \ (mod\ 11) = 121 \ (mod\ 11) = 0 \\ \lambda_4(4) &= 10 \cdot 4^2 + 32 + 7 \ (mod\ 11) = 199 \ (mod\ 11) = 1 \\ \lambda_4(5) &= 10 \cdot 5^2 + 40 + 7 \ (mod\ 11) = 297 \ (mod\ 11) = 0\end{aligned}$$

p_5 computes:

$$\begin{aligned}\lambda_5(x) = \prod_{j \in C, j \neq 5} \frac{x-j}{5-j} &= \frac{x-3}{5-3} \cdot \frac{x-4}{5-4} = \frac{(x-3)(x-4)}{(5-3)(5-4)} \\ &= (x^2 - 7x + 12)((5-3)(5-4))^{-1} \ (mod\ 11)\end{aligned}$$

The inverse of $((5-3)(5-4)) = 2$ is 6 since $2 \cdot 6 \ mod\ 11 = 1$. We now have the following polynomial

$$\begin{aligned}\lambda_5(x) = (x^2 - 7x + 12)6 \ (mod\ 11) &= (x^2 + 4x + 1)6 \ (mod\ 11) \\ &= 6x^2 + 24x + 6 \ \ (mod\ 11) \\ &= 6x^2 + 2x + 6 \quad (mod\ 11)\end{aligned}$$

We verify the following

$$\begin{aligned}\lambda_5(3) &= 6 \cdot 3^2 + 6 + 6 \ \ (mod\ 11) = 66 \ \ (mod\ 11) = 0 \\ \lambda_5(4) &= 6 \cdot 4^2 + 8 + 6 \ \ (mod\ 11) = 110 \ (mod\ 11) = 0 \\ \lambda_5(5) &= 6 \cdot 5^2 + 10 + 6 \ (mod\ 11) = 166 \ (mod\ 11) = 1\end{aligned}$$

To construct the polynomial p we take each polynomials and multiply by the corresponding shares. More formally we apply $p(x) = \sum_{i \in C} p(i)\lambda_i(x)$ to construct the $p(x)$.

$$\begin{aligned}p(x) &= s_3\lambda_3(x) + s_4\lambda_4(x) + s_5\lambda_5(x) \\ &= s_3(6x^2 + x + 10) + s_4(10x^2 + 8x + 7) + s_5(6x^2 + 2x + 6) \\ &= (6s_3 + 10s_4 + 6s_5)x^2 + (s_3 + 8s_4 + 2s_5)x + (10s_3 + 7s_4 + 6s_5)\end{aligned}$$

Since the polynomial is of the form $p(x) = s + a_1x + a_2x^2$ we have that

$$\begin{aligned}s &= 10s_3 + 7s_4 + 6s_5 \ (mod\ 11) \\ a_1 &= s_3 + 8s_4 + 2s_5 \quad (mod\ 11) \\ a_2 &= 6s_3 + 10s_4 + 6s_5 \ (mod\ 11)\end{aligned}$$

We can now replace the variables with shares $s_3 = 6, s_4 = 6, s_5 = 8$

$$\begin{aligned}s &= 10 \cdot 6 + 7 \cdot 6 + 6 \cdot 8 \ (mod\ 11) = 150 \ (mod\ 11) = 7 \\ a_1 &= 6 + 8 \cdot 6 + 2 \cdot 8 \quad\ \ (mod\ 11) = 70 \ \ (mod\ 11) = 4 \\ a_2 &= 6 \cdot 6 + 10 \cdot 6 + 6 \cdot 8 \ (mod\ 11) = 144 \ (mod\ 11) = 1\end{aligned}$$

The reconstruction gives us the final polynomial $p(x) = 7 + 4x + x^2$ and thereby the secret value 7.

4.2.3 Verifiable Secret Sharing (VSS)

Where the basic model of secret sharing assumes that every participants involved is honest, the VSS requires its participants to prove so. The objective of the VSS is to resist malicious participants such as malicious participants can mislead as follows [Sch99]

- Dealer is sending incorrect shares to some or all participants in the distribution phase
- Participants is submitting incorrect shares during the reconstruction phase

By requiring proof of correctness of the shares, from the participants, in the distribution and the reconstruction phase, the VSS model solves the problem of malicious participants. These proofs is constructed in such a way that only the participants is able to construct and verify the proofs. Where it is a logical requirement that only the participants is able to construct the proofs, it is a different story with the verification. In fact in most case it would be ideal if anybody could validate the proofs.

4.2.4 Public Verifiable Secret Sharing (PVSS)

In a PVSS schemes it is required that, not only the participants but anybody, is able to validate the shares. It is therefore not assumed that there are private channels between the participants. All communication in PVSS schemes is done over authenticated public channels using public key encryption, which also means that the secret is only computationally hidden. A common structure for the PVSS protocols is as follows.

Initialization Each participants registers itself and must have a public key.

Distribution consists of a distribution and a verification phase

1. *Distribution of the shares*
   (a) *Dealer creates shares*
   (b) *Dealer publishes encrypted shares*
   (c) *Dealer publishes a* $proof_D$
2. *Verification of the shares*
   (a) *Anybody who knows the public key can verifier the shares*
   (b) *If the verification on* $proof_D$ *fails the dealer fails and the protocol is aborted*

Reconstruction consists of a decryption phase and pooling the shares phase

1. *Decryption of the shares*
   (a) *The participants decrypt their shares*
   (b) *The participants publishes a* $proof_{p_i}$
2. *Pooling the shares*
   (a) $proof_{p_i}$ *are used to exclude dishonest participants*
   (b) *Reconstruction of the secret by any qualified set of participants*

4.2.5 Homomorphic Secret Sharing

A homomorphism is a transformation from one algebraic structure into another of the same type so that the structure is preserved. Importantly, this means that for every kind of manipulation of the original data, there is a corresponding manipulation of the transformed data.

A homomorphic encryption scheme is a crypto system that allows computations to be performed on data without decrypting it. It is an encrypting scheme which allows computations to be carried out on ciphertext, thus generating an encrypted result which, when decrypted, matches the result of operations performed on the plaintext.

Homomorphic Secret Sharing is a type of secret sharing algorithm in which the secret is encrypted via homomorphic encryption. In the PVSS scheme we use this property that one can sum the shares which are equal to the sum of the secrets.

Chapter 5

Electronic voting protocol

In this section we look at the electronic voting protocol described in [Sch99]. The protocol is based on the PVSS protocol described in the same article. We will only describe the electronic voting protocol but as doing so, the relevant elements from the PVSS protocol will be taking into the description. For the rest of this chapter we will be referring to the electronic voting protocol, as simply the protocol, unless specified otherwise.

Giving the complexity of the protocol we divided this description into three parts. In the first part we will describe the protocol as simple as possible, leaving out mathematical justification and proofs. We provide a calculated example for low values for the basic parts of the protocol in B.1. In second part we will be looking at the mathematical justification, describing this in the same order as in the first part. Finally we will look at the proofs in the last part.

5.1 The protocol

The overall concept of the protocol is to allow a group of voters to cast there votes in the form "no" or "yes" , and publish the result to a public bulletin board. When a deadline is reach or when all votes have been casted a group of talliers takes the votes and calculates the end result. Using MPC and PVSS protocols this is done in such a way that none other then the voter knows the value of his vote, but everyone can validates the correctness and consistence of the vote. Below we present a simple overview showing the progress of the protocol.

Short overview of the protocol The protocol is divided into three main parts, each part representing a different phase in the election, roughly put one can say pre-election, the election and post-election.

1. **Initialization** This represents the pre-election phase where the preparation for the actually election happens. Here the requirements for the election is demented and the different authorities is registered.

   (a) The system publishes the system parameters and the security requirements. In our implementation an admin user will log into the

system and start the election. The admin will choose the security parameter and the amount of required talliers. Last he will initiate the computation of the system parameters.

(b) Each tallier generates a private and a public key.

(c) Each tallier registries their public keys on the bulletin board.

(d) Each voter signs it credentials on the bulletin board.

2. **Ballot casting** This represent the actually election, where the registered voters castes there votes and publish the result to the bulletin board.

   (a) Each voter votes 1 or 0

   (b) Each voter generates a random secret.

   (c) Each voter creates shares of the secret to each tallier and encrypts it with the corresponding public key of the tallier. Each voter supply this secret share with evidence of its consistency with a $DLEQ$ proof.

   (d) Each voter supply evidence for a valid vote with $PROOF_U$.

3. **Tallying** This represents the post-election phase where the tallying authorities, which where registered in Initialization, counts the votes together and publish the result to the bulletin board.

   (a) At least t tallier accumulates and decrypts their shares.

   (b) One authority completes the final computation of the total votes.

In the following we will describe the central parts of the protocol, leaving out the complexity of mathematical justification and proofs. The idea is to get an understanding of how the protocol works without know exactly why it works.

For efficiency we will limit the computation of the votes to a finite number of talliers. There are m voters and n talliers.

5.1.1 Initialization

The bulletin board publishes all system parameters which is the public elements a prime q, the generators g and G and a security parameter t.

$$\begin{aligned} q &\in_R \{2^{l-1}, ..., 2^l\},\ where\ l > 1024 \\ f &\in_R \{2, ..., 2q-1\} \rightarrow g & = f^2\ mod\ 2q+1 \\ F &\in_R \{2, ..., 2q-1\} \rightarrow G & = F^2\ mod\ 2q+1 \\ t &\in \mathbb{Z}_q^* & = \{1, 2, 3, ..., q-1\} \end{aligned}$$

The prime q is uniformly randomly chosen from $\mathbb{Z}$, but in practise we use the subset 2^{l-1} to 2^l. l is chosen larger then 1024 because of the security requirements described in section 3.4.3. The generators g and G are computed as squares from the sets f and F, the reason for this is elaborated in section 5.2.1. $2q+1$ must also be a prime and the reason why we use $2q+1$ is because we are

working with Shamirs secret sharing in the exponents which also is described in section 5.2.1. t is chosen based on the system requirements for our fault tolerance against corrupted parties.

The tallier generates a private key x_i and a public key y_i.

$$Private\ key : x_i \in_R \mathbb{Z}_q^* = \{1, 2, 3, ..., q-1\}$$
$$Public\ key : y_i = G^{x_i},\ i \in \{1, 2, 3, ..., n\}$$

Every tally generates a uniformly randomly chosen private key from $\mathbb{Z}_q^*$ where q is a prime. The star notation refers to a set with none greatest common divisor larger then 1 respectively with q, which is described in section 3.3. The public key is computed on an exponentation on G, which essentially gives us the security of the discrete logarithm problem described in section 3.4.2.

5.1.2 Ballot casting

The Ballot casting consists of *distribution of the shares* and *verification of the shares.*

First the voter either votes "no" or "yes" corresponding to 0 or 1. The voter select a uniformly random secret $s \in \mathbb{Z}_q$. The PVSS protocol is then used to distribute shares which contain a combination of the secret s and the vote. Every voter will construct a random polynomial at degree $t-1$ and then evaluate the shares to each of the talliers.

The voter casts his vote, either 0 or 1. The voter creates a random secret s and a random polynomial of degree at most $t-1$ and computes the shares.

$$Vote : v \in \{0, 1\}$$
$$Random\ secret : s \in_R \mathbb{Z}_q$$
$$Random\ polynomium : p(x) = s + \alpha_1 x^1 + \alpha_2 x^2 +, ..., +\alpha_{t-1} x^{t-1},\ \alpha_j \in_R \mathbb{Z}_q$$
$$Secret\ Shares : p(0) = s,\ p(1),\ p(2), ...,\ p(n)$$

The degree of the polynomium is based on the security parameter t described in section 5.1.1. Each voter chooses uniformly random the coefficients α in Z_q and a random secret s and computes the shares using Shamirs secret sharing which is described in section 4.2.2.

The voter distributes the encrypted share and creates the proofs $PROOF_U$ and $DLEQ$.

$$Encryption\ of\ the\ share : Y_i = y_i^{p(i)}, 1 \leq i \leq n$$
$$Hidden\ vote : U = G^{s+v}$$

Each voter creates encrypted shares to tally 1, tally 2,..., tally n. The $p(i)$ refers to the share in a point corresponding to a given tally. The shares are encrypted

using the tallys public key y_i. U is a DL problem that hides the vote as the exponentiation. As the vote v only can hold the values 0 or 1 this wouldn't make a hard problem. By reusing the secret s and adding this to v, then given a large s, U should be a hard problem. The above is then published to the bulletin board.

Besides the above, the proofs $PROOF_U$ and $DLEQ$ are computed and published. The $PROOF_U$ proofs that the vote is either 0 or 1 without revealing the actual value of the vote. The $DLEQ$ proofs that the shares are constructed correctly and consistent. Both proofs are elaborated in details later in section 5.3

5.1.3 Tallying

Tallying is the process of counting the votes. Here the tallier uses their private keys to collectively compute the final tally, based on the valid ballots.

The tally decrypts their shares and publishes a $DLEQ$ proof

$$Multiplum\ of\ encrypted\ shares : Y_i^* = (\prod_{j=1}^{m} Y_{ij})\ (mod\ 2 \cdot q + 1)$$

The homomorphic secret sharing property ensures that each tally will be able to multiply the shares and then decrypt. Let Y_{ij} be the value Y_i computed by the j-th voter, which is the encrypted share $Y_i = y_i^{p(i)}$, as described in the section 5.1.2. This means that the i is referring to tally 1, tally 2 and tally 3 etc. and j is referring to voter 1, voter 2 and voter 3 etc. Y_i^* is then the multiplum of encrypted shares for a given tally i. Tally i is now able to decrypt the multiplum Y_i^* using his private key x_i.

$$Decrypted\ multiplum\ of\ shares : S_i^* = (Y_i^*)^{\frac{1}{x_i}}\ (mod\ 2 \cdot q + 1)$$

S_i^* is the decrypted multiplum of all tally i shares. As this is the multiplum of shares then no information of the individual share is revealed and tally i can safely publish the decrypted result to the bulletin board.
Note that besides decrypting the shares the talliers will publish a $DLEQ$ proof which shows that the decrypting was done correct. See figure A.2 of the $DLEQ$ proof. Also note that we need to computing the inverse of the key x_i. To compute the inverse we can use Extended Euclidean algorithm described in section 3.2.

A master authority applies Lagrange interpolation
After the tallier has published their decrypted shares S_i^* a master authority will be able to compute the sum of the secrets from the voters.

$$(S_1^*)^{\lambda_1 \cdot\ (mod\ q)} \cdot (S_2^*)^{\lambda_2\ (mod\ q)} \cdot (S_n^*)^{\lambda_n\ (mod\ q)}\ (mod\ 2 \cdot q + 1) = G^{\sum_{j=1}^{m} s_j}$$

With all the S_i^* we can compute the sum of the secrets. We apply the lambdas to exponents on S_i^*, which is computed from the Lagrange interpolation formular, λ_j from section 4.2.1.1. We can then multiply the S_i^* which can be reduces to the sum of the exponents, which are equal to the sum of the secrets.

A master authority computes the votes
The last step is to isolate the votes and then compute the final result. By multiplying U_j from the voters we obtain the following.

$$(\prod_{j=1}^{m} U_j) \ (mod\ 2 \cdot q + 1) = G^{\sum_{j=1}^{m} s_j + v_j}$$

From the previous step we computed $G^{\sum_{j=1}^{m} s_j}$. To isolate the sum of votes v in the exponent we can multiply $(\prod_{j=1}^{m} U_j)$ by the inverse of $(G^{\sum_{j=1}^{m} s_j})^{-1}$ in the following.

$$G^{\sum_{j=1}^{m} s_j + v_j} \cdot (G^{\sum_{j=1}^{m} s_j})^{-1} = G^{\sum_{j=1}^{m} v_j}$$

To solve the computing of the votes one can compute $G^0, G^1, G^3, ..., G^{v_j}$ by exhaustive search. The final vote count will be the exponent raised on G. A more efficient algorithm is to use Baby-step giant-step algorithm described in section 3.4.3.

5.2 Protocol details

In this part we will elaborate on the mathematical justification based on our explanation of the protocol.

5.2.1 Initialization

Elaboration of computation of $2q + 1$
In our implementation we will pick a prime, q, so we avoid doing the gcd computation. The protocol states that we have to compute in a group of order q. This means that when we are doing operations in the exponent this property should be satisfied $g^q = 1$ where q is prime. If we are doing $mod\ q$ in the exponent we have $g^q = g^0$. The reason for doing operation in the exponent $mod\ q$ is because we are using Sharmir secret sharing which require a finite field.

One can see that given a generator $g = 2$ and a prime $q = 5$, then $2^5\ mod\ 5 = 32\ mod\ 5 = 2$. For this to be true, we take the square of numbers modulo a prime in this form $2q + 1$. This is also called a strong prime. By using this mathematical structure this property holds. We can choose $b = a^2$. Then we see the property holds $b^q = 1\ mod\ 2q + 1$. Using the same values as before, it is clear that $(2^2)^5\ mod\ 11 = 1024\ mod\ 11 = 1$. Fermat little theorem states that $b^{q-1}\ mod\ q = 1$ where q is prime. So we know if we pick our q and b (as

a square) in this form $(a^2)^{q+1-1} \ mod \ 2q+1 = 1 = a^{2q} \ mod \ 2q+1 = 1$ the property holds. This means if we are working in the exponents we $(mod \ q)$ and if we are working in the bases we $(mod \ 2q+1)$.

Elaboration of the generators
The generators is randomly chosen in the set between 2 and $2q-1$. We remove 1 because if 1 get squared it will always gives 1 and can therefor not be used as a generator. We remove $2q$ because $(2q)^2 = 1 \ mod \ 2q+1$. That means if you square $2q$ it will also give 1 and can therefor not be used as a generator.

5.2.2 Ballot casting

Elaboration of computation of Y_i
The Y_i is the shares encrypted using the talliers public key y_i as described in section 5.1.2.

$$
\begin{aligned}
Voter\ 1\ &: Y_{1,1} = y_{1,1}^{p_1(1)}, Y_{2,1} = y_{2,1}^{p_2(2)}, .., Y_{n,1} = y_n^{p_n(n)} \\
Voter\ 2\ &: Y_{1,2} = y_{1,2}^{p_1(1)}, Y_{2,2} = y_{2,2}^{p_2(2)}, .., Y_{n,2} = y_n^{p_n(n)} \\
Voter\ m &: Y_{1,m} = y_{1,m}^{p_1(1)}, Y_{2,m} = y_{2,m}^{p_2(2)}, .., Y_{n,m} = y_n^{p_n(n)}
\end{aligned}
$$

When computing the encryption of the shares Y_i, then each voter will compute the above to each of the talliers. For clarification we add the tally to the notation such that we have Y_{ij} where i is the i-th tally and j is the j-th voter referring to $Y_{1,1},\ Y_{1,2}$ etc.

Constructing the variable C_j and X_i to proofs $DLEQ$ and $Proofu$
As briefly described in section 5.1.2, the voter publishes proofs that he have voted and distributed this vote accurately. The proof $Proofu$ uses the variable C_0 and the $DLEQ$ uses the variable X_i. In order to construct the proofs each voter creates the following variables.

$$
\begin{aligned}
&hidden\ coeffiens : C_j = g^{\alpha_j},\ j \in \{0,1,2,3,....,t-1\},\ where\ \alpha_0 = s \\
&multiplum\ of\ hidden\ coeffiens : X_i = \prod_{j=0}^{t-1} C_j^{i^j} = g^{p(i)},\ 1 \leq i \leq n
\end{aligned}
$$

C_j holds all the coeffiens α_j including the secret α_0, as these are hidden in the exponent of g, this is secured by the DL problem.

The $p(i)$ raised in the exponent of g is the polynomial $p(i) = \alpha_0 + \alpha_1 i^1 + \alpha_2 i^2 +, ..., +\alpha_{t-1} i^{t-1}$ created by each voter in the Ballot casting phase described in section 5.1.2. The way we can reduce the following statement $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$ to $g^{p(i)}$ is as follows.

$$X_i = \prod_{j=0}^{t-1} C_j^{i^j} = \prod_{j=0}^{t-1} (g^{\alpha_j})^{i^j} = g^{\sum_{j=0}^{t-1} \alpha_j \cdot i^j} = g^{\alpha_0 \cdot i^0 + \alpha_1 \cdot i^1 + \alpha_2 \cdot i^2, ..., \alpha_{t-1} \cdot i^{t-1}} = g^{p(i)}$$

To clarify, if the voter creates 3 shares, that means the voter has to compute X_1, X_2 and X_3 which means one X_i foreach share. This makes sense because the voter has to prove his honesty foreach of the shares.

$$X_1 = \prod_{j=0}^{t-1} C_j^{i^j} = C_0^{1^0} \cdot C_1^{1^1} \cdot C_2^{1^2} = g^{\alpha_0 \cdot 1^0 + \alpha_1 \cdot 1^1 + \alpha_2 \cdot 1^2} = g^{p(1)}$$
$$X_2 = \prod_{j=0}^{t-1} C_j^{i^j} = C_0^{2^0} \cdot C_1^{2^1} \cdot C_2^{2^2} = g^{\alpha_0 \cdot 2^0 + \alpha_1 \cdot 2^1 + \alpha_2 \cdot 2^2} = g^{p(2)}$$
$$X_3 = \prod_{j=0}^{t-1} C_j^{i^j} = C_0^{3^0} \cdot C_1^{3^1} \cdot C_2^{3^2} = g^{\alpha_0 \cdot 3^0 + \alpha_1 \cdot 3^1 + \alpha_2 \cdot 3^2} = g^{p(3)}$$

5.2.3 Tallying

Elaboration of computation of Y_i^* and S_i^*
As described in section 5.1.3 the Y_i^* is then the multiplum of encrypted shares for a given tally i and S_i^* is the decrypted multiplum of all tally i shares.

$$Multiplum\ of\ encrypted\ shares: Y_i^* = (\prod_{j=1}^{m} Y_{ij})\ (mod\ 2 \cdot q + 1) = y_i^{\sum_{j=1}^{m} p_j(i)}$$

$$Decrypted\ multiplum\ of\ shares: S_i^* = (Y_i^*)^{\frac{1}{x_i}}\ (mod\ 2 \cdot q + 1) = G^{\sum_{j=1}^{m} p_j(i)}$$

Y_i^* is equal to $y_i^{\sum_{j=1}^{m} p_j(i)}$ and S_i^* is equal to $G^{\sum_{j=1}^{m} p_j(i)}$.

$$y_i^{\sum_{j=1}^{m} p_j(i)} = (G^{x_i})^{\sum_{j=1}^{m} p_j(i)} = G^{x_i \sum_{j=1}^{m} p_j(i)} = (G^{x_i \sum_{j=1}^{m} p_j(i)})^{\frac{1}{x_i}} = G^{\sum_{j=1}^{m} p_j(i)}$$

We derive S_i^* by applying the talliers private key to Y_i^*. Note that we are raising to the multiplicative inverse of the private key in q. The p_j is the evaluations by the j-th voter. More concrete this can be written as the following.

$$Y_1^* = y_1^{(p_1(1)+p_2(1)+p_3(1),..,p_n(1))}, \quad S_1^* = G^{(p_1(1)+p_2(1)+p_3(1),..,p_n(1))}$$
$$Y_2^* = y_2^{(p_1(2)+p_2(2)+p_3(2),..,p_n(2))}, \quad S_2^* = G^{(p_1(2)+p_2(2)+p_3(2),..,p_n(2))}$$
$$Y_n^* = y_n^{(p_1(n)+p_2(n)+p_3(n),..,p_n(n))}, \quad S_n^* = G^{(p_1(n)+p_2(n)+p_3(n),..,p_n(n))}$$

Each tallier can publish S_i^* and Y_i^*. Note that the exponent on the y_i and G is the evaluation from each voter in some point in a given polynomial $h(1) = p_1(1) + p_2(1) + p_3(1), .., p_n(1),\ h(2) = p_1(2) + p_2(2) + p_3(2), .., p_n(2), ...,\ h(n) = p_1(n) + p_2(n) + p_3(n), .., p_n(n)$.

$$\text{Tally 1 publish: } S_1^* = G^{\sum_{j=1}^{m} p_j(1)} \text{ and } Y_i^* = y_i^{\sum_{j=1}^{m} p_j(1)}$$
$$\text{Tally 2 publish: } S_2^* = G^{\sum_{j=1}^{m} p_j(2)} \text{ and } Y_i^* = y_i^{\sum_{j=1}^{m} p_j(2)}$$
$$\text{Tally } n \text{ publish: } S_n^* = G^{\sum_{j=1}^{m} p_j(n)} \text{ and } Y_i^* = y_i^{\sum_{j=1}^{m} p_j(n)}$$

Elaboration of the step where the master authority applies Lagrange interpolation

$$(S_1^*)^{\lambda_1 \cdot \ (mod\ q)} \cdot (S_2^*)^{\lambda_2\ (mod\ q)} \cdot (S_n^*)^{\lambda_n\ (mod\ q)}\ (mod\ 2 \cdot q + 1) = G^{\sum_{j=1}^{m} s_j}$$

We can substitute the S^* with G. The final result in the exponents is a evaluation in some polynomium in 0 which corresponds to the sum of secrets s computed by the voters.

$$\begin{aligned}
&= G^{\sum_{j=1}^{m} \lambda_j p_j(1)} \cdot G^{\sum_{j=1}^{m} \lambda_j p_j(2)} \cdot ... \cdot G^{\sum_{j=1}^{m} \lambda_j p_j(n)} \\
&= G^{\sum_{j=1}^{m} \lambda_j p_j(1) + \sum_{j=1}^{m} \lambda_j p_j(2) + ... + \sum_{j=1}^{m} \lambda_j p_j(n)} \\
&= G^{\sum_{j=1}^{m} (\lambda_j p_j(1) + \lambda_j p_j(2) + , ..., + \lambda_j p_j(n))} = G^{\sum_{j=1}^{m} p_j(0)} = G^{\sum_{j=1}^{m} s_j}
\end{aligned}$$

The $\sum_{j=1}^{m} p_j(0)$ correspond to the sum of the secret values of s for the voters. More formal it corresponds to the evaluation of some polynomial $h(0) = s_1 + s_2, ..., s_n$.

Elaboration of the step where the master authority computes the votes

$$Sum\ of\ the\ secrets\ and\ the\ votes : (\prod_{j=1}^{m} U_j)\ (mod\ 2 \cdot q + 1) = G^{\sum_{j=1}^{m} s_j + v_j}$$

From the previous we saw that by multiplying U_j we obtained the sum of the secrets and the votes in the exponent. To recap more concrete we have the following.

The voters computes U

$$\begin{aligned}
Voter\ 1 : U_1 &= G^{s_1 + v_1} \\
Voter\ 2 : U_2 &= G^{s_2 + v_2} \\
Voter\ n : U_m &= G^{s_m + v_m}
\end{aligned}$$

The master authority mulitplies the U

$$(\prod_{j=1}^{m} U_j) = U_1 \cdot U_2 \cdot, ..., \cdot U_m = G^{s_1+s_2,...,s_m+v_1+v_2,...,v_m}$$

To isolate the votes in the exponent we multiply the $(\prod_{j=1}^{m} U_j)$ by the inverse of $(G^{\sum_{j=1}^{m} s_j})^{-1}$. This leads to the following mathematical justification.

$$\frac{G^{\sum_{j=1}^{m} s_j+v_j}}{G^{\sum_{j=1}^{m} s_j}} = G^{\sum_{j=1}^{m} s_j+v_j-\sum_{j=1}^{m} s_j} = G^{\sum_{j=1}^{m} v_j}$$

5.3 Proofs

In this section we will elaborate the mathematical justification of the proofs $DLEQ$ and $PROOF_U$. We will present an interactive and a non-interactive proof of the $DLEQ$ between the voter and the verifier. The non-interactive $DLEQ$ proof is elaborated in appendix A.1. As mentioned there is also a $DLEQ$ proof provided by the tallier which is elaborated in appendix A.2.

5.3.1 DLEQ interactive proof between voters and verifier

DLEQ stands for discrete logarithm equality and it proofs that the exponent are equal $X_i = g^{p(i)}$ and $Y_i = y_i^{p(i)}$ without revealing $p(i)$ and if the prover is honest, then it should be the case that we get the same computed values in the end meaning $a_1 = g^w = g^r \cdot X_i^C$ and $a_2 = y_i^w = y_i^r \cdot Y_i^C$. The prover must compute same amount of X_i as he creates shares. In practice this means that the prover supply a $DLEQ$ proof foreach of the shares. We will present the protocol and after that we will give concrete examples. Last we will describe the mathematical justification. An example of the proof is calculated in appendex B.2.

The verification of the shares in the interactive proof happens by the following interaction between the prover and the verifier.

DLEQ protocol

Input : g, X_i, y_i, Y_i *where* $X_i = g^{\alpha_i}$ *and* $Y_i = y_i^{\alpha_i}$

	Prover		Verifier
Step 1	$w \in_R \mathbb{Z}_q$		
	$a_1 = g^w$		
	$a_2 = y_i^w$	$\xrightarrow{a_1,\ a_2}$	
Step 2			$C \in_R \mathbb{Z}_q$
		$\xleftarrow{C}$	
Step 3	$r = w - p(i) \cdot C$		
			checks if :
Step 4		$\xrightarrow{r}$	$a_1 = g^r \cdot X_i^C$
			$a_2 = y_i^r \cdot Y_i^C$

Figure 5.1: $DLEQ$ interactive

Note that the verifier sends a challenge C, after the prover has computed a_1 and a_2. The check only passes if the prover used the same exponents. The proof shows that there exist some element α such that $g^\alpha = X_i$ and $y_i^\alpha = Y_i$. In the following there is an argument why $DLEQ$ works through Zero knowledge proof.

Correctness for $DLEQ$
Correctness means if the prover is honest and the statement is true, then the honest verifier always accept. Correctness is shown by verifying the $a_1 = g^w \stackrel{?}{=} g^r \cdot X_i^C$ and $a_2 = y_i^w \stackrel{?}{=} y_i^r \cdot Y_i^C$ are well constructed. Correctness for a_1 is shown by the following.

$$\begin{aligned} a_1 &= g^r \cdot X_i^C \\ &= g^r \cdot (g^{p(i)})^C \\ &= g^r \cdot g^{p(i)\cdot C} \\ &= g^{r+p(i)\cdot C} \\ &= g^{w-p(i)\cdot C+p(i)\cdot C} = g^w \end{aligned}$$

Correctness for a_2 is shown by the following.

$$\begin{aligned} a_2 &= y_i^r \cdot Y_i^C \\ &= y_i^r \cdot (y^{p(i)})^C \\ &= y_i^r \cdot y^{p(i)\cdot C} \\ &= y_i^{r+p(i)\cdot C} \\ &= y_i^{w-p(i)\cdot C+p(i)\cdot C} = y_i^w \end{aligned}$$

Example on $DLEQ$ **and why we need a random challenge**
We will show a concrete example why we need a challenge and it needs to be random. If the challenge is already known by the prover, for example assume it is 1, then the prover can "prepare" and cheat with a wrong statement.

1. The prover sends $a_1 = g^6$, $a_2 = y_i^7$, $X_i = g^2$ and $Y_i = y_i^3$ to verifier.
2. The verifier creates a challenge $C = 1$ to prover.
3. The prover computes $r = w - p(i) \cdot C = 6 - 2 \cdot 1 = 4$ and sends r to verifier.
4. The verifier knows the following $a_1 = g^4 \cdot X_i$ and $a_2 = y_i^4 \cdot Y_i$ and now he verifies:
   (a) The verifier checks if: $a_1 = g^4 \cdot X_i^C = g^4 \cdot g^{2 \cdot 1} = g^6$
   (b) The verifier checks if: $a_2 = y_i^4 \cdot Y_i^C = y_i^4 \cdot y_i^{3 \cdot 1} = y_i^7$

Even though the exponents are not the same, both checks passes, despite that the prover is dishonest. This shows that if there is no random challenge there wouldnt be soundness because the prover could cheat.

Example on $DLEQ$ with a random challenge which satisfies soundness
Next example shows the verification with a random challenge.

1. The prover sends $a_1 = g^6$, $a_2 = y_i^7$, $X_i = g^2$ and $Y_i = y_i^3$ to verifier.
2. The verifier creates a challenge $C = 9$ $(mod\ 5)$ to prover.
3. The prover computes $r = w - p(i) \cdot C = 6 - 2 \cdot 4$ $(mod\ 5) = 3$ and sends r to verifier.
4. The verifier knows the following $a_1 = g^3 \cdot X_i$ and $a_2 = y_i^3 \cdot Y_i$ and now he verifies:
   (a) The verifier checks if: $a_1 = g^3 \cdot X_i^C = g^3 \cdot g^{2 \cdot 4} = g^3 \cdot g^3 = g^6 = g^1 = g$
   (b) The verifier checks if: $a_2 = y_i^3 \cdot Y_i^C = y_i^3 \cdot y_i^{3 \cdot 4} = y_i^3 \cdot y_i^{12} = y_i^3 \cdot y_i^2 = y^5 = y^0 = 1$

Note that soundness is fulfilled because the check doesn't pass because $a_1 = g^6$ is different from $a_1 = g$ and $a_2 = y_i^7$ is different from $a_2 = y_i^0$. Recall that soundness is if the statement is false then it should fail with overwhelming probability.

The mathematical justification for soundness
In the following we are showing that a prover will fail with overwhelming probability if he is dishonest which satisfies soundness. Since the verifier doesent know if the first step $X_i = g^{p(i)}$ and $Y_i = y_i^{p(i)}$ has been computed correctly. We denote these exponents by $a_1 = g^w$ and $a_2 = y_i^{w'}$ and $X_i = g^{m_i}$ and $Y_i = y_i^{m_i'}$. We know $a_1 = g^w$ is equal to $g^r \cdot X_i^C = g^r \cdot g^{m_i \cdot C} = g^{r + m_i \cdot C}$ and $a_2 = y_i^{w'}$ is equal to $y_i^r \cdot Y_i^C = y_i^r \cdot y_i^{m_i' \cdot C} = y_i^{r + m_i' \cdot C}$. Based on this we can now write two inequalities. In order for these inequalities to be true we can rewrite.

$$a_1 = g^w \ = g^r \cdot X_i^C = g^r \cdot g^{m_i \cdot C} = g^{r + m_i \cdot C}$$
$$a_2 = y_i^{w'} = y_i^r \cdot Y_i^C = y_i^r \cdot y_i^{m_i' \cdot C} = y_i^{r + m_i' \cdot C}$$

We can now write two inequalities

$$w = r + m_i \cdot C \ (mod\ q) \implies r = w - m_i \cdot C \ (mod\ q)$$
$$w^{'} = r + m_i^{'} \cdot C \ (mod\ q) \implies r = w^{'} - m_i^{'} \cdot C \ (mod\ q)$$

These two inequalities has to be equal, therefor we can rewrite

$$w - m_i \cdot C = w^{'} - m_i^{'} \ (mod\ q) \implies (w - w^{'}) - (m_i - m_i^{'}) \cdot C = 0 \ (mod\ q)$$

The prover has to be honest if this equation must be true. It is overwhelming unlikely that, if the prover has been dishonest, where $m_i \neq m_{i'}$, that he will succeed. Since the C is known afterwards the construction of w and $w^{'}$ the probability will be $\frac{1}{q}$ for a convincing the verifier. Lets clarify with an example with a dishonest prover $C = 5$, $(w - w^i) - (m_i - m_i^{'}) = 2$ and $q = 5$. The dishonest prover will then succeed because $2 \cdot 5 = 0 \ (mod\ 5)$. Since the q is a large number the dishonest prover should fail with overwhelming probability.

Zero knowledge
The zero knowledge in this context means that the verifier doesn't learn anything about the $p(i)$. One way to argue zero knowledge is by showing that the values sent in the protocol doesn't depend on the $p(i)$. So if one can construct the values $a_1,\ a_2,\ r$ without knowing $p(i)$ shows that they do not depend on $p(i)$ and we thereby do not learn anything about $p(i)$. One way of doing this is though experiment where on can change the order of the protocol. In is out of this thesis scope to go further depth on this subject.

5.3.2 Description of $\mathbf{PROOF_U}$

In this section we show with $PROOF_U$ that the voter either votes 1 or 0. This is achieved by the voter proving that there is consistency between the exponents of how U and C_0 is constructed from $U = G^{s+v}$ and $C_0 = g^s$. The exponents only vary when 1 or 0 is voted. We will show the interactive proof and through Fiat–Shamir we transform an interactive proof of knowledge into a non-interactive proof of knowledge. The protocol illustration includes both scenarios where the voter votes either 0 or 1. If the voter votes 0 step 1a, 2, 3a, 4 will be followed. If the voter votes 1 step 1b, 2, 3b, 4 will be followed. An example of the proof is calculated in appendex B.3.

$PROOF_U$ protocol

$Public: U = G^{s+v},\ C_0 = g^s$

	Prover	Verifier
Step 1a	$if\ vote(v) = 0$ $w \in_R \{1, ..., q-1\}$, $r_1 \in_R \{1, ..., q-1\}$, $d_1 \in_R \{1, ..., q-1\}$, $a_0 = g^w$, $a_1 = g^{r_1} \cdot C_0^{d_1}$, $b_0 = G^w$, $b_1 = G^{r_1} \cdot (\frac{U}{G^{1-v}})^{d_1} = G^{r_1} \cdot (\frac{U}{G})^{d_1}$	
	$\xrightarrow{a_0, a_1, b_0, b_1}$ **Publish to bulletin**	
Step 1b	$if\ vote(v) = 1$ $w \in_R \{1, ..., q-1\}$, $r_0 \in_R \{1, ..., q-1\}$, $d_0 \in_R \{1, ..., q-1\}$, $a_0 = g^{r_0} \cdot C_0^{d_0}$, $a_1 = g^w$, $b_0 = G^{r_0} \cdot (\frac{U}{G^{1-v}})^{d_0} = G^{r_0} \cdot U^{d_0}$, $b_1 = G^w$	
	$\xrightarrow{a_0, a_1, b_0, b_1}$ **Publish to bulletin**	
		Publish to bulletin
Step 2		$C \in_R \{0, ..., q-1\}$ $\xleftarrow{C}$
Step 3a	$if\ vote(v) = 0$ $d_0 = C - d_1\ mod\ q$, $r_0 = w - s \cdot d_0\ mod\ q$	
	$\xrightarrow{d_0,\ r_0,\ d_1,\ r_1}$ **Publish to bulletin**	
Step 3b	$if\ vote(v) = 1$ $d_1 = C - d_0\ mod\ q$, $r_1 = w - s \cdot d_1\ mod\ q$	
	$\xrightarrow{d_0,\ r_0,\ d_1,\ r_1}$ **Publish to bulletin**	
Step 4		**Verification:** $C = d_1 + d_0$, $a_0 = g^{r_0} \cdot C_0^{d_0}$, $b_0 = G^{r_0} \cdot U^{d_0}$, $a_1 = g^{r_1} \cdot C_0^{d_1}$, $b_1 = G^{r_1} \cdot (\frac{U}{G})^{d_1}$

Figure 5.2: $PROOF_U$

Explanation of the protocol
In step 1 the voter publish $a_0,\ b_0,\ a_1,\ b_1$. Note that the difference between voting 0 or 1 is just by swapping the values between the variables $a_0,\ b_0$ and $a_1,\ b_1$. The point is that the verifier will not be able to distinguish the value of the vote and thereby gain knowledge about the vote.

Step 1: Voter either votes 0 or 1:

1. Voter votes 0 and creates: $v = 0,\ w \in_R \{1, ..., q-1\},\ r_1 \in_R \{1, ..., q-1\},\ d_1 \in_R \{1, ..., q-1\}$ and publish: $a_0 = g^w,\ b_0 = G^w,\ a_1 = g^{r_1} \cdot C_0^{d_1},\ b_1 = G^{r_1} \cdot (\frac{U}{G^{1-v}})^{d_1} = G^{r_1} \cdot (\frac{U}{G})^{d_1}$.
2. Voter votes 1 and creates: $v = 1,\ w \in_R \{1, ..., q-1\},\ r_0 \in_R \{1, ..., q-1\},\ d_0 \in_R \{1, ..., q-1\}$ and publish: $a_1 = g^w,\ b_1 = G^w,\ a_0 = g^{r_0} \cdot C_0^{d_0},\ b_0 = G^{r_0} \cdot (\frac{U}{G^{1-v}})^{d_0} = G^{r_0} \cdot (\frac{U}{1})^{d_0} = G^{r_0} \cdot U^{d_0}$

Step 2: The verifier creates a challenge $C \in_R \{0, ..., q-1\}$ to the voter.

Step 3: The outcome from step 3 is that the voter publish $d_0,\ r_0,\ d_1,\ r_1$. Note that the voters computation depends on the challenge from the interaction between the verifier. Voter either votes 0 or 1:

1. Voter votes 0 computes: $v = 0,\ d_0 = C - d_1\ mod\ q,\ r_0 = w - s \cdot d_0\ mod\ q$
2. Voter votes 1 computes: $v = 1,\ d_1 = C - d_0\ mod\ q,\ r_1 = w - s \cdot d_1\ mod\ q$

Step 4: In step 4 the verifier will be able computes and verify consistency. $C = d_1 + d_0,\ a_0 = g^{r_0} \cdot C_0^{d_1},\ b_0 = G^{r_0} \cdot U^{d_0},\ a_1 = g^{r_1} \cdot C_0^{d_1},\ b_1 = G^{r_1} \cdot (\frac{U}{G})^{d_1}$

We can turn this into a non-interactive proof by replacing step 2 with the voter using a hash function and thereby avoiding interaction with the verifier. The prover will then compute a hash of $C = H(U,\ C_0,\ a_0,\ b_0,\ a_1,\ b_1)$. Because of time constraint we are not able to elaborate further on this optimization.

Mathematical justification for correctness
In the following we derive the mathematical justification from step 4. Here we will replace with earlier expression from above and replace by the actual value of the vote. This means is that the math depends on the value of the vote.

Explanation of $a_0 = g^{r_0} \cdot C_0^{d_0}$

The voter votes 0 and we show that $a_0 = g^w \stackrel{?}{=} g^{r_0} \cdot C_0^{d_0}$ is well constructed

$$\begin{aligned} a_0 &= g^{r_0} \cdot C_0^{d_0} \\ &= g^{w - sd_0} \cdot g^{sd_0} \\ &= g^{w - sd_0 + sd_0} \\ &= g^w \end{aligned}$$

The voter votes 1 is trivial because a_0 is constructed from $a_0 = g^{r_0} \cdot C_0^{d_0}$

Explanation of $b_0 = G^{r_0} \cdot U^{d_0}$

The voter votes 0 and we show that $b_0 = G^w \stackrel{?}{=} G^{r_0} \cdot U^{d_0}$ is well constructed

$$\begin{aligned} b_0 &= G^{r_0} \cdot U^{d_0} \\ &= G^{w-sd_0} \cdot G^{(s+0)\cdot d_0} \\ &= G^{w-sd_0} \cdot G^{sd_0} \\ &= G^w \end{aligned}$$

The voter votes 1 and we show that $b_0 = G^{r_0} \cdot (\frac{U}{G^{1-1}})^{d_0} \stackrel{?}{=} G^{r_0} \cdot U^{d_0}$

$$\begin{aligned} b_0 &= G^{r_0} \cdot (\frac{U}{G^{1-v}})^{d_0} \\ &= G^{r_0} \cdot (\frac{U}{G^0})^{d_0} \\ &= G^{r_0} \cdot (\frac{U}{1})^{d_0} \\ &= G^{r_0} \cdot U^{d_0} \end{aligned}$$

Explanation of $a_1 = g^{r_1} \cdot C_0^{d_1}$

The voter votes 1 and we show that $a_1 = g^w \stackrel{?}{=} g^{r_1} \cdot C_0^{d_1}$ is well constructed

$$\begin{aligned} a_1 &= g^{r_1} \cdot C_0^{d_1} \\ &= g^{w-sd_1} \cdot g^{sd_1} \\ &= g^{w-sd_1+sd_1} \\ &= g^w \end{aligned}$$

The voter votes 0 is trivial because a_1 is constructed from $a_1 = g^{r_1} \cdot C_0^{d_1}$.

Explanation of $b_1 = G^{r_1} \cdot (\frac{U}{G})^{d_1}$

The voter votes 1 and we show that $b_1 = G^W \stackrel{?}{=} G^{r_1} \cdot (\frac{U}{G})^{d_1}$ is well constructed

$$\begin{aligned} b_1 &= G^{r_1} \cdot (\frac{U}{G})^{d_1} \\ &= G^{w-sd_1} \cdot (U \cdot G^{-1})^{d_1} \\ &= G^{w-sd_1} \cdot (G^{s+1})^{d_0} \cdot G^{-d_1} \\ &= G^{w-sd_1} \cdot G^{sd_0+d_0} \cdot G^{-d_1} \\ &= G^w \end{aligned}$$

The voter votes 0 and we show that $b_1 = G^{r_1} \cdot (\frac{U}{G})^{d_1} \stackrel{?}{=} G^{r_1} \cdot (\frac{U}{G^{1-v}})^{d_1}$ is well constructed

$$\begin{aligned} b_1 &= G^{r_1} \cdot (\frac{U}{G^{1-v}})^{d_1} \\ &= G^{r_1} \cdot (\frac{U}{G^{1-0}})^{d_1} \\ &= G^{r_1} \cdot (\frac{U}{G})^{d_1} \end{aligned}$$

We will not go to depth in Soundness and Zero knowledge for $PROOF_U$. Here we will refer to another paper. [CDS94]

Part III

Practical

Chapter 6

Designing the application

The first part about Voting, Mathematical understanding, MPC and Electronic voting protocol served as an introduction to one of the goals of this thesis: an implementation of the electronic voting protocol. This chapter will describe and discuss our theoretical aspects behind our implementation of the electronic voting application.

6.1 Introduction

One of the goals of the thesis is to design and implement a scalable electronic voting application. As software engineers our focus is on the software architecture and we will follow the definition from [BCK12].

Definition 6.1.0.1: Software architecture

The software architecture of a system is the set of **structures** needed to reason about the system, which comprise software **elements**, **relations** among them, and the **properties** of both.

Our approach towards implementing a software architecture is based on a systematic analysis of the demands for the application. To achieve this we will use methods and techniques from previous causes in Software architecture such as Quality attribute workshop [BEL+03], Quality attribute scenario (QAS) and architectural decisions etc. A quality attribute (QA) according to [BCK12] is as follows.

Definition 6.1.0.2: Quality attribute

..A quality attribute is a measurable or testable property of a system that is used to indicate how well the satisfies the needs of its stakeholders..

A core observation is that a QA should be measurable or testable quality. The key point is when working with QA we use them in a given context/scenario and therefor we informally call these as QAS.

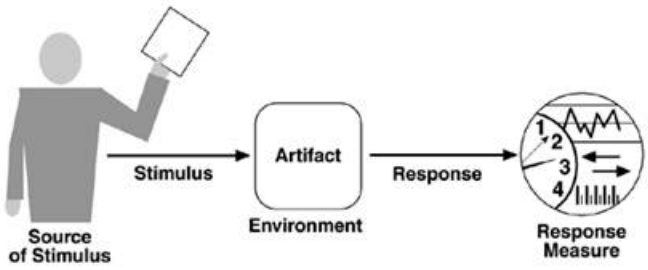


Figure 6.1: The parts of a quality attribute scenario

One of the core aspects of the definition of software architecture, is that software architecture is a set of structures, which we can use to reason about the system. To assist and visualize these structures, elements, relations and properties we use Module-, Component & Connector (C&C)- and Allocation viewpoints [CCH16].

1. Module viewpoint is concerned with how functionality of the system maps to static development units. The focus will be on elements such as classes and interfaces and relationships such as associations, generalizations, realizations and dependencies.

2. Component & Connector viewpoint is concerned with the runtime mapping of functionality to components of the architecture. Components are the executing things that perform a function. Connectors are the communication channels between components. The purpose is to focus on the flow of data and responsibilities such as a network call or method call etc.

3. Allocation viewpoint is concerned with how software entities are mapped to environmental entities. Here the focus are on the physical stuff such as computer or a network. We specify the environment in order to make the software running.

These viewpoint originates from $3 + 1$ article [CCH16], where the $+1$ is the architectural requirements. These architectural requirements can be formulated through QAS.

We will discuss different tactics on software architecture for achieving the business goal for the electronic voting application. A tactic according to [BCK12] is defined as.

Definition 6.1.0.3: Tactic

Tactic is a design decision that influences the achievement of a quality attribute response.

We will introduce a case which gives an overall description of how a user creates a vote through an electronic voting application. The purpose of the case is describing business/mission requirement of the electronic voting application. Here we will emphasize that it should be clear it reflects the security requirements

from the first part. We use the general security requirement for an electronic voting scheme described in chapter 2 as functional requirements for the application. These requirements are well studied and discussed and should be comprehensive for an electronic voting scheme.

6.2 Method

To ensure a solid systematic approach for this part of the thesis we will use

1. Case
2. Functional requirements
3. Quality attribute workshop
   (a) Quality attributes
   (b) Quality attribute scenario
4. Tactics

Case The purpose of the case is an informal description of the requirements for the electronic voting application. We use this as a introduction to a business/mission for the most important requirements for the electronic voting application.

Functional requirements The purpose of the functional requirement is to capture the electronic voting application behavior based on the case description. However as described in the introduction to this chapter, we will be using the security requirements of a general electronic voting scheme as functional requirements for our application. Even though the functional requirements is not in focus in this thesis we will use these requirements as guidelines to help define the architecture. Therefor throughout this chapter we will only be referring to the requirements in relations with the architecture requirements.

Quality attribute workshop The purpose of the Quality Attribute Workshops (QAWs) is a systematic method for identifying a system's architecture critical quality attributes, such as availability, security and modifiability, that are derived from mission or business goals. For the scope of this thesis we will follow the phases in the QAW on a theoretical level, to derive the most important QA for the electronic voting application. We will use the structure of the QAW but we will not hold a practical workshop. Based on the QA we will formulate the most important Quality attribute scenarios and describe related tactics.

Tactics For the selected QAS we will describe tactics that meets QAS response. For a given tactic we will describe the tactic and support it with diagrams which illustrate the influence on the architecture.

6.3 General design concepts

In the follwoing we will explain some general design concepts which will be used through out our software sections [Chr10].

Design pattern A solution to a repeated design problem in a given context.

Maintainability (According to ISO 9126) The capability of a software product to be modified. Modifiaction may include corrections, improvements or adaptation of the software to changes in the enviroment and in requirements and functinal specifications.

Variability point A well defined section of the code whose behavior it should be possible to vary.

Change by addition We are only adding new code instead of modifying existing code.

Coupling Coupling is the degree of how dependent one software module is on other software modules.

Cohesion Cohesion refers to the degree of how related the responsibility of a software module belong together.

Test stubs Test stubs are replacements that simulate the behaviors of a software module that a module undergoing tests depends on.

The compositional process is as follows [Chr10].

1. Identify the behavior that varies.
2. Always program against an interface which encapsulate the variable behavior.
3. Delegate the responsibility to a specialized class which handle the concrete responsibility.

6.4 Case

A user applies for voting for a given election and a registration authority will either accept or reject his application. If the user is accepted then he should be able to logon a voting page and cast his vote. When all the registered voters have casted their votes or a deadline is reached, the system should tally the votes and then publish the result on a webpage. of cause only valid votes are included in the tallying process. The tally process should be handled by registered talliers. During this process none of the talliers should be able relate a vote to a voter. Nor should a tally be able to manipulate the tallying process by either adding, removing or alter votes. Each tally should be hold accountable for his participant in the tallying process. If the tally is discovered in cheating he is replaced by another tally during the tallying process. This replacement should not have influence on the result nor should it required re-election.

6.5 Architectural requirements

6.5.1 Quality Attributes workshop

In this section we will present 8 phases of the QAW as described in [BEL+03]. Despite that a workshop is not held we still see clear benefits by using this model namely to determine the qualities for the electronic voting application before it is implemented. We are well aware that the outcome of the workshop is not perfect and will only cover the perspectives from a software developer and the security requirements.

QAW Presentation and Introductions QAW facilitators describe the motivation for the QAW and explain each step of the method.

Business/Mission Presentation A representative of the stakeholder community presents the business and/or programmatic drivers for the system.

Architectural Plan Presentation A technical stakeholder presents the system architectural plans as they stand with respect to early documents, such as high-level system descriptions, context drawings, or other artifacts that describe some of the system's technical details.

Identification of Architectural Drivers Architectural drivers often include high-level requirements, business/mission concerns, goals and objectives, and various quality attributes. During this step, the facilitators and stakeholders reach a consensus about which drivers are key to the system.

Scenario Brainstorm Stakeholders generate real-world scenarios for the system. Scenarios comprise a related stimulus, an environmental condition, and a response. Facilitators ensure that at least one scenario addresses each of the architectural drivers identified in Step 4.

Consolidation Scenarios that are similar in content are consolidated.

Prioritization Stakeholders prioritize the scenarios through a voting process.

Refinement The top four or five scenarios are further clarified and the following are described:

1. the business/programmatic goals that are affected by those scenarios
2. the relevant quality attributes associated with those scenarios

The above concludes the overview of the phases in the workshop. We will now use the structure of the phases to derive and define the QAS which will be the basic for our architecture.

6.5.1.1 3. Architectural Plan Presentation

We will use this step to wrap up our first iteration for a high level overview of the system. This first overview is all based on the knowledge we got from the first part. We illustrate this overview through a component connector view. This viewpoint is concerned with the run-time functionality of the system. At this point we have a component for a voter, tallier, observer, bulletin board and a

registration process. We see the voter, tallier and the observers as clients and as active processes which all are communicating with the bulletin board. Therefor their must be a connector between the clients and bulletin board. The purpose of the registration authority is that every active participant must be registered through some registration authority before they can attend the electronic voting election.

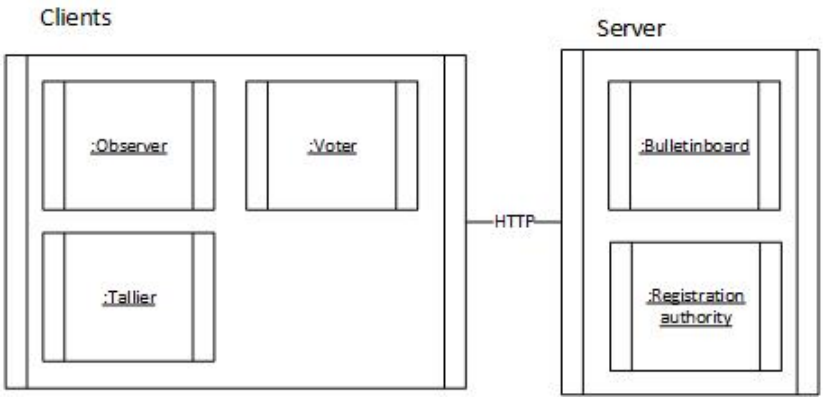


Figure 6.2: Initial draft of component & connector viewpoint

6.5.1.2 4. Identify architectural drivers

The purpose of this step is to identify architectural drivers. Architectural drivers are the keys to realizing quality attribute goals for the system. The architectural drivers are often found through requirements and business goals. We found architectural drivers by reflection and discussion against the security requirements of earlier described electronic voting application. Based on our architectural drivers, we begin to see the following quality attributes.

Voters point of view	
Usability	The system should be easy to use for a voter and give feedback
Availability	The system should be available when needed and if errors occours then the user should be least affected by this.
Interoperability	A voter should be able to cast a vote from a given device with a internet connection

Table 6.1: Voters point of view

Robustness	
Performence	Should be able to handle a large amount of user within a reasonable time
Testability	Given the nature of systems complexity it should be easily testable to ensure robustness and reliability
Security	The integrity should withhold even though if cheating occurs.
Universal verifiability	
Interoperability	Not only participant but also passive observers should be able to validate through out the election and afterwards.
Future proof	
Modifiable	Only a registered user should be able to vote. This registration should be easily replaceable depending on the nature of the election.
Modifiable	The system should modifiable such that core elements are replaceable

Table 6.2: Owners point of view

6.5.1.3 5. Scenario brainstorming

The purpose of this step is to design QAS, based on our architectural drivers. That is, here we form QAS in a form such as described in [BCK12]. Here there should be a clear stimulus and response measurement.

Scenario #	Description
1	A user should be able to cast a vote under runtime and the PVSS client registers the vote with a confirm message, within 2 minutes experimentation.
2	An internal crash occurs and the bulletin board is out of reach during normal operation. The response is that the error is logged and the system is running in degraded mode. The system should be up running within 5 minutes.
3	A PVSS client cast a vote to the bulletin board from a given device with internet connection under runtime and the system is updated and 100% of the information is exchanged and processed correctly.
4	An observer client validates a vote from the the bulletin board under runtime. The validation is processed, and the bulletin board is updated and 100% of the information is exchanged and processed correctly.
5	5 mill. users intiate their votes to the bulletin board under normal operation. The votes are processed and saved with average latency of 2 seconds.

6	An unitester should be able to code a unit on the system under development and the test suite are executed and result are captured and 85% of the system are coverage within 3 hours.
7	A developer should be able to make a change to the registration code under runtime and the change are made and tested within 3 hours.
8	A developer should be able to make a change to the random number generator code under runtime and the change are made and tested within 3 hours.
9	A cheater cast a invalid vote to the bulletin board under normal operation. All valid data should be preserved and the system should be able to detect invalid from valid data before the total counting of the votes.

Table 6.3: First step towards quality attributes scenarios

6.5.1.4 6. Scenario consideration

The purpose of this step is to merge scenarios that have similar features. At the workshop there were no scenarios that could be merged.

Usability is very vague described. It turns out that usability is closely related to modifiability in terms of preparing the architecture so the user interface can easily be replaced. Therefore, it's about building an architecture where business logic does not have a high coupling with the user interface, so one can easily add a new user interface.

Scenario 3 and 4 are both about *Interoperability*. We choose to merge scenario 4 with scenario 3, which means that scenario 4 is removed. This means that scenario 3 should contain the parts where the votes get validated. And of cause the scenario should take into account that an observer client should be able to communicate with the bulletin board.

6.5.1.5 7. Scenario prioritization

The purpose of this step is to draw up a priority list based on the total votes. The list is long but we will limit this thesis to focusing on a few and the rest we will comment. The list is primarily prioritized based on the security requirements.

Scenario 3 There is focus on the fact that as many people as possible can access electronic voting application. One way to achieve this is to create the application as a web application. This ensure that everyone the have an internet connection and a device with a modern browser should be able to access the application. In addition to a web application, we want to ensure that as many devices as possible can communicate with our electronic voting system. One way to ensure this, is to construct a standardized interfacing which is compatible with different clients.

Scenario 9 Since the PVSS is a public verifiable protocol, there will be a high focus on fulfilling the *Universal verifiability* requirement. This means if there are invalid votes they should be detected through public verification. The PVSS protocol has proves which ensures that if there is a "cheater" who cheats with their vote they will high with probability be discovered.

Scenario 6 To ensure *Accuracy* as in that final tally is computed correctly, there will be high focus on ensuring the complex part of the code is testable.

Scenario 7 To ensure *Eligibility* and *Uniqueness* there will be high focus on creating a flexible registration module which first of all ensures authentication and authorization such that only registered voters can vote and only have permissions to vote one time. But the module should be flexible enough to be change to integrate to other registration data.

Scenario 8 There will be high focus on ensuring the *Fairness* property regarding that none should be able to gain any knowledge of the outcome of the election. Also the *Uncoercibility* property part regarding no one should be able to extract the value of a vote. Therefor the need to have a module which can generate/compute large numbers. The security can change over time and the need of computing larger numbers will therefor be a demand. The code should therefor be flexible if there is need for replacing the module with another number generator.

Scenario 2 If the electronic voting application should be used by many participants then the architecture must be able to carry out the task it is supposed to do when needed. The architecture must be ready to mask or repair its errors within a given time period.

Scenario 5 If the electronic voting application should be used by many participants then the architecture must be ready to handle such a load.

Scenario 1 The electronic voting application should be easy to use. One way of finding the best candidate to an user interface is to create usability test. Based on the test result we must be able to change the user interface with breaking the whole codebase.

6.5.1.6 8. Scenario Refinement

The purpose of this step is to form QAS, where divide into source, stimulus, artifact, environment, response and response measure. We will work on the first five of the scenarios. The rest can be found in appendex C.1, C.2 and C.3.

Scenario(s):		# 1: A PVSS client cast a vote to the bulletin board from a given device with internet connection under runtime and the system is updated and 100% of the information is exchanged and processed correctly
Relevant Quality Attributes:		Interoperability
Scenario Parts	**Source:**	A webbrowser
	Stimulus:	Cast a vote
	Artifact	Bulletin board system
	Environment:	Runtime
	Response:	If the vote is valid it is accepted. If the vote is invalid it is rejected and the vote is removed from the bulletin board
	Response Measure:	100 % of the information is exchange and processed correctly

Table 6.4: Interoperability QAS

Scenario(s):		# 2: A cheater cast a invalid vote to the bulletin board under normal operation. All valid data should be preserved and the system should be able to detect invalid from valid data before the total counting of the votes
Relevant Quality Attributes:		Security
Scenario Parts	**Source:**	A cheater
	Stimulus:	Cast a invalid vote, with purpose of influencing the finale votes
	Artifact	Bulletin board system
	Environment:	Runtime
	Response:	If the vote is valid it is accepted. If the vote is invalid it is rejected and the vote is removed from the bulletin board
	Response Measure:	100 % of the information is exchange and processed correctly

Table 6.5: Security QAS

Scenario(s):		# 3: An unitester should be able to code a unit on the system under development and the test suite are executed and result are captured and 85 % of the system are coverage within 3 hours.
Relevant Quality Attributes:		Testability
Scenario Parts	**Source:**	Unittester
	Stimulus:	Code unit completed
	Artifact	Code
	Environment:	Design time
	Response:	Result captured
	Response Measure:	85 % path coverage in three hours

Table 6.6: Testability QAS

Scenario(s):		# 4:A developer should be able to make a change to the registration code under runtime and the change are made and tested within 3 hours
Relevant Quality Attributes:		Modifiability
Scenario Parts	**Source:**	Developer
	Stimulus:	Needs to replace the registration module
	Artifact	Code
	Environment:	Design time
	Response:	Replacement made and Unit tested
	Response Measure:	In three hours

Table 6.7: Modifiability QAS

Scenario(s):		# 5: A developer should be able to make a change to the random number generator code under runtime and the change are made and tested within 3 hours.
Relevant Quality Attributes:		Modifiability
Scenario Parts	**Source:**	Developer
	Stimulus:	Needs to replace the random generator
	Artifact	Code
	Environment:	Design time
	Response:	Replacement made and Unit tested
	Response Measure:	In three hours

Table 6.8: Modifiability QAS

6.5.2 Tactics

This section is about tactics which satisfies the QAS derived from the QAW. A tactic is a design decision that influences the achievement of a quality attribute response. We will discuss our choosen tactics based on our QAS developed from the QAW. We will describe them in the same order in which they are arranged above. For each tactic there will be a description and an argument of the choosen tactic. Depending on the tactics, we will complement with the necessary views that affect the architecture including module-, component and connector- or allocation viewpoint.

6.5.2.1 Interoperability

This QA is related to QAS 1. Interoperability is about how systems meaningfully exchange information through interfaces in a given context. Since there will be different devices interacting with the bulletin board there is a demand on designing a standard interface which can serve these devices. *Discover Service* stands for the location of a data exchange service and that it is visible to those who need it. The service can be located by type of service, by name, by location or by some other attribute.

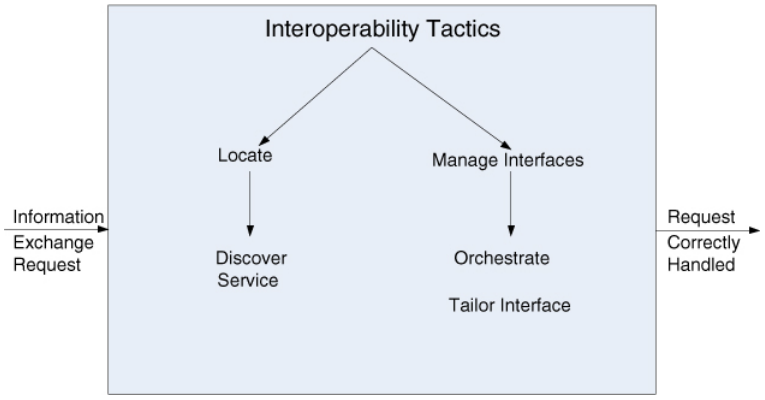


Figure 6.3: Interoperability tactics [BCK12]

With *Discover service* we will create a clear line between clients and the bulletin board. We will implement a REST API for this tactic.

REST - Representation State Transfer
The idea behind REST is that every resource has it's own URL (name) and you use the different HTTP methods to interact with those resources. REST consists of set of principles [FT02], but we will only use a subset of these principles as follows.

1. Transferring a representation of data in a format matching one of standard data types such as JSON, XML, HTML etc.

2. A resource is information that is identified by a URL provided by the server. A URL is Uniform Resource Locator which describe the address of a particular resource on the internet.

3. Interactions are stateless where each request contains all the information necessary. This is because of the scaling property.

The HTTP methods such as GET, POST, PUT and DELETE allow us to interact with the resources through an interface.

6.5.2.2 Security

This QA is related to QAS 2. Security is concerned with the ability to protect data and information from unauthorized access while still providing access to people/systems that are authorized. The PVSS protocol ensures that only valid votes are accepted and counted. This prevents that invalid votes are counted in the final counts and thereby they will not effect the result. The *Verify message integrity* tactic employs techniques such as checksums or hash values to verfy the integrity of a message. After the system has detected an attack the system must react on the attack.

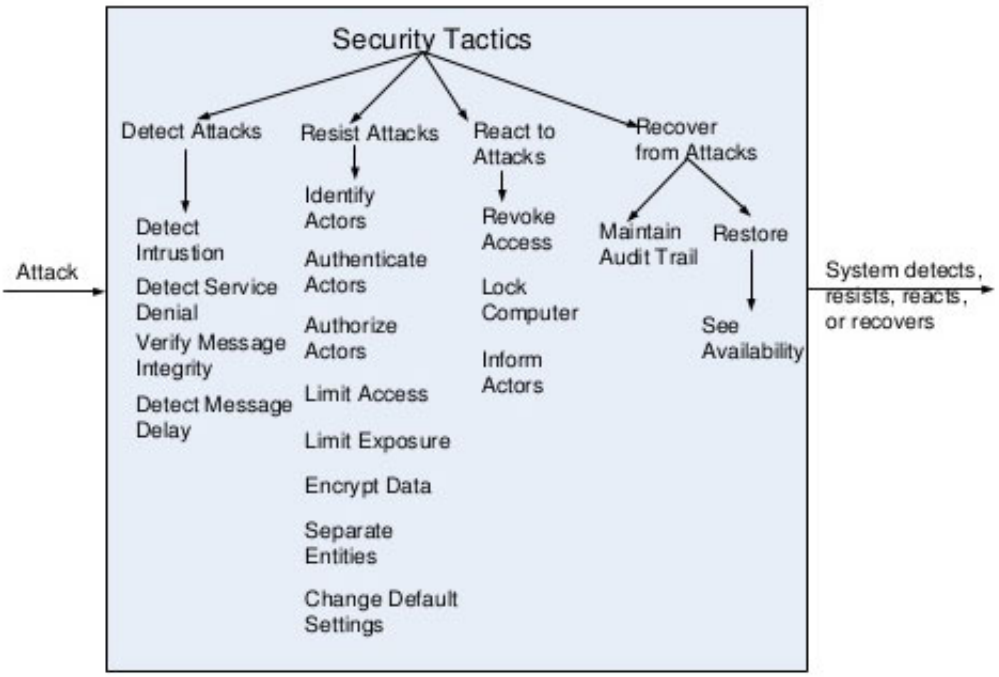


Figure 6.4: Security tactics [BCK12]

The $DLEQ$ and $PROOF_U$ proofs verifies the message integrity. With the proofs the protocol will be able to detect attacks. The verify message integrity consists of:

1. The $DLEQ$ proof provided in ballot casting process ensures consistency in the encryption process.

2. The $PROOF_U$ ensures that the voter votes either 0 or 1.

3. The *DLEQ* proof in the tallying process ensures that the decryption is done correct.

As already described this tactic is a part of the protocol which is described in section 5.3. We use the *Verify message integrity* tactic in relation with the proofs which ensures that the vote is either 0 or 1 and that the shares are constructed correctly and consistent. Our work consists of realizing the tactic in code. Another important decision to make, is if an attack is detected then the system, must react. The tactic *Inform actors* includes other systems to be notified. One way could be marking the vote in the database as not valid. The means that the vote is not counted in the tallying phase.

Component and Connector viewpoint
This viewpoint shows the runtime functionality regards how responsibility are flows in creating and verifying the proofs between the prover, bulletin board and the verifier.

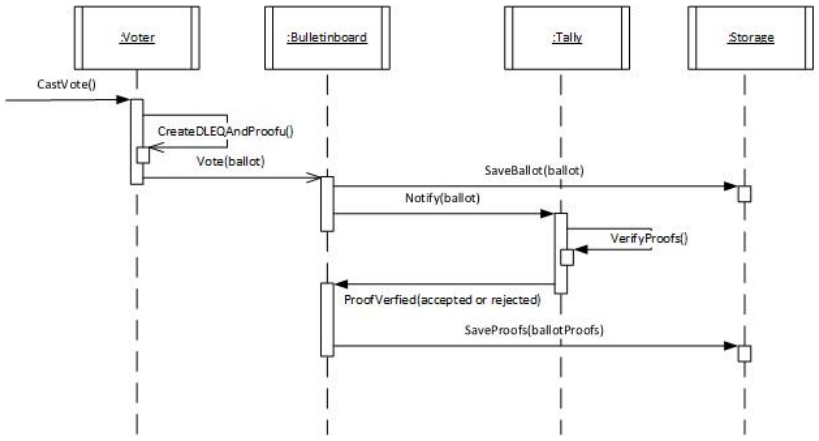


Figure 6.5: Sequence diagram which shows an overall flow of verifying the proofs

Step 1 The vote is casted and the voter creates the data for the proofs. The voter is an active process which the double bars on the box indicates. The voter is active process because it constantly awaits input from the user.

Step 2 The bulletin board recieves the ballot from the voter. It saves the ballot into a storage. It notifies the tally with the ballot. The bulletin board is an active process since it always listen on a specific port with purpose to serve incoming request.

Step 3 The tally verifies the proofs and revoke a method on the bulletin board with a parameter which indicate if the proof was accepted or rejected. For now we have choosen that the tally is an active process under the entire election.

Step 4 The bulletin board saves the ballot proof into a storage. We will emphasize that we always saves and never update. This constraint helps the system resist unauthorized users from updating any data in the storage. The storage is an active process because it constantly awaits incoming request.

6.5.2.3 Testability

This QA is related to QAS 3. Testability is concerned with the ease with which the software can be made to demonstrate its faults. This application contains a fair amount of computation which is the core of the application - namely to compute the final votes. Testing is needed, to ensure accuracy of the computations. When new features are introduced to the system, one should be able to execute all tests and results are captured and 85 % of the system are coverage within 3 hours.

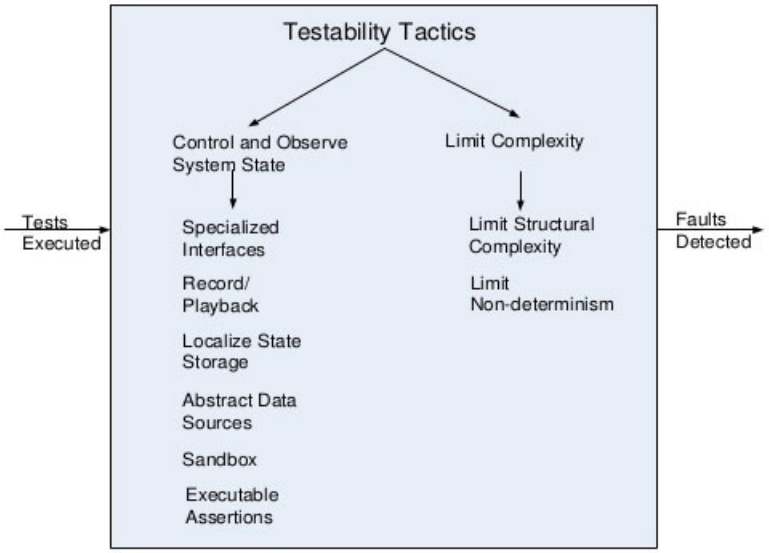


Figure 6.6: Testability tactics [BCK12]

The Voter client is build in Javascript which means that we have to adapt to the opportunities (dynamic, untyped, and interpreted run-time language) which the language provides. This tactic will be limited to webclient since they contain all computations. The tactic *Limit structural complexity* is about isolating, encapsulating dependencies and reduce dependencies between components. These principles leads to limited complexity and thereby better testability.

For a system to be testable we need to be able to control components inputs and be able to change its internal state and then observe its output. One of the ways for achieving this is through various design patterns such a strategy pattern. In general one should strive to use composite patterns that encapsulates responsibility which we use in section 6.5.2.5

Component and Connector viewpoint
Figure 6.7 shows the flow on the Voter client of the various functions which need to be executed in order to compute the ballot casting and proofs.

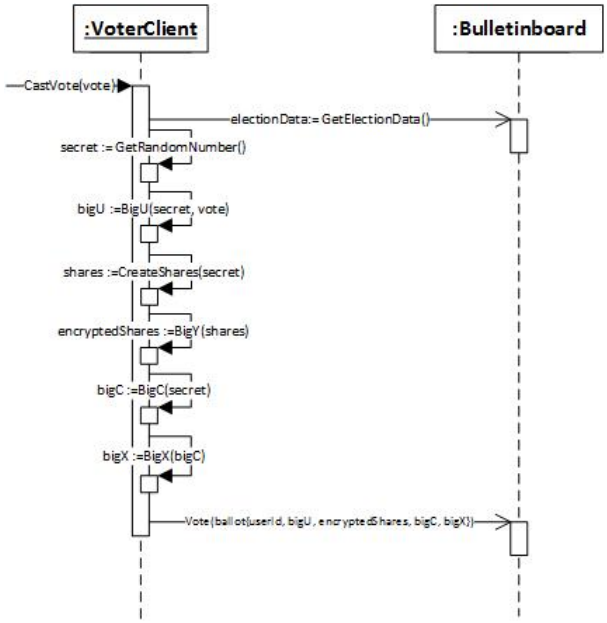


Figure 6.7: Voter client functions which needs to be testable

All functions ought to be relatively easy to test because they are fairly isolated. The communication with the bulletin board can be abstracted away in a testing environment through Test stubs [Chr10].

6.5.2.4 Modifiability

This QA is related to QAS 4. Modifiability is concerned with the ease with which the system supports change. In a reel electronic voting application scenario there will be need of a certain registration process. Depending on the scenario the registration process may go through state authorities, google, linkedin or even a third solution. Therefor the system must be able to support changes on the registration process depending on the use.

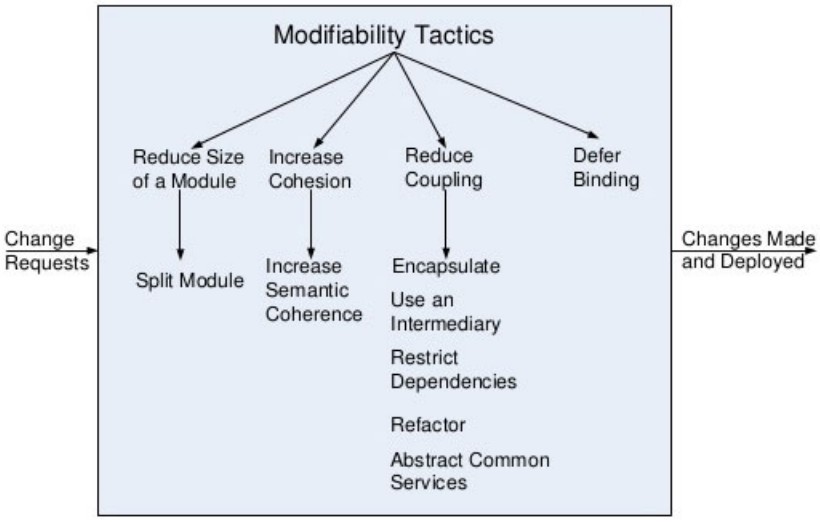


Figure 6.8: Modifiability tactics [BCK12]

Module viewpoint
Figure 6.9 shows the module viewpoint of a the strategy pattern on the process of registering a voter.

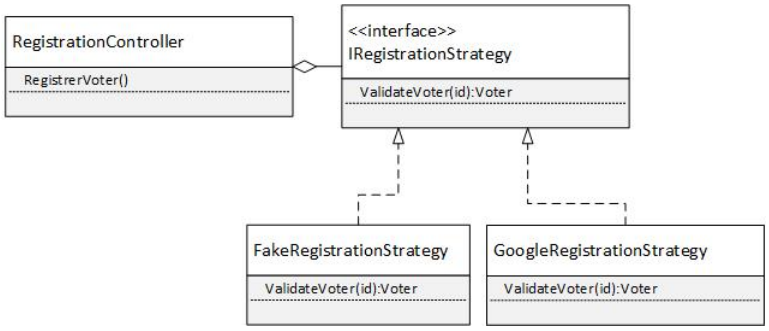


Figure 6.9: Module viewpoint of the strategy pattern on the registrations process

The RegistgrationsController gets an instance of the type IRegistrationStrategy. It can either be an instance of a GoogleRegistrationStrategy or a FakeRegistgrationStrategy. Each of these classes has their implementation of the method ValidateVoter. The GoogleRegistrationStrategy should implement a registration process to Google API. The FakeRegistgrationStrategy just contain "true" which means that the voter is a valid voter in the registration process. This strategy is only meant for testing purposes.

The process which led to this pattern is the compositional process which is as follows.

Identify the behavior that varies: In this case it is the registration process.

Use interface which encapsulate the variable behavior: In this case we create an interface IRegistrationStrategy which defines a responsibility of registering a voter.

Delegate the responsibility to a specialized classes: In this case we have FakeRegistgrationStrategy and a GoogleRegistrationStrategy.

By implementing a strategy pattern on the registration process, we have used the tactic *Encapsulate* and efficiently encapsulated this functionality and made the system ready to change to other registration authorities. Answering the question if we are able to reach the response measure from the QAS, depends on how complicated it is to integrate against an external registration system.

6.5.2.5 Modifiability

This QA is related to QAS 5. As described modifiability is concerned with the ease with which the system supports change. To further prove the security of the implementation we must take into account that the random generator easily can be changed. This is an advantage if we need to work with, for example, larger numbers in the future.

The tactic *encapsulate*, from figure 6.8, reduces the coupling between modules. The goal is to create an interface to the number generate so that we are able to shield of the concrete implementation of a given number generator and thereby reduce the coupling between modules. If we later then need to change the implementation we can create a new implementation of the number generator and replace the old one.

This tactic is limited to webclient since they contain all computations and the number generator. The following will be a proposal for an implementation of the concrete number generator and how one can encapsulate it with an interface.

Module viewpoint
This viewpoint shows the use of a strategy pattern as described in [Chr10]. The idea with the strategy pattern is to define a family of business rule, encapsulate each one and make them interchangeable. The strategy pattern lets the business rules vary independently from clients that use it. It delegates the responsibility to an object instead of doing the work it self.

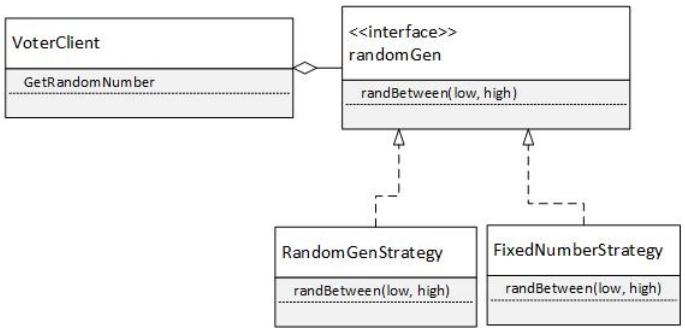


Figure 6.10: Module viewpoint of the strategy pattern on the random number generator

Figure 6.10 shows the VoterClient gets an instance of a randomGen which has a method randBetween. It can either be an instance of a RandomGenStrategy or a FixedNumberStrategy. Each of these classes has their implementation of the method randBetween. The RandomGenStrategy implements the reel random number generator. The FixedNumberStrategy returns a controlled value which we can set, for testing purposes.

The process which leds to this pattern is the compositional process which is as follows.

Identify the behavior that varies: In this case it is the random number generator.

Use interface which encapsulate the variable behavior: In this case we create an interface randomGen which defines a responsibility of returning a random number.

Delegate the responsibility to a specialized classes: In this case we have RandomGenStrategy and a FixedNumberStrategy.

By implementing a strategy pattern it should be relative easy to switch out the random number generator and thereby making changes to this part of the system.

Chapter 7

The application

In chapter 6 we described the theoretical aspect behind our system. A software architecture was designed and the most significant scenarios was formalized and solved on a theoretical level. In this chapter we describe how we implemented the design into a application.

7.1 Introduction

Implementing a software design can be a difficult task, as design modules rarely maps directly into code. Using the approach of different viewpoints and scenarios helps in this regard, but challenges due arise which could not have been foresee nor illustrate in the design phase. Challenges can come from our design of the development environment to the physical restricts of the deployment environment.

7.2 Method

The final result is described in a top-down approach. We start with an informal description of the system, and from here gradually go into details starting with the module views which describes the static structures of the implementation. Then, follows the component and connector view to describe the dynamic structures. After the presentation of the implementation we will look at how choosen tactics have been implemented and how we solved the functional requirements of an electronic election.

7.3 Development environment

The implementation have been done using Microsoft ASP.NET framework, C# and Javascript programming language. ASP.NET is a widely used and supported framework for creating web application. This framework is used both as the backend RESTful server, bulletin board and for the web server hosting the clients. The clients, voter and tallier is implemented using Javascript. Javascript have the benefits of running inside all modern browser and is widely

used, thus giving us a very large set of libraries to our disposal.

7.4 Final result

In this section we describe the final implementation of the software design.

7.4.1 Overview

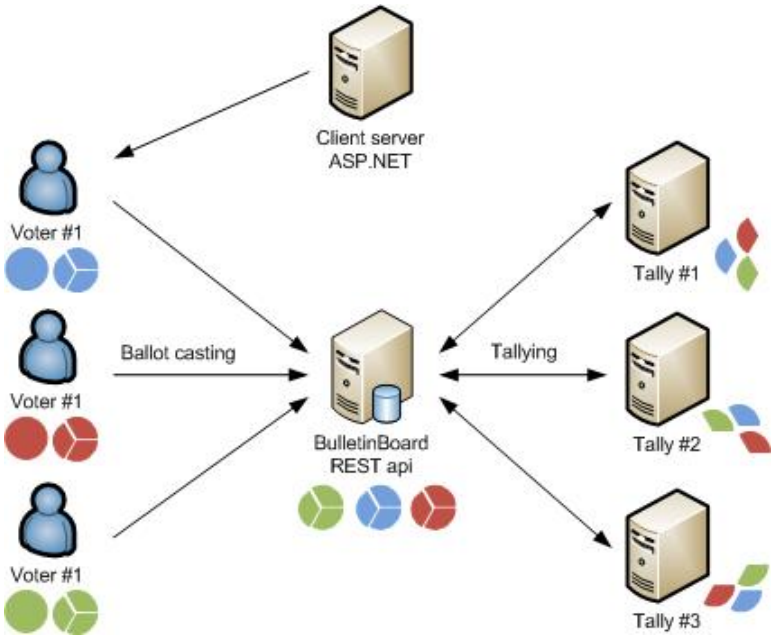


Figure 7.1: Overview of the system

Figure 7.1 illustrates an overview of the system and the information flow in the different phases of the protocol. From left to right we see the voters each casting there votes and hiding their votes within a secret. Though secret sharing the vote is divided and encrypted into pieces corresponding to the number of talliers. The ballot casting phase ends with the voters publishing there votes and secret to the bulletin board. Ending the ballot casting phase starts the tallying phase where the talliers each collects there individual pieces of the secret, multiplying the shares together, decrypting the multiplum and publishes the decrypted multiplum to the bulletin board. The last segment of the tally phase, where the votes are calculated is not illustrated in the figure, but is none the less implemented by letting a dedicated tally do the calculating and posting the result to the bulletin board.

The overview does not introduce many new elements to the implementation not previously known from the protocol, in fact the only new element introduced is the Client server which serves as a web-server for the voters, to which the voters logon in order to cast there votes. However the overview clearly shows the

topology of the implementation, namely a star topology with a server (bulletin board) in the middle and Clients all around. There is no directly communication between the clients, all communication goes through the Server. This is due to the nature of an election, though we require the talliers to have a persistent connection to the server. We don't expect voters to stay connected throughout an entire election. The reason the bulletin board needs to have an persistent connection to the talliers is because after the ballot casting phase. The bulletin board needs to be able to notify the talliers for tallying phase.

7.4.2 Viewpoints

We use viewpoints as we further describes the implementation in details. We start with the module viewpoints which shows the static elements of the implementations, elements like packages, interfaces, classes and relations.

7.4.2.1 Module view

Starting from the top we look at how the implementation is structured into packages, where each packages contains elements that focuses on the same responsibility. Structuring our implementation this way follows the design principles of high cohesion and low coupling introduced in chapter 6 and lays the ground for code that is flexible and easily maintainable.

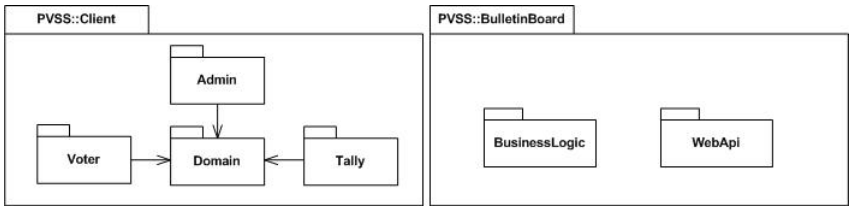


Figure 7.2: Package overview

Figure 7.2 shows a package overview of the entire implementation. The figure show both the packages included in the implementation of the Client and the bulletin board. The Client include packages containing logic specific for a Voter, Tally, Admin and the Domain package which holds logic used across the Client. The package overview of the bulletin board with only the two packages BusinessLogic and WebApi looks on the surface very simply, but below we unfold each package showing a more complex structure.

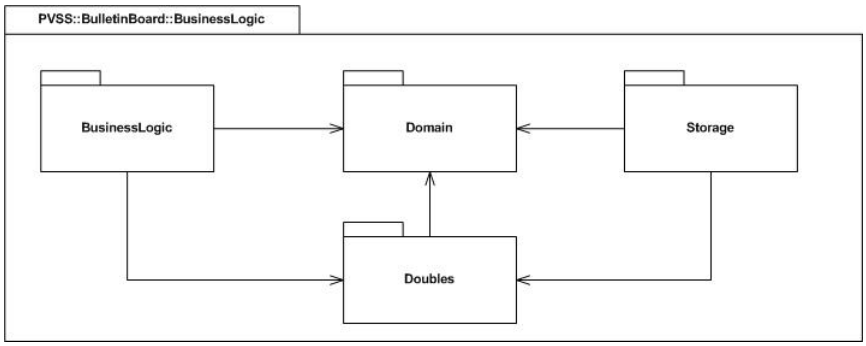


Figure 7.3: Package overview of bulletin board business logic

Figure 7.3 shows the packages revealed when unfolding the BusinessLogic package. This include the sub-packages BusinessLogic, Storage, Doubles and Domain. The packages BusinessLogic, Storage and Domain is fairly self explanatory. The package Double holds the logic we used in order to unit test our implementation.

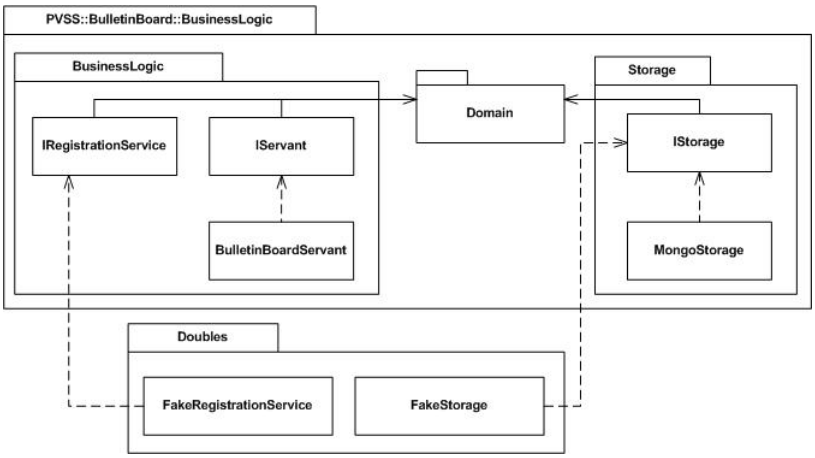


Figure 7.4: Module view of bulletin board

Figure 7.4 shows the interfaces and classes within the BusinessLogic package of the bulletin board. Note that the package Doubles have been left outside the container of the BusinessLogic, this is due to the fact that its responsibilities only evolves around testing and the packages is not used in the production environment. Also noticeable is the Domain package have not been specified further. The reason for this is, that it simply holds the representation of physical elements like a ballot, a voter, a share etc, which contain no business logic.

IRegistrationService is an interface that holds the responsibility of the *Registration authority* specified in section 2.3. Is validates and registries eligible voters. For this implementation we only included the class **FakeRegistrationService** which implements responsibilities from the IRegistrationService.

IServant is an interface that have the responsibilities of the *bulletin board* also the described in section 2.3. It handles the communication with the Clients and insures the persistence of the election data such as the ballots and shares etc. The class **BulletinBoardServant** implements the responsibilities of the IServant. A key functionality of the BulletinBoardServant is that most of its methods is read-only, this helps to ensure that no honest or dishonest user removes information.

IStorage is an interface that have the responsibilities of persisting and retrieving data. Both the classes **MongoStorage** and **FakeStorage** implements the responsibilities of the IStorage. The **MongoStorage** also acts as a client to a Mongo database to which it stores the data, where the **FakeStorage** simply stores the data in memory, meaning that the data is lost when the application stops.

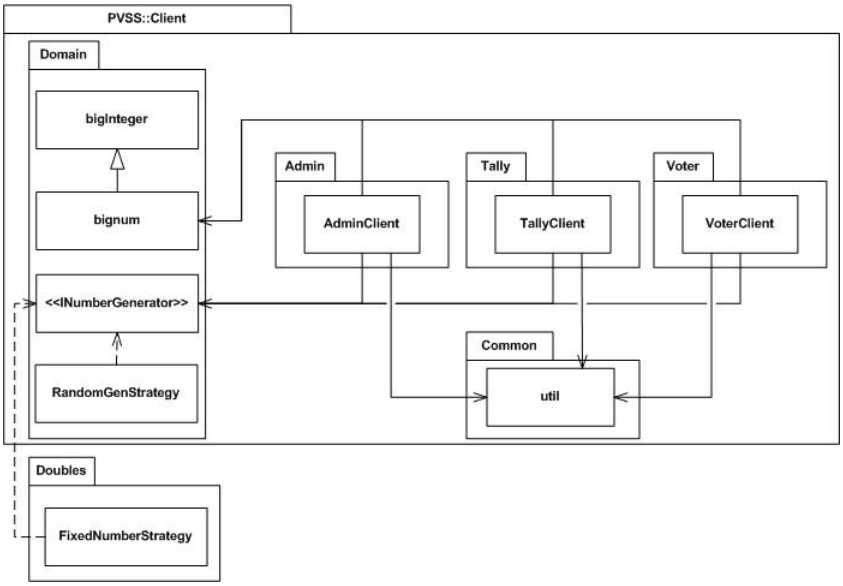


Figure 7.5: Module view of Client

Figure 7.5 shows the interfaces and classes of the Client. Again the package Double is left outside the content of the Client, due to the same arguments stated above.

util is a simple class that holds some helper function, that is used across the system.

bignum is a class that inherits the functionlitites from its superclass **bigInteger**. This structure enables us to expand or alter the functionalities of the **bigInteger** without changing the class.

INumberGenerator is an interface which have the responsibility of generating random integers and primes. The classes **RandomGenStrategy** and

FixedNumberStrategy both implements the responsibilities of INumberGenerator. **FixedNumberGenerator** is used for testing and enabling us to control the numbers return.

AdminClient is a class that holds the responsibilities of creating an election and generating the public elements used in the election. The AdminClient is depending on the class Bignum, and the Interface INumberGenerator.

VoterClient is a class that holds the responsibilities of a *Voter* described in the protocol, section 5.1.2. This class, like the AdminClient is depending on Bignum and INumberGenerator.

TallyClient is a class that holds the responsibilities of a *Tally* described in the protocol, section 5.1.3. This class, is also depending on the Bignum and INumberGenerator.

This last module view concludes the overall static structure of the implementation, we however have not further specified the elements in the Webapi package shown in figure 7.2. The reason for this, is that it does not hold any elements related to the protocol but only elements regarding the implementation of the REST interface describe in the Interoperability tactic in section 6.5.2.1 and this implementation is discussed later.

7.4.2.2 Component and Connector view

For this next part we will look at how the dynamic components in the implementation interacts during runtime. We will present an overall insight of the system's interaction.

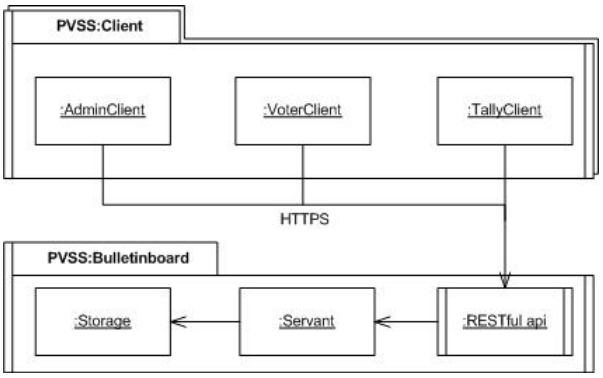


Figure 7.6: Overall Component and Connector view

Figure 7.6 shows how the system interact overall. As we have seen in the module views, there is no interacting between the components in the Client. They each operates independently and they all communicates with the bulletin board through the REST API. The communication between the Clients and the bulletin board is done through an HTTPS connection. This connection uses Secure Socket Layer (SSL) or Transport Layer Security (TLS) to encrypt the messages sent or received. All successful communication done with the REST API is

forward to the Servant component that handles the business logic. If there is a need for persisting data, then the Servant sends the data to the Storage.

The following sequence diagram shows the overall flow of the system from the ballot casting phase to the result. The internal method calls and calculations done on each individual class is not show, in order to present a better overview.

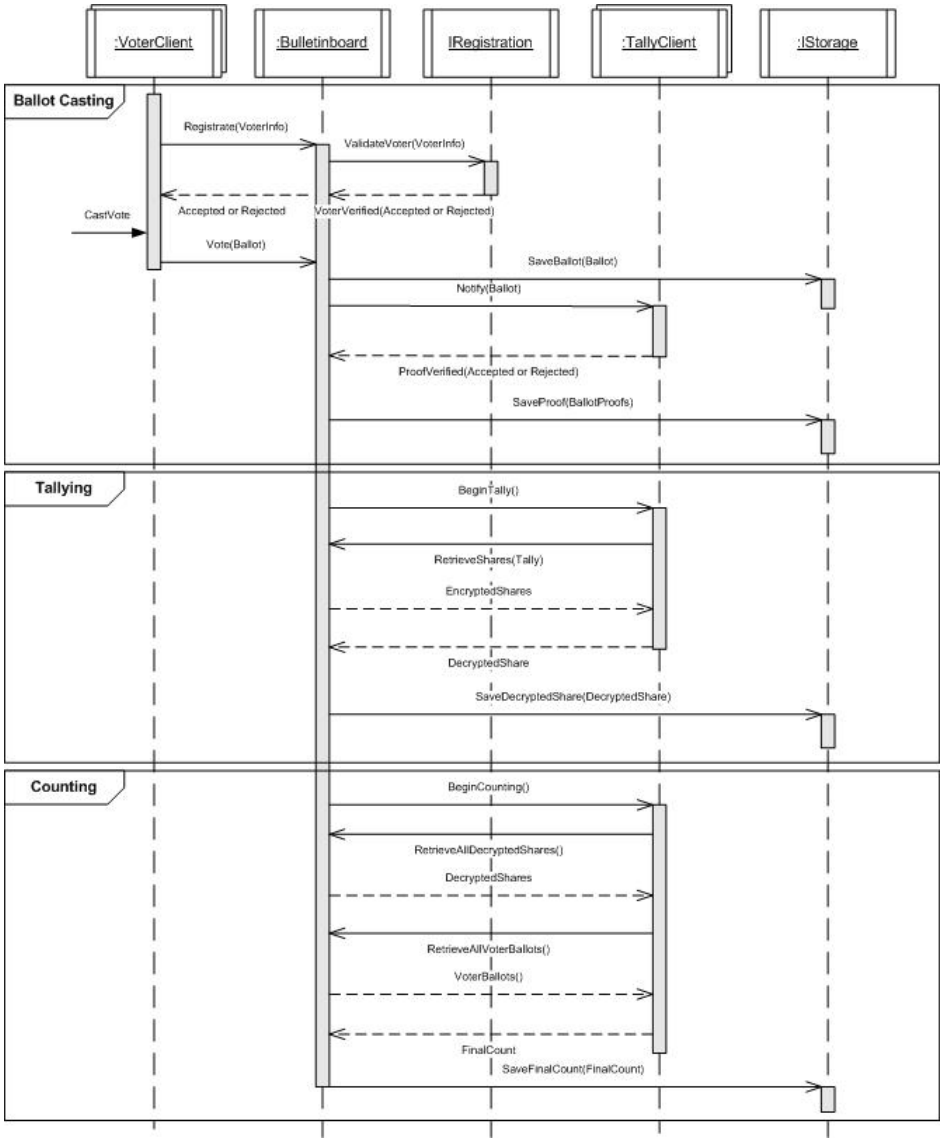


Figure 7.7: Overall Sequence diagram

7.4.2.3 Allocation view

Finishing out the viewpoints we lastly have a deployment view showing the systems requirements to the deployment environment.

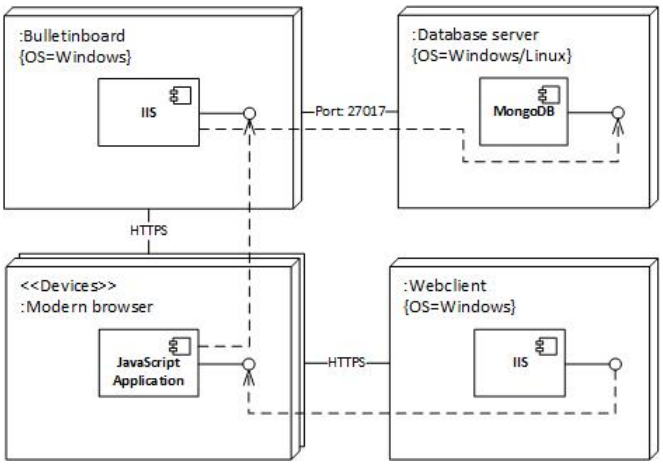


Figure 7.8: Final deployment view

Figure 7.8 shows that the system requires three servers, a database server running a mongo database, a web-server with a Internet Information Server (IIS) installed to host the bulletin board RESTful Webapi and lastly a web-server also with IIS installed to host the Webclient. We also require that devices that is to communicate with the system needs to have a modern browser installed, that is able to execute Javascripts.

7.5 Implementing the tactics

In this section we will highlight selected implementation of previously described tactics from chapter 6.

7.5.1 Interoperability: REST API

As stated earlier we use a REST interface which should be easily accessible by everyone. This means that our bulletin board is build on some of the REST principles. Figure 7.9 illustrate an informal webservice model which describes our resources on the bulletin board. The process of constructing this diagram gives overview and insight of how we could design a REST interface on the different resources.

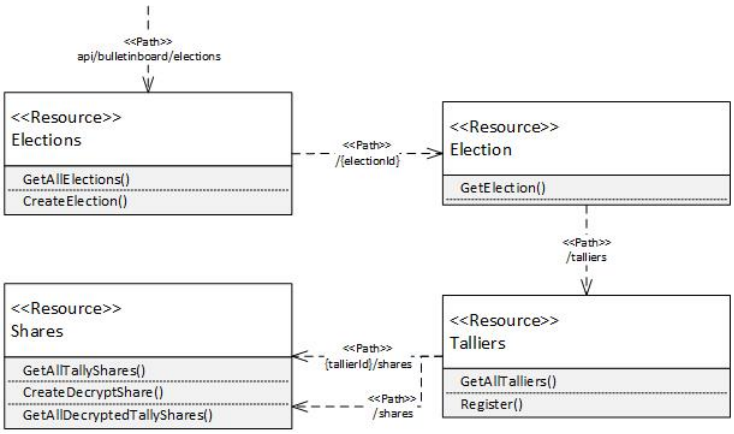


Figure 7.9: Overview of REST electronic voting web service model

Figure 7.1 shows our final REST api. The urls shows how the REST api can be accessed together with the HTTP methods which shows which methods can be invoked on the different urls.

Resource	URL	HTTP methods
Elections	/elections	GET GetAllElections POST CreateElection
Election	/elections/{electionId}	GET GetElection
Votes	/elections/{id}/votes	GET GetAllVotes POST CreateVote
Talliers	/elections/{id}/talliers	GET GetAllTalliers POST Register
Shares	/elections/{id}/talliers/{id}/shares	GET GetAllTallyShares POST CreateDecryptShare
Decrypted shares	/elections/{id}/talliers/shares	GET GetAllDecrypted TallyShares

Table 7.1: REST api to the bulletin board

Since our webclients is build purely on Javascript, all communication is done through ajax call to the bulletin board. Listings 7.1 illustrates how one can interact with REST api. The example shows how to create an election. This is called by the Admin client which starts the election.

```
1    $.ajax({
2        type: "POST",
3        url: "api/bulletinboard/elections",
4        data: jsonData,
5        success: function (data) {
6            if (callBack)
7                callBack(data);
8        },
9        contentType: "application/json"
10   });
```

Listing 7.1: Javascript example

7.5.2 Modifiability: Number generator

Implementing this tactic proved to be fairly easy. However as the dynamic nature of Javascript does not support the concept of interfaces as we know it from Java and C#, we had to introduce a method to secure that the injected NumberGenerator strategy applies to its responsibilities. To construct this method we utilizes the concept of *Duck Typing* which basically states that if an object walks like a duck and quarks like a duck, then to the concerns of Javascript it is a duck!.

```
1     // example duck typing method
2     var InterfaceMethods = function (obj /*, method list as strings
       */) {
3         var i = 1, methodName;
4         while ((methodName = arguments[i++])) {
5             if (typeof obj[methodName] != 'function') {
6                 return false;
7             }
8         }
9         return true;
10    }
```

Listing 7.2: Implementation of Duck typing

In the following listing 7.3 we show how the InterfaceMethod method is used to check for an interface object. It checks if the methods specified in the parameters is to be found on the object passed as the first parameter. In the example below it checks for the methods "randBetween" and "randPrimeBetween".

```
1  var voterClient = function (serv, options) {
2      ...
3      ...
4      var randomGen;
5
6      if (options) {
7          if (options.randomGen) {
8              if (util.InterfaceMethods(options.randomGen, '
       randBetween', 'randPrimeBetween')) {
9                  randomGen = options.randomGen;
10             }
11         }
12     }
```

Listing 7.3: Checking for an interface

As shown in the listing 7.3 the NumberGenerator strategy is simply injected into the VoterClient upon creation. Methods within the VoterClient can now execute the methods randBetween and randPrimeBetween on the injected object through the variable randomGen.

7.6 Analyzing the application

In this section we will list the security requirements from section 2.5. We will then evaluate these requirements against our application.

7.6.1 Electronic voting secure requirements

Voter Privacy *No one should be able to link a vote back to the specific voter, and only the voter should know his vote. These requirements shall hold*

during and after the election.

This requirement is meet by our implementation, infact its one of the core elements in the election voting protocol from section 5.1. Every voter only publish his vote though $U = G^{v+s}$ where G is a generator and $v \in \{0, 1\}$ and $s \in \mathbb{Z}_q$. As s is only known by the voter and is uniformly random picked then the sum of s and v is also unknown for any potential adversary under the security of the DL problem. And one could ague that even if an adversary should be break the DL problem then, unless he knows s, the vote v would still be unknown.

Eligibility *Only Eligible and registered voters can vote.*

Unlike Voter Privacy, this requirement of non-eligible voters not being able to vote is not fulfill by the protocol it self. First we need a list of eligible voters which acts as a reference list for which voter are allowed to vote. Second our registration service handles registration which is described in section 6.5.2.4. This enable us to change the registration service accordingly to the nature of the election.

Uniqueness *Only one vote per registered voter should be counted.*

Uniqueness is about making sure only one vote is counted for each eligible voter. This requirement is meet by having a separate table in the database with all eligible voters containing an id and a 0/1 (no or yes). This table will be updated when votes arrive to the bulletin board.

Fairness *None should be able to gain any knowledge of the outcome of the election, before the ending. This is to prevent voters of voting accordingly to any leaked information.*

This is achieved in our implementation by only starting the tally phase when all votes are casted or when a deadline is reached and the ballot casting phase ends. Only the bulletin board have the authority to notify the talliers when to begin the tallying phase.

Through the property of secret sharing used in the protocol, the secret to decrypt the vote is shared amoungst three or more talliers. These shares are again encrypted with the public key of the corresponding tally. As a consequence, no one except the tally with the corresponding private key is able to decrypt the share.

Uncoercibility *Nobody should be able to extract the value of a vote. This is to prevent anybody from compelling a voter by force, intimidation, or authority to cast a vote in a specific way.*

Since our application is classified as a *remote internet voting* described in section 2.2, we are not able to control the physical environments to where the votes are casted. Therefor we are not able to fulfill this requirement to its fullest. However the protocol ensure that no coercer is able to extract a specific value of a vote, due to properties already mentioned in the previous requirements.

Receipt-freeness *The voting system should not produce a receipt that reveals any information about the casted vote. This is to prevent a voter from trading his vote.*

Our application only confirms the success of a voting, not the value of the vote. There is no functionality that enables any participants nor observers to gain information about the value of a vote.

Accuracy *The final tally should be correctly computed from valid casted votes. It should not be possible to manipulate the final tally without being detected.*

Our implementation utilizes the proofs from the protocol to fulfill this requirement.

Under the ballot casting phase of the protocol, we require that the voters proofs the correctness of there votes. This is done through the $Proof_U$ and $DLEQ$ proofs, which is required to be published along side the encrypted vote U. Should one of the proofs fail, then the vote is marked invalid and this vote is ignore in the preceding processes.

In the tally phase of the protocol, the tally multiplies its shares and then decrypts them and publishes the end result. Along with this result we require that the $DLEQ$ proof is published aswell. This allows us to verify the correctness of the tallying process for each tally. Should a $DLEQ$ proof from a tally fail then the tally's shares is ignored in the preceding process. The protocol requires the shares from at least t talliers, in order to extract and calculate the end result. Should it be the case that this requirement is not fulfill an re-election is require but untill then the implementation simply ignores the shares from the tally in question.

Should an adversary gain access to the database serving the bulletin board, it would be possible for this adversary to manipulate the verdict of a proof but not the proofs them self, as the construction of the proofs prevent this. Though our application does not take this scenario into account it would be fairly easy to make a functionality that reevaluates the proofs, should such a breach have been detected. One could argue that an adversary with full access to the database could also remove elements such as a casted vote or the information that a given voter have voted. The later would effectually enable the given voter the ability to double vote. This scenario is not handled in our application. The first scenario is also possible but given the fact that our application have voter privacy and the votes is secure under the DL problem then the adversary would not know if he is removing and "no" or a "yes" vote.

Universal Verifiability *It should be possible for any participants and observers to validate individual votes as well as the final tally of the election.*

The protocol is basically designed around this requirement as voters publishes their ballot and proofs to the bulletin board. The Tally will also publish a proof under the tallying phase. Any participant or observers can validate the proofs since it is public.

Everything on the bulletin board is publicly available both for participants and none participants of the election. $Proof_U$ verifies that an encrypted vote U is either 0 or 1 and the $DLEQ$ proofs verifies the consistence of the shares for both the voters and the talliers. The end result can be calculated from the publicly known information available after the tallying phase.

Individual Verifiability *Every registered voter should be able to verify that his vote is counted correctly.*

This requirement is contradicting with the requirement of receipt-freeness. The challenge is how can one voter verify that his vote is calculated and included in the final tally correctly. This can not be done directly - however we state with the following two new informal requirements, every votes is calculated and included in the final tally and as such his vote is calculated and included correctly.

1. All votes from the ballot casting is included in the final tally.
2. Every vote from the ballot casting is calculated correctly.

With the protocol we can fulfill these two requirements. However we have not implemented the necessary functionality in the application. We will elaborate on these two statements under the discussion section 8.2.2.

Part IV

Reflecting

Chapter 8

Discussion

In this chapter we will discuss and reflect our challenges regards to our work with this thesis objectives.

8.1 Theoretically

The learning curve has been steep regards to learning the electronic voting protocol. The literature is on a very high mathematical academic level seen from the perspective of a software developer. We see this is as a challenge that a protocol is only described for such a narrow audience, especially one that requires knowledge of such a specific field as cryptography. Most of the concepts used to describe the protocol assumes a certain background knowledge. When reading the article on protocol there is often reference to other literature that the protocol is build upon. This makes the article hard to read, as the reading flow is interrupted.

With this thesis we have tried to break down the protocol such that it is understandable for a broader audience. We have constructed this thesis such that the reader gradually gains the required mathematical knowledge to understand the basic elements of the protocol. The description of the protocol has been divided into a basic and a detailed description. The basic description will give the reader basic knowledge about the protocol and should provide enough knowledge for a simple implementation of the protocol. Of cause this simple implementation will not fulfill all the security requirements for an electronic voting application.

8.1.1 Knowledge accumulation

Starting this thesis we had the subjects of multiparty computation and secure secret sharing in mind, but after guidance with our supervisor we agreed on working with the electronic voting protocol described in [Sch99]. In order to connect the theoretically part of the thesis with the practical part, we felt that additionally knowledge about electronic voting in generally were needed. After reading other articles on the subject we got a better overview on the concepts of electronic voting. It turns out that the security requirements for electronic voting is one of the key elements linking the theoretical part together with the

practical part. This research helped us being able to include different issues, that we by our self would not consider immediately, such as our consideration regarding *Eligibility* and our reflection on the *Individual verifiability*. In general our research on the subject have given us a more critical approach to which security parameter we will have to take into consideration. For example we have given a short description of different known attacks on the discrete logarithm problem. Knowing these different attacks gives knowledge for how large q must be, which is essential for our practical part.

8.2 Practical

We have spend much time on understanding the protocol and the security requirements regarding electronic voting. This has been costly for the practical part of this thesis. We have not been able to develop a complete application, but rather a proof of concept, where we have tested the protocol and our architectural strategies. We would like to have spend more time on the practical part so that we could have tested our QAS such as performance and the availability. Since these two attribute is central for a software architecture on a large scaled application.

Basically the entire protocol is about doing computations such as exponentiation, multiplying, addition, modulo reduction, hash functions and generating random large primes etc. Doing it and doing it right is a challenge - all these computation together makes the system complex and challenging to debug. We created several proof of concept with small numbers, which was manageable to follow. We haven't found a optimal way yet for debugging the computations with large numbers.

As developers we are concerned with achieving maintainability and readability of the code, therefor code quality is an important aspect to take into account. The protocol works with variables like g, G, Y, Y^*, $DLEQ$, and $Proof_u$ etc. We ask our self what is the best naming conventions for these variables. Our naming conventions is based on trying to be consistent and give variables signing names.

8.2.1 Generating Prime

Generating very large primes is a computationally hard task and thus time consuming. Given the fact that the prime q used in our implementation is public known, it is easy to think that one could simply use one or a set of very large hardcoded primes. However, by performing a large precomputation for a given prime, an adversary can quickly calculate arbitrary discrete logarithms in that group. And efficiently reducing the computation cost for all targets that uses this group [ABD+15].

8.2.2 Individual Verifiability

Here we will elaborate on these two informal requirements from section 7.6.1 regarding *Individual Verifiability*.

1. All votes from the ballot casting is included in the final tally.

2. Every votes from the ballot casting is calculated correctly.

As stated earlier our job is to convince the voter about the above statements. If this is possible, we mean that we have argumented for our statements and thereby the security requirement. To elaborate these requirements we will need some illustration of the ballots and the tallying phase. Figure 8.1 shows a simplified list of all valid ballots from the ballot casting phase. Figure 8.1 also shows a simplified list of the multiplied shares and the corresponding proofs by each tally.

As for the first statement we refer that anyone can verify the correctness of the votes and the consistency of each share. By definition all is public and anyone are able to verify the validity of the outcome of the proofs under the election.

So if the list is trusted and accepted by everyone then with high probability we can say it is correct. To ensure that all ballots are included in the final tally we have to be convinced, that each of the shares are included in the tallying phase. Each tally will compute Y^*, S^* and a proof based on the shares belonging to them. The key is that anyone are able to confirm the Y^* to ensure that all shares are included in the computation of Y^*. As for the second statement all can verify that the decryption of the shares are done correctly. All information needed in order for calculating the final tally is publicly available on the bulletin board and thus anyone are able to recalculate the final tally.

Ballot casting	
Ballot	Valid
1	Yes
2	Yes
3	Yes
4	Yes
5	Yes
6	Yes

Ballots with proofs validation

Tallying phase			
Tally	Y^*	S^*	Valid
1	10	3	Yes
2	8	9	Yes
3	5	8	Yes
4	9	4	Yes
5	9	3	Yes
6	2	8	Yes

Tallying phase

Figure 8.1: Tables showing the states of a Ballots and Tally shares after each phase in a election

8.3 Lesson learned

Looking back, we should have been better at narrowing the scope of the theoretical part of this thesis. However this is a hard task with our experience within the field of cryptography. With our programming experience and our knowledge in the field of software we can certainly say it has not been a trivial task to implement and at the same time getting the required knowledge of the different parts of the protocol. As a consequence we had to prioritize the project using the elements of "The Project Manager's Three-Legged Stool". Here we

have three elements time, cost and quality which are parameters in a project. For this thesis time and cost are relatively fixed, which leads to one parameter to adjust namely quality. As stated earlier there is still work for us to do, on the practical part of our application.

Our choice of development environment has been Visual Studio with ASP.NET, C# and Javascript as main programming languages. Based on our knowledge that these technologies is widely used within our profession, the choice was clear for us. We have encountered challenges in our process to gather knowledge about certain technical aspects, revolving different cryptographic libraries in our development environment. The documentation for these libraries covering our development environments have been poor. A consequence of this have been that we have use more time then planned, on our application. We have read a lot of forums and documentation using other technologies on this subject. An recommendation will therefor be that one should be careful when choosing the tools to implement an application in this field.

Chapter 9

Conclusion

In this thesis three objectives were studied: The theory behind Shamir secret sharing and multiparty computation as well as the cryptographical concepts needed to understand it. The electronic voting protocol and the mathematical justification behind it. Design and implement an secure and scalable web based electronic voting application based on the protocol. We will in the following summarize our main achievements.

By combining the theoretical knowledge gained through session with our supervisor and through studying the field of cryptography, with applied practical experience gained by implementing the protocol in a proof-of-concept application. We have accomplish, most of the objectives in this thesis.

1. We have gained comprehensive knowledge in regards to Sharmirs secret sharing and multiparty computation as well as the concepts surrounding them. Furthermore we feel that we have managed, through elaboration and the use of examples, to describe these concepts in such a way that peers with a similar backgrounds as us, as a software developers, can learn the concepts with a lesser steep learning curve.

2. We have taken the description of the PVSS protocol and the electronic voting protocol described in [Sch99] merging them together, for then to divide the description into three different parts each elaborating the protocol further. We see this structure as an optimal way for learning the protocol while trying the concepts in practice, as it allows for iterative implementing the protocol.

Gaining knowledge about the theoretical concepts behind the protocol, have helped us understand how to implement the protocol. Even though we have practical programming experience, we none-the-less faced several situations where the theory helped us clarify how to solve the situation.

1. While implementing the electronic voting application we had the chance to combine the theoretical knowledge with our programming experience. This has been a huge advantage in our learning process. One thing is to understand the mathematics behind the protocol. Another thing is knowing what is needed in order to implement the protocol. We have found

it beneficial to gain comprehensive knowledge of the mathematics and cryptography concepts behind, not only the protocol but also electronic voting in general. This unfortunately have been more time consuming then expected, which in the end had the consequence that we have not yet reach our objective with our application. Nevertheless we got insight into designing and implementing a cryptographic protocol.

2. Our design of the electronic voting application have been heavily influenced by the security requirements for electronic voting. We see these requirements as an essential part of developing an electronic voting application. While most of the requirements is taken into account by the protocol it self, there where some requirements that we had to incorporate into the application our-self. Even though our web based electronic voting application is not finished, we have prepared an architecture which is build upon known design principles from [BCK12] and [Chr10] such as Quality attributes (QA), Quality attribute scenarios, tactics and design patterns. Furthermore we would like to highlight these QA, *Interoperability*, *Modifiability*, *Security* and *Testability* which the architecture is build upon.

Bibliography

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 5–17, New York, NY, USA, 2015. ACM.

[BCK12] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice, 3rd ed.* Addision-Wesley, 2012.

[BEL+03] Mario Barbacci, Robert Ellison, Anthony Lattanze, Judith Stafford, Charles Weinstock, and William Wood. Quality attribute workshops (qaws). Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

[CCH16] Henrik Bærbak Christensen, Aino Vonge Corry, and Klaus Marius Hansen. The 3+1 an approach to software architecture description using uml: Revision 2.4. Workingpaper, Århus Universitetsforlag, May 2016.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, pages 174–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[Cet08] O. Cetinkaya. Analysis of security requirements for cryptography voting protocols (extended abstract). In *2008 Third International Conference on Availability, Reliability and Security*, pages 1451–1456, March 2008.

[Cet09] O. Cetinkaya. Cryptography in electronic voting systems. In *International Conference on eGovernment and eGovernance*, pages 297–310, March 2009.

[Chr10] Henrik Bærbak Christensen. *Flexible, reliable software: Using Patterns and Agile Development.* Chapman & Hall, 2010.

[DGS02] Ivan Damgård, Jens Groth, and Gorm Salomonsen. *The Theory and Implementation of an Electronic Voting System*, pages 70–100. Kluwer Academic Publishers, 2002.

[FT02] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman & Hall, 2014.

[PPP09] Christof Paar, Bart Preneel, and Jan Pelzl. *Understanding Cryptography*. Springer Science and Business Media, 2009.

[Sch99] Berry Schoenmakers. *A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting*, pages 148–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[Yao82] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, November 1982.

Appendix A

Proofs

A.1 $DLEQ$ non-interactive proof between voters and verifier

In section 5.3.1 we elaborated an interactive $DLEQ$ proof between voter and verifer. Here we present how one can turn an interactive proof to a non interactive proof. This is also known as the Fiat Shamir where we are transforming an interactive proof into a non interactive proof which is described in section 3.4.5. Instead of the verifier computes a challenge, the prover computes the challenge as a random function as described in 3.4.4. We will present two ways off doing this transformation, a non optimized and an optimized version. Last we will describe an example on how this can be done.

The prover computes $a_1 = g^w \ (mod\ q)$ *and* $a_2 = y_i^w \ (mod\ q), w \in_R \mathbb{Z}_q$. Then the prover computes the hash $C = H(X_i, Y_i, a_1, a_2)$. Then the prover computes $r = w - p(i) \cdot C \ (mod\ q)$. Last the prover publish a_1, a_2, r, C.

The verifier computes the following computations $a_1 = g^r \cdot X_i^C \ (mod\ q)$ and $a_2 = y_i^r \cdot Y_i^C \ (mod\ q)$ and $C = H(X_i, Y_i, a_1, a_2)$.

DLEQ protocol			
$Input: g, X_i, y_i, Y_i$ *where* $X_i = g^{x_i}$ *and* $Y_i = y_i^{x_i}$			
	Prover		Verifier
Step 1	$w \in_R \mathbb{Z}_q$ $a_1 = g^w \ (mod\ q)$ $a_2 = y_i^w \ (mod\ q)$ $C = H(X_i, Y_i, a_1, a_2)$ $r = w - p(i) \cdot C \ (mod\ q)$		
Step 2		$\xrightarrow{a_1, a_2, r, C}$	*checks if :* $a_1 = g^r \cdot X_i^C$ $a_2 = y_i^r \cdot Y_i^C$ $C = H(X_i, Y_i, a_1, a_2)$

Figure A.1: *DLEQ* non interactive

Note in above that there is no interaction between the prover and the verifier.

DLEQ **optimized**
In the following we will show how one can improve the amount of computation of the challenge C. Instead of computing the challenge n times, one can compute it once and reuse the challenge.

1. The prover publish $a_{1,i} = g^{w_i} \ (mod\ q)$ *and* $a_{2,i} = y_i^{w_i} \ (mod\ q)$ for $1 \leq i \leq n,\ w_i \in_R \mathbb{Z}_q$.
2. The prover computes the hash $C = H(X_i, Y_i, ..., X_n, Y_n, a_{1,1}, a_{2,1}, a_{1,2}, a_{2,2}, ..., a_{1,n}, a_{2,n})$.
3. The prover computes r_i: $r_i = w_i - p(i) \cdot C \ (mod\ q)$ and publish r_i, C.
4. The verification contains of the following computation:
   (a) The verifier checks if: $a_{1,i} = g^{r,i} \cdot X_i^C \ (mod\ q)$
   (b) The verifier checks if: $a_{2,i} = y_i^{r_i} \cdot Y_i^C \ (mod\ q)$
   (c) The verifier checks if: $C = H(X_i, Y_i, ..., X_n, Y_n, a_{1,1}, a_{2,1}, a_{1,2}, a_{2,2}, ..., a_{1,n}, a_{2,n})$

DLEQ **computation voter**
Hence the hash contains all the a_i the prover will compute this proof once for all $p(i)$ which improve efficiency. This means that if there are 3 shares, then the above computation has to be done 3 times, one for each tally, but same hash can be computed once for every tally.

1. The prover publish $a_{1,1}, a_{2,1}, a_{1,2}, a_{2,2}, a_{1,3}, a_{2,3}$.
2. The prover publish C, r_1, r_2, r_3.
3. The prover selects $w_1, w_2, w_3 \in_R \mathbb{Z}_q$.
4. The prover computes $a_{1,i} = g_i^w \ (mod\ q)$ *and* $a_{2,i} = y_i^{w_i} \ (mod\ q)$.

5. The prover computes the hash $C = H(X_i, Y_i, ..., X_n, Y_n, a_{1,1}, a_{2,1}, a_{1,2}, a_{2,2}, ..., a_{1,n}, a_{2,n})$.
6. The prover computes $r_i : r_i = w_i - p(i) \cdot C \ (mod\ q)$.
7. The verification contains of the following computation:
   (a) The verifier checks if: $a_{1,i} = g^{r,i} \cdot X_i^C \ (mod\ q)$
   (b) The verifier checks if: $a_{2,i} = y_i^{r_i} \cdot Y_i^C \ (mod\ q)$
   (c) The verifier checks if: $C = H(X_i, Y_i, ..., X_n, Y_n, a_{1,1}, a_{2,1}, a_{1,2}, a_{2,2}, ..., a_{1,n}, a_{2,n})$

Zero knowledge proof for the $DLEQ$
The same arguments holds for correctness, soundness and zero knowledge, which is described in section 5.3.1 about the interactive $DLEQ$.

A.2 $DLEQ$ proof by the talliers

The talliers will do computations on each of their shares. Each tally uses the $DLEQ$ to prove that the decryption of their shares is done correctly. It proofs that the exponent are equal $G = y_i^{x_i}$ *and* $Y_i = S_i^{x_i}$ without revealing x_i and if the prover was honest, then it should be the case that we get same computed values in the end meaning $a_1 = G^w = G^r \cdot y_i^C$ and $a_2 = S_i^w = S_i^r \cdot Y_i^C$.

First we show the interactive proof and then transform it to a non-interactive proof. The input values are $(G,\ y_i,\ S_i, Y_i)$ where $G = y_i^{x_i}$ and $Y_i = S_i^{x_i}$. We have some initial values $g_1 = G,\ h_i = y_i,\ g_2 = S_i,\ h_2 = Y_i,\ \alpha = x_i$ and $w \in_R \{0, ..., q-1\}$.

In step 1 the prover computes $a_1 = G^w,\ a_2 = S_i^w$. In step 2 the verifier creates a challenge C. In step 3 the tally computes $r = w - C \cdot x_i$. In step 4 the verifier computes $a_1 = G^r \cdot y_i^C,\ a_2 = S_i^r \cdot Y_i^C$.

DLEQ protocol by the talliers			
Input : $G,\ y_i,\ S_i,\ Y_i$ *where* $G = y_i^{x_i}$ *and* $Y_i = S_i^{x_i}$			
Output : 0 *or* 1			
	Prover		Verifier
Step 1	$w \in_R \mathbb{Z}_q$		
	$a_1 = G^w$		
	$a_2 = S_i^w$	$\xrightarrow{a_1, a_2}$	
Step 2			$C \in_R \mathbb{Z}_q$
		$\xleftarrow{C}$	
Step 3	$r = w - x_i \cdot C$		
			checks if :
Step 4		$\xrightarrow{r}$	$a_1 = G^r \cdot y_i^C$
			$a_2 = S_i^r \cdot Y_i^C$

Figure A.2: $DLEQ$

Note the interaction in step 2 where the verifier creates a challenge to the prover. Through Fiat–Shamir we transform an interactive proof of knowledge into a non-interactive proof of knowledge by replacing step 2 with a hash algorithm $C = H(G,\ y_i,\ S_i,\ Y_i,\ a_1,\ a_2)$.

Mathematical justification
To justify correctness of the computations in step 4, we can do the following verification on $a_1 = G^w \stackrel{?}{=} G^r \cdot y_i^C$ and $a_2 = S_i^w \stackrel{?}{=} S_i^r \cdot Y_i^C$.

$$a_1 = G^r \cdot y_i^C = G^r \cdot y_{x_i}^C = G^{r+x_iC} = G^{w-Cx_i+x_iC} = G^w$$
$$a_2 = S_i^r \cdot Y_i^C = S^r \cdot Y_{x_i^C} = S^{w-Cx_i} \cdot S_i^{x_i \cdot C} = S^{w-Cx_i+x_i \cdot C} = S_i^w$$

To show soundness and zero knowledge the same process from section 5.3.1 can be followed.

Appendix B

Calculations

B.1 Simple review of calculations in the protocol

We will present a simplified example of the calculations through the protocol, which illustrate casting the votes and tallying the final counts of the votes. The structure follows the protocol as described in section 5.1. This example does not contain calculations of the proof. These are described in section 5.3. In this calculation there are 3 voters (m) and 3 talliers (n). The example shows 3 voters which cast their votes and how 3 talliers are able to reconstruct the sum of all votes.

The bulletin board publishes all system parameters which is the public elements a prime q, the generators g and G and a security parameter t.

Public elements	
Prime q	5
Security parameter t	3
Generator G	5
Generator g	9
Lambda $\lambda_1,\ \lambda_2,\ \lambda_3$	$3, -3, 1$

Table B.1: The lambda is based on the calculation from section 4.2.1.1

The tallier generates a private key x_i and a public key y_i.

Talliers		
	Public key y_i	Private key x_i
Tally 1	5	1
Tally 2	3	2
Tally 3	4	3

Table B.2: Public and private keys for the talliers

The voters casts their votes, either 0 or 1. and creates a random secret

s and a random polynomial of degree at most $t-1$ and computes the shares.

Voter 1	
Vote v	1
Random secret s	2
Random polynomial $p(x)$	$2+2x+4x^2$
Voter 2	
Vote v	1
Random secret s	3
Random polynomial $p(x)$	$3+x+4x^2$
Voter 3	
Vote v	1
Random secret s	4
Random polynomial $p(x)$	$4+3x+4x^2$

Table B.3: 3 voters creates their vote, secret and polynomial

The voters creates their shares $p(x)$				
Voter/point	$p(0)$	Tally 1	Tally 2	Tally 3
$p_1(x)=2+2x+4x^2$	2	3	2	4
$p_2(x)=3+x+4x^2$	3	3	1	2
$p_3(x)=4+3x+4x^2$	4	1	1	4

Table B.4: The shares are computed $p_1(1) = 3 \ (mod\ 5)$, $p_1(2) = 2 \ (mod\ 5)$, $p_1(3) = 4 \ (mod\ 5)$ etc.

The voter distributes the encrypted share.

Encryption of the shares $Y_i = y^{p(i)}$ using the talliers public key			
Voter/Talliers	Tally 1	Tally 2	Tally 3
Voter 1	$5^3=4$	$3^2=9$	$4^4=3$
Voter 2	$5^3=4$	$3^1=3$	$4^2=5$
Voter 3	$5^1=5$	$3^1=3$	$4^4=3$

Table B.5: The encryption consist of raising the share in the exponent on the Talliers public key such as $y_i^{p_j(i)} \ (mod\ 11)$.

The tallier multiplies the encrypted shares Y_i^* and decrypt the multiplum of shares S_i^*.

Tallier computes Y^*	
Tally 1	$Y_1^* = (4 \cdot 4 \cdot 5) = 80 = 3 \ (mod\ 11)$
Tally 2	$Y_2^* = (9 \cdot 3 \cdot 3) = 81 = 4 \ (mod\ 11)$
Tally 3	$Y_3^* = (3 \cdot 5 \cdot 3) = 45 = 1 \ (mod\ 11)$
Tallier computes S^*	
Tally 1	$(x_1)^{-1} = 1 \cdot x = 1 \ (mod\ 5) = 1,\ S_1^* = 3^1 = 3 \ (mod\ 11)$
Tally 2	$(x_2)^{-1} = 2 \cdot x = 1 \ (mod\ 5) = 3,\ S_2^* = 4^3 = 9 \ (mod\ 11)$
Tally 3	$(x_3)^{-1} = 3 \cdot x = 1 \ (mod\ 5) = 2,\ S_3^* = 1^2 = 1 \ (mod\ 11)$

Table B.6: The talliers computes Y^* and S^*.

A master authority applies Lagrange interpolation

Apply lambda to S^*	
$S_1^{*\cdot\lambda_1}$	$3^{3 \ (mod\ 5)} = 3^3 \ (mod\ 11) = 5$
$S_2^{*\cdot\lambda_2}$	$9^{-3 \ (mod\ 5)} = 9^2 \ (mod\ 11) = 4$
$S_3^{*\cdot\lambda_3}$	$1^{1 \ (mod\ 5)} = 1^1 \ (mod\ 11) = 1$
Multiply S^* which is the sum of the secrets	
$G^{\sum_{j=1}^{m} s_j} = S_1^{*\cdot\lambda_1} \cdot S_2^{*\cdot\lambda_2} \cdot S_3^{*\cdot\lambda_3}$	$5 \cdot 4 \cdot 1 \ (mod\ 11) = 9$

Table B.7: A master authority multiples the decrypted shares.

A master authority computes the votes

The values of $U = G^{s+v}$	
U_1 for voter 1	$5^{2+1} = 4 \ (mod\ 11)$
U_2 for voter 2	$5^{3+1} = 9 \ (mod\ 11)$
U_3 for voter 3	$5^{4+1 \ (mod\ 5)} = 5^0 = 1 \ (mod\ 11)$
Multiply the U_i which is the sum of the secrets and the votes	
$\prod_{j=1}^{m} U_j = U_1 \cdot U_2 \cdot U_3 = G^{\sum_{j=1}^{m} s_j + v_j}$	$(4 \cdot 9 \cdot 1) \ (mod\ 11) = 3$

Table B.8: A master authority multiplies all the votes.

Compute the final sum of the votes $G^{\sum_{j=1}^{m} v_j}$ **through** $U_i/(S_i^*)^\lambda = G^{\sum_{j=1}^{m} s_j+v_j}/G^{\sum_{j=1}^{m} s_j}$	
$G^{\sum_{j=1}^{m} s_j+v_j}$	3
$G^{\sum_{j=1}^{m} s_j}$	9
Inverse of $(G^{\sum_{j=1}^{m} s_j})^{-1}$	$9 \cdot x = 1 \ (mod\ 11) = 9 \cdot 5 = 45 \ (mod\ 11) = 1$
$G^{\sum_{j=1}^{m} s_j+v_j}/G^{\sum_{j=1}^{m} s_j}$	$3 \cdot 5 \ (mod\ 11) = 4$

Table B.9: We can isolate the sum of all votes by multiplying with inverse. Hereafter one can use exhaustive search to extract the final tally.

The final computation is solving the following $G^x \ (mod\ 11) = 4$ where x is the total vote count. Since $5^3 \ (mod\ 11) = 4$ the total vote count is 3, which is the correct vote count since we know that the three voters voted 1 which gives a total of 3 votes.

B.2 Simple review of calculation of $DLEQ$ between voter and verifier

We will present a simplified example of the calculations through the $DLEQ$, which proofs that the shares are constructed correctly and consistent as described in section 5.3.1. The calculation will be based on the values from appendix B.1. We will use voter 1 with a polynomial $2 + 2x + 4x^2$ and we will use tally 1, 2 and 3 as the verifiers. This means we will also use calculation with these values $y_1 = 5,\ Y_1 = 4,\ y_2 = 3,\ Y_2 = 9,\ y_3 = 4,\ Y_3 = 3$ since the $DLEQ$ needs the variables g, X_i, y_i, Y_i. Since the example is based on 3 shares, there will be 3 corresponding $DLEQ$ proves. To compute $X_1,\ X_2,\ X_3$ we first need to compute the C_j.

Voter 1 computes $C_j \ (g^\alpha)$		
	α_i	g^α
C_0	2	$9^2 \ (mod\ 11) = 4$
C_1	2	$9^2 \ (mod\ 11) = 4$
C_2	4	$9^4 \ (mod\ 11) = 5$

Table B.10: The α is the coefficiens from voter 1 polynomial, where $\alpha_0 = 2,\ \alpha_1 = 2,\ \alpha_2 = 4$.

Voter 1 computes $X_1 = \prod_{j=0}^{t-1} C_j^{1^j}$ **for share** $p(1)$		
	i^j	C^{i^j}
$C_0^{0^1}$	$1^0 = 1$	$4^1 = 4\ (mod\ 11)$
$C_1^{1^1}$	$1^1 = 1$	$4^1 = 4\ (mod\ 11)$
$C_2^{2^1}$	$1^2 = 1$	$5^1 = 5\ (mod\ 11)$
$X_1 = \prod_{j=0}^{t-1} C_j^{1^j} = g^{p(1)} = (4 \cdot 4 \cdot 5) = 3\ (mod\ 11)$		

Table B.11: Computation of X_1 by voter 1.

Voter 1 computes $X_2 = \prod_{j=0}^{t-1} C_j^{2^j}$ **for share** $p(2)$		
	i^j	C^{i^j}
$C_0^{0^2}$	$2^0 = 1$	$4^1 = 4\ (mod\ 11)$
$C_1^{1^2}$	$2^1 = 2$	$4^2 = 5\ (mod\ 11)$
$C_2^{2^2}$	$2^2 = 4$	$5^1 = 9\ (mod\ 11)$
$X_2 = \prod_{j=0}^{t-1} C_j^{2^j} = g^{p(2)} = (4 \cdot 5 \cdot 9) = 4\ (mod\ 11)$		

Table B.12: Computation of X_2 by voter 1.

Voter 1 computes $X_3 = \prod_{j=0}^{t-1} C_j^{3^j}$ **for share** $p(3)$		
	i^j	C^{i^j}
$C_0^{0^3}$	$3^0 = 1$	$4^1 = 4\ (mod\ 11)$
$C_1^{1^3}$	$3^1 = 3$	$4^3 = 9\ (mod\ 11)$
$C_2^{2^3}$	$3^2 = 9 = 4$	$5^4 = 9\ (mod\ 11)$
$X_3 = \prod_{j=0}^{t-1} C_j^{3^j} = g^{p(3)} = (4 \cdot 9 \cdot 9) = 5\ (mod\ 11)$		

Table B.13: Computation of X_3 by voter 1.

DLEQ protocol

Input : $g = 9, X_1 = 3, y_1 = 5, Y_1 = 4$
where $X_1 = g^{p(1)} = 9^3 = 3$ *and* $Y_1 = y_1^{p(1)} = 5^3 = 4$

	Prover		Verifier
Step 1	$w = 4$		
	$a_1 = g^w = 9^4 = 5$		
	$a_2 = y_2^w = 5^4 = 9$	$\xrightarrow{a_1,\ a_2}$	
Step 2			$C = 3$
		$\xleftarrow{C}$	
Step 3	$r = w - p(1) \cdot C$		
	$r{=}4{-}3{\cdot}3{=}{-}5{=}0$		
			checks if :
			$a_1 = g^r \cdot X_1^C$
Step 4		$\xrightarrow{r}$	$a_1{=}9^0{\cdot}3^3{=}1{\cdot}5{=}5$
			$a_2 = y_1^r \cdot Y_1^C$
			$a_2{=}5^0{\cdot}4^3{=}1{\cdot}9{=}9$

Figure B.1: *DLEQ* interactive proof for X_1

DLEQ protocol

Input : $g = 9, X_2 = 4, y_2 = 3, Y_2 = 9$
where $X_2 = g^{p(2)} = 9^2 = 4$ *and* $Y_2 = y_2^{p(2)} = 3^2 = 9$

	Prover		Verifier
Step 1	$w = 4$		
	$a_1 = g^w = 9^4 = 5$		
	$a_2 = y_2^w = 3^4 = 4$	$\xrightarrow{a_1,\ a_2}$	
Step 2			$C = 3$
		$\xleftarrow{C}$	
Step 3	$r = w - p(2) \cdot C$		
	$r{=}4{-}2{\cdot}3{=}{-}2{=}3$		
			checks if :
			$a_1 = g^r \cdot X_2^C$
Step 4		$\xrightarrow{r}$	$a_1{=}9^3{\cdot}4^3{=}3{\cdot}9{=}5$
			$a_2 = y_2^r \cdot Y_2^C$
			$a_2{=}3^3{\cdot}9^3{=}5{\cdot}3{=}4$

Figure B.2: *DLEQ* interactive proof for X_2

DLEQ protocol

$Input : g = 9, X_3 = 5, y_3 = 4, Y_3 = 3$

$where\ X_3 = g^{p(3)} = 9^4 = 5\ and\ Y_3 = y_3^{p(3)} = 4^4 = 3$

	Prover		Verifier
Step 1	$w = 4$		
	$a_1 = g^w = 9^4 = 5$		
	$a_2 = y_3^w = 4^4 = 3$	$\xrightarrow{a_1,\ a_2}$	
Step 2			$C = 3$
		$\xleftarrow{C}$	
Step 3	$r = w - p(3) \cdot C$		
	$r{=}4{-}4{\cdot}3{=}{-}8{=}2$		
			$checks\ if:$
			$a_1 = g^r \cdot X_i^C$
Step 4		$\xrightarrow{r}$	$a_1{=}9^2{\cdot}5^3{=}4{\cdot}4{=}5$
			$a_2 = y_3^r \cdot Y_3^C$
			$a_1{=}4^2{\cdot}3^3{=}5{\cdot}5{=}3$

Figure B.3: *DLEQ* interactive proof for X_3

B.3 Simple review of calculation of $PROOF_u$ between voter and verifier

We will present a simplified example of the calculations through the proof $PROOF_u$. As described in section 5.3.2 the $PROOF_U$ proofs that the vote either is 0 or 1 without revealing the actual value of the vote. The calculation will be based on the values from appendix B.1. The example will show a voter which votes 1. We use voter 1 with vote $v = 1$ and his secret $s = 2$ in this example.

$PROOF_U$ **protocol**

$Public: U = G^{s+v} = 5^{(2+1)} = 4,$
$C_0 = g^s = 9^2 = 4$

	Prover	Verifier
Step 1b	$vote(v) = 1$ $w = 4,$ $r_0 = 4,$ $d_0 = 4,$ $a_0{=}g^{r_0}\cdot C_0^{d_0}{=}9^4\cdot 4^4{=}5\cdot 3{=}4,$ $a_1{=}g^{w}{=}9^4{=}5$ $b_0{=}G^{r_0}\cdot U^{d_0}{=}5^4\cdot 4^4{=}9\cdot 3{=}5,$ $b_1{=}G^{w}{=}5^4{=}9$	
	$\xrightarrow{a_0,a_1,b_0,b_1}$ **Publish to bulletin**	
		Publish to bulletin
Step 2		$C = 3$
		$\xleftarrow{C}$
Step 3b	$d_1{=}C{-}d_0 \ mod\ q{=}3{-}4{=}{-}1{=}4,$ $r_1{=}w{-}s{\cdot}d_1 \ mod\ q{=}4{-}(2{\cdot}4){=}1$	
	$\xrightarrow{d_0,\ r_0,\ d_1,\ r_1}$ **Publish to bulletin**	
		Verification:
Step 4		$C{=}d_1{+}d_0{=}4{+}4{=}8{=}3,$ $a_0{=}g^{r_0}\cdot C_0^{d_0}{=}9^4\cdot 4^4{=}5\cdot 3{=}4$ $b_0{=}G^{r_0}\cdot U^{d_0}{=}5^4\cdot 4^4{=}9\cdot 3{=}5,$ $a_1{=}g^{r_1}\cdot C_0^{d_1}{=}9^1\cdot 4^4{=}9\cdot 3{=}5,$ $b_1{=}G^{r_1}\cdot(\frac{U}{G})^{d_1}{=}5^1\cdot(4/5)^4$ $b_1{=}5{\cdot}(4{\cdot}9)^4{=}5{\cdot}4{=}9$

Figure B.4: $PROOF_U$

Appendix C

Quality attribute scenario

C.1 QAS - Availibility

Scenario(s):		# 6: An internal crash occurs and the bulletin board is out of reach during normal operation. The response is that the error is logged and the system is running in degraded mode. The system should be up running within 5 minutes.
Relevant Quality Attributes:		Availibility
Scenario Parts	**Source:**	Internal
	Stimulus:	Crash
	Artifact	Bulletin board
	Environment:	Normal operation
	Response:	Error is logged
	Response Measure:	The system is running in degraded mode in max 5 minutes

Table C.1: Availibility QAS

C.2 QAS - Performences

Scenario(s):		# 7: 5 mill. users intiate votes to the bulletin board under normal operation. The votes are processed and saved with average latency of 2 seconds.
Relevant Quality Attributes:		Performence
Scenario Parts	**Source:**	5 mill. users
	Stimulus:	Initiate their votes
	Artifact	Bulletin board
	Environment:	Normal operation
	Response:	The votes are processed and saved
	Response Measure:	With average latency of 2 seconds

Table C.2: Performence QAS

C.3 QAS - Modifiability

Scenario(s):		# 9: A developer needs to replace the user interface for the voter client under design time. The replacement is made within 3 hours.
Relevant Quality Attributes:		Modifiability
Scenario Parts	**Source:**	A developer
	Stimulus:	Needs to replace the user interface
	Artifact	Voter client interface
	Environment:	Design time
	Response:	Replacement made
	Response Measure:	Within 3 hours

Table C.3: Modifiability QAS

Appendix D

Installation guides

D.1 Technology stack

We have used the following technologies:

1. IDE: Microsoft Visual studio
2. Languages: ASP.NET, Javascript and C#
3. Database: Mongo database
4. Testning: Jasmine
5. Modern browser

D.2 Installation guide for Visual studio and IIS express

1. Download Visual studio:
   https://www.visualstudio.com/vs/visual-studio-express/
2. Consult microsoft for installations guide for Visual studio:
   https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio
3. If the IIS express do not install with installation of Visual studio then follow this installation guide.

   Download IIS express:
   https://www.microsoft
   .com/en-us/download/details.aspx?id=48264

D.3 Running the code

D.3.1 Installing the electronic voting application

1. Unzip the file containing the project PublicVerifiableSecretSharing.zip into a local folder

2. Start the project by clicking the PublicVerifiableSecretSharing.sln inside the project folder just unzipped.

3. Set PVSS.Client.Web as StartUp project, as illustrated below

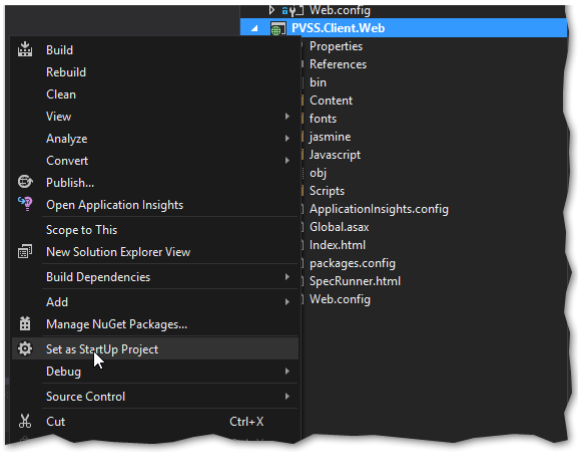


Figure D.1: Visual Studio - set start project

4. Start the application by press F10 inside Visual Studio.

   This will start both the Client web server and BulletinBoard webAPI. In the bottom right corner an icon of IIS Express should be visible, by right click this icon, it should be visuable that both web service is running

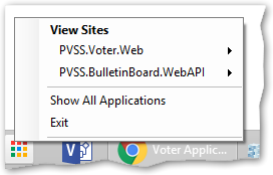


Figure D.2: Homepage

5. When starting the application the following screen should be presented in your default browser as illustrated below. Alternatively if the programs runs in the visual studio, then paste this url http://localhost:5751/ into a browser.

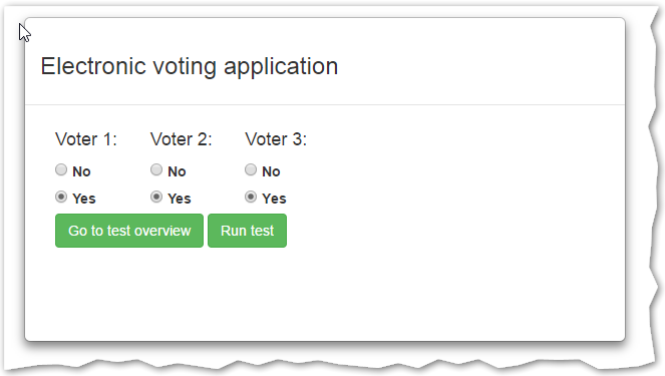


Figure D.3: Homepage

6. By clicking the button [Run test] an election will start with 3 voters.

   Hereafter a log of all the actions in the electronic voting scheme will be listed below. It is possible to change what the voters vote by clicking on the radio buttons. By default all voters votes "yes".

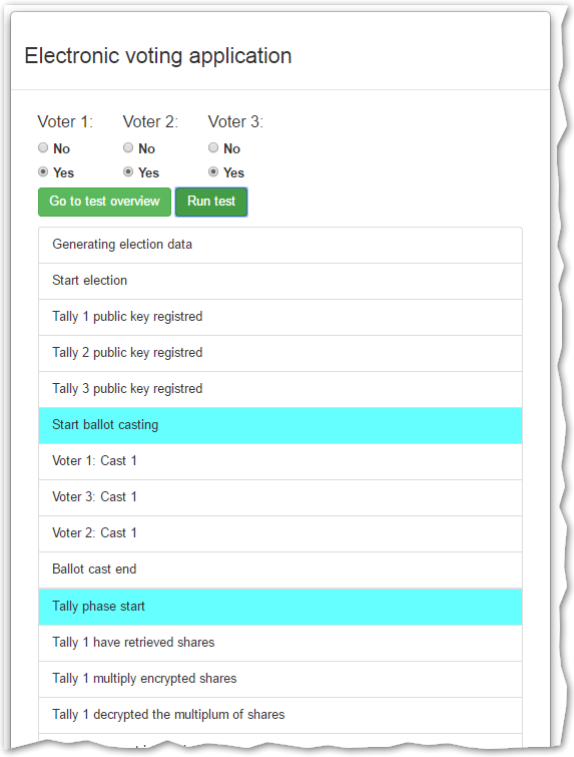


Figure D.4: Homepage with result

7. Navigate to Jasmine tests by clicking the [Go to test overview] on the homepage

This page shows the test results of the unit tests define in PVSS.Client .Web.jasmine.spec

Figure D.5: Jasmine testpage