Bern University of Applied Sciences (BFH)

Bachelor Thesis

CHVote Demonstrator

Authored by Kevin Häni <kevin.haeni@gmail.com>
Yannick Denzer <yannick@denzer.ch>
Supervised by Prof. Dr. Rolf Haenni <rolf.haenni@bfh.ch>

Bern, November 10, 2017

Contents

1 Introduction 5

1.1 Electronic voting 5

1.2 Project task 5

2 Management Summary 7

3 CHVote Protocol 9

3.1 Protocol 9

3.2 Goals 9

3.3 Protocol Participants 9

3.3.1 Voter 9

3.3.2 Election Administrator 9

3.3.3 Election Authorities 10

3.3.4 Bulletin Board 10

3.3.5 Printing Authority 10

3.4 The basic idea of the CHVote protocol 10

4 Project Plan 11

4.1 Project Method 11

4.2 Requirements 11

4.2.1 General Requirements 11

4.2.2 Election-Overview 11

4.2.3 Election Administrator 11

4.2.4 Printing Authority 12

4.2.5 Voter 12

4.2.6 Bulletin Board 12

4.3 Timeplan 12

5.4.1 Project structure 17

5.4.2 Public parameters 17

5.4.3 Coding style 18

5.4.4 Return types 20

5.4.5 Protocol 21

6 Journal 31

6.1 Week 1 31

6.1.1 Reflexion 31

6.1.2 Next steps 31

6.2 Week 2 31

6.2.1 Reflexion 31

6.2.2 Next steps 32

6.3 Week 4 32

6.3.1 Reflexion 32

6.3.2 Next steps 32

7 Conclusion 37

7.1 Python drawbacks 37

1 Introduction

1.1 Electronic voting

Since 2015, it is possible for Swiss citizens registered in the cantons Geneva and Neuchâtel and living abroad, to vote electronically. However, these systems did not yet meet the requirements in terms of security and transparency, to be accepted as a secure E-Voting platform on a nationwide scale.

One of the requirements that is hardest to achieve, is that the system must ensure the voters privacy while at the same time, it must be verifiable that only valid votes have been counted.

A contract was formed between the state of Geneva and the Bern University of Applied Sciences to work out a new protocol which does meet the complex requirements set up by the government. In 2017, the resulting specification document written by Haenni Rolf, Philipp Locher and Reto E. Koenig has been officially published and a proof-of-concept / prototype has been successfully implemented by the State of Geneva.

1.2 Project task

Understanding such a complex protocol isn't easy and might be the reason why many people still do not trust electronic e-voting systems. In close consultation with the authors of the CHVote specification, we agreed to develop an application that allows users to get a hands-on experience with the CHVote e-voting system and makes it possible to show to an audience how the future of voting in Switzerland might possibly look like.

For this reason, we have implemented the protocol according to the specification and developed a web-based application on top of it, which allows to perform every step of an election, from generating the electorate data, to casting and confirming ballots from a voters point of view, to the post-election processes like mixing, decryption and tallying.

2 Management Summary

3 CHVote Protocol

3.1 Protocol

The protocol our solution is based on, does not originate from us! The concept and specification has been created by Rolf Haenni and his team at Institute for Security in the Information Society (RISIS) of the Bern University of Applied Sciences. For this project, we have implemented the protocol according to their specification. In this chapter we summarize the most important aspects of the protocol for better understanding.

3.2 Goals

As pointed out earlier, one of the big challenges an E-Voting protocol has to solve is the verifiability of the voting result while still ensuring the privacy of all voters. Another big problem e-voting systems are facing is the risk of a voting client being infected by malware which manipulates casting of a vote without the voters notice. Both of these issues are addressed by the use of modern cryptography.

3.3 Protocol Participants

3.3.1 Voter

A voter is a person who is eligible to vote in his state. Every voter must possess a voting card that has been sent to him prior to an election event and that contains several codes such as the voting code and the confirmation code, which are used to identify the voter during the vote casting process.

A voter uses a voting client (website) to form a ballot according to the protocol, by entering his selection, a voting code to cast, and later in the process, his confirmation code to confirm his vote. Verification codes and a finalization code are displayed by the voting client to ensure the voter that his vote has been cast as intended and hasn't been manipulated by a third party.

3.3.2 Election Administrator

The election administrator, typically a person of the government, sets up the election by providing the necessary information such as the candidates and the voters. He is also responsible for determining and publishing the final results of the election.

3.3.3 Election Authorities

3.3.4 Bulletin Board

3.3.5 Printing Authority

3.4 The basic idea of the CHVote protocol

Before the actual election, voting sheets are generated and printed for the whole electorate and delivered to the voters by a trusted mailing service. The voting sheets contain several codes, namely:

- voting code
- confirmation code
- finalization code
- one verification code for every candidate

The voting and confirmation code are authentication codes used to authenticate the voter.

The voter first selects candidates by entering their indices. The voting client then forms a ballot containing the voters selection encrypted with the authorities public key and authenticated with the voters personal voting code. Additionally, the ballot contains a query that queries the authorities for the corresponding verification codes of the selected candidates, without the server knowing which candidates the voter has selected. The voter then checks if the returned verification codes match the codes of the candidates he has chosen on the printed voting sheet. If the selection was somehow manipulated by malware, the returned verification codes would not match the printed ones and the voter would have to abort the vote casting process. This way the integrity of the vote casting can be assured even in the presence of malware. Privacy on the other hand cannot be protected since the malware will learn the plaintext of the voter's selection.

In order to verify that a voter has formed the ballot correctly by choosing exactly the number of candidates he is supposed to choose, the following trick is being used: the verification codes are derived from $n = \sum_{n=1}^{t} n_j$ random points on t polynomials (one for every election event j) of degree $k_j - 1$, that each election authority has chosen randomly prior to the election. By learning exactly $k = \sum_{n=1}^{t} k_j$ points on these polynomials, the voting client is able to reproduce these polynomials and therefore is able to calculate a particular point with $x = 0$ on these polynomials. The corresponding y values are incorporated into the second voting credential from which the confirmation code is derived. Only if the voter knows these values (by submitting a valid candidate selection), he will be able to confirm the vote that he casted.

Since there is still a connection between the encrypted ballot and the voter at this point, the encrypted candidate selection is extracted from the ballots before tallying. After that, every authority is shuffling/mixing these encryptions in order to make it impossible to find out which voter has submitted which encrypted ballot. This mixing of the encrypted votes is done by using the homomorphic property of the ElGamal encryption scheme. Re-encryption of the ballots multiplied with the neutral element 1 yields a new ciphertext for the same plaintext.

The public key that is used for encryption has been generated jointly by all authorities. Therefore in order to decrypt the result, all authorities must provide their share of the private key. The measure of multiple authorities participating in the whole e-voting process ensures the security of the whole election even if only one authority can be trusted.

4 Project Plan

4.1 Project Method

4.2 Requirements

4.2.1 General Requirements

	Description	Must	Priority	Iteration	Status
R1	The CHVote protocol is implemented as specified in the latest specification document. The only exclusion are the algorithms for channel security.	Must	High	1	Done
R2	The application is web-based shows updates within the same demo-election in real-time.	Must	High	1	Done
R3	The system supports 1-out-of-3 type of elections (e.g. elect 1 of 3 possible candidates)	Must	High	1	Done
R4	The system supports internationalization. Providing more than one language is not required.	Must	High	1	Done
R5	Users can create new elections	Must	High	1	Done
R6	The system can handle k-out-of-n type of elections	Must	Medium	1	Pending

4.2.2 Election-Overview

	Description	Must	Priority	Iteration	Status
R6	The overview shows which phase the election is currently in	Must	Medium	Iteration 2	
R7	A graphical scheme of the chVote protocol gives an overview of all participating parties	Must	Medium	Iteration 2	

4.2.3 Election Administrator

	Description	Must	Priority	Iteration	Status
R8	An election can be set up by providing all required information such as the candidates, number of parallel voters, the number of voters and the number of selections (simplified JSON input)	Must	High	Iteration 1	Done
R8	The election can be set up without entering the parameters in JSON format and allows easier set up of elections with multiple parallel election events	Low	Iteration 2		
R9	The election administrator view allows to perform the tallying and displays the final result of an election in numbers and a pie chart	Must	High	Iteration 1	
R10	During election setup, the security parameters can be chosen from a set of predefined parameters	Can	Low	Iteration 2	

4.2.4 Printing Authority

	Description	Must	Priority	Iteration	Status
R8	Users can generate and display voting cards for an election.	Must	High	Iteration 1	Done
R8	Voting cards hide sensitive information behind a scratch card	Can	High	Iteration 2	

4.2.5 Election Authority

	Description	Must	Priority	Iteration	Status
R8	The election authority view shows all information known to an election authority	Must	High	Iteration 1	Done
R8	After a voter has submitted a ballot, all election authorities can check and respond to the voters submission	Must	High	Iteration 1	Done
R8	In the post-election phase, all election authorities can perform the mixing and decryption tasks	Must	High	Iteration 2	
R9	Each authority has a setting such that it automatically processes all tasks	Can	High	Iteration 2	Partially done

4.2.6 Voter

	Description	Must	Priority	Iteration	Status
R8	Users are able to go through the whole vote-casting process for every voter	Must	High	Iteration 1	Done
R8	The voting card of a voter is displayed on screen. The voting and confirmation codes can be copied into the input textfields by double clicking	Must	Medium	Iteration 1	Partially done

4.2.7 Bulletin Board

	Description	Must	Priority	Iteration	Status
R8	The bulletin board view shows what information is publicly available	Must	High	Iteration 1	Done
R8	The bulletin board view is extended with verification-functionality	Can	Low	Iteration 2	

4.3 Timeplan

5 Implementation Details

5.1 Components

From a highlevel perspective, our application consists of two components which themselve may consist of multiple sub components:

- Webapplication
- Backend
  - REST Service
  - Datasync Service
  - VoteSimulation (based on our CHVote crypto-library)

5.2 Architecture

Each visitor of our app is served a singlepage-webapplication (SPA) that runs in the webbrowser. The app consists of webcomponents that are bound to a local datastore that reflects the current state of the election which the user has previously selected.

When the user triggers some action that results in a mutation on the backend, a HTTP call to our REST backend service is created.

5.3 Frontend

5.4 Backend

5.4.1 Project structure

We decided to put every algorithm of the specification in its own file together with related unit tests. The files are structured according to the actors of the protocol, for example:

- **Common**: contains common cryptopgraphic algorithms and the security parameters used by multiple algorithms
- **ElectionAuthority**: contains all the algorithms used by the election authority
- **PrintingAuthority**: contains all the algorithms used by the printing authority
- **VotingClient**: contains all the algorithms used by the voting client

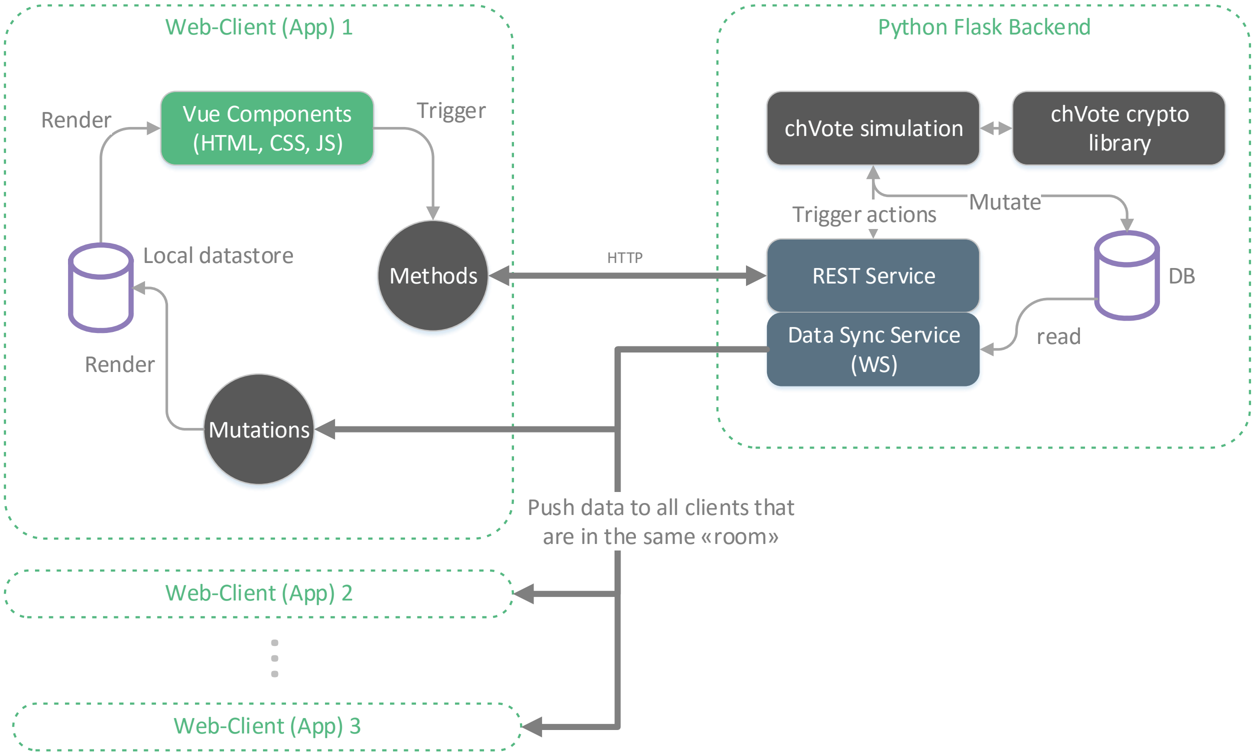

Figure 5.1: Architecture

- **ElectionAdministration**: contains all the algorithms used by the election administrator
- **Utils**: contains helper classes and miscellaneous utility functions
- **Protocol**: contains the protocol implementation
- **profiles**: contains JSON files that are used to define election parameters

5.4.2 Public parameters

There exist two types of public parameters:

The **security relevant parameters**, e.g:

- The order of the prime groups: p, $'p$, $\hat{p}$
- The length of the voting, confirmation, return and finalization codes
- The number of authorities: s

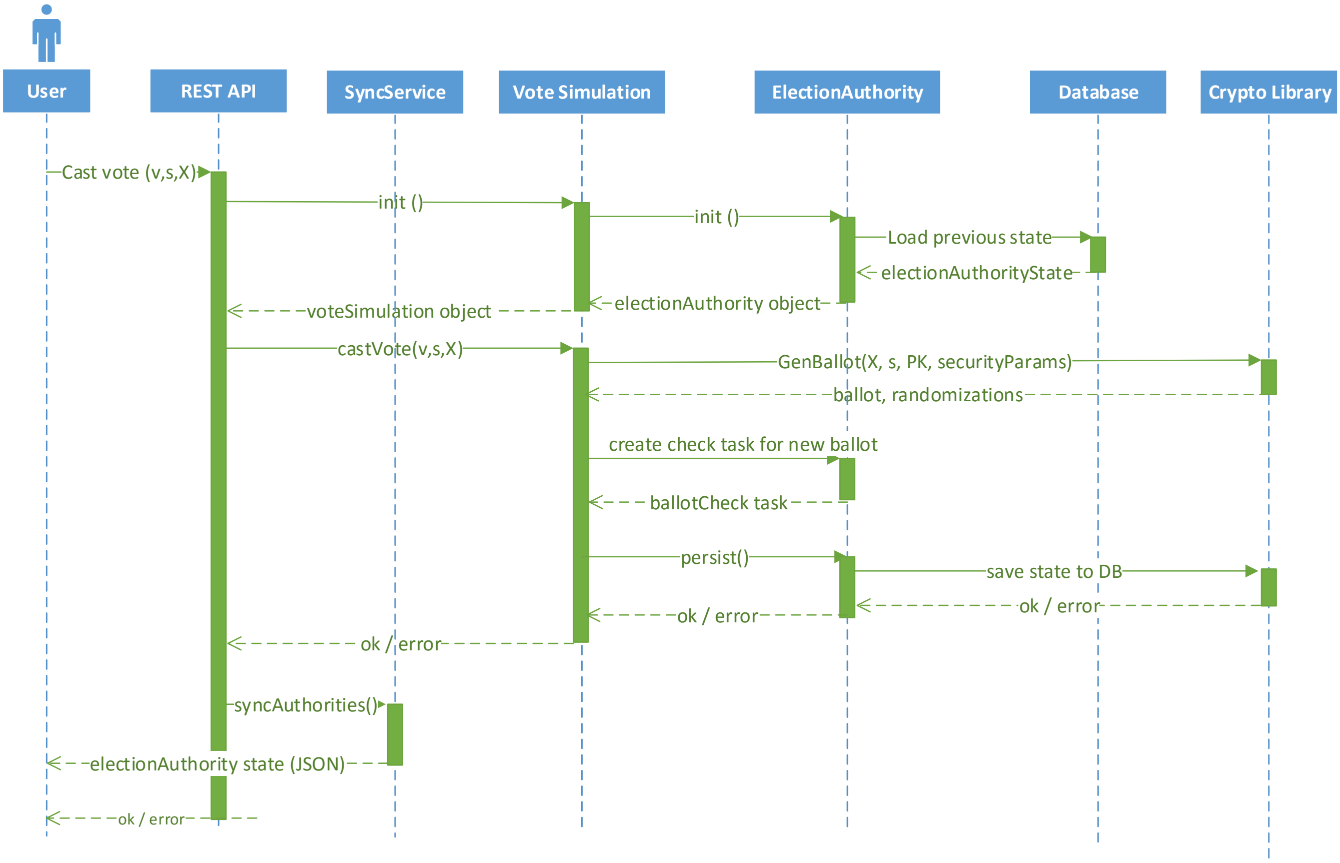

Figure 5.2: Vote casting sequence diagram

and **public election parameters**, e.g.:

- The size of the electorate: N_E
- The number of candidates: n
- The list of candidate descriptions: c

The security parameters are typically used within the algorithms and remain unchanged for a longer time period, whereas the public election parameters are only used by the protocol implementations and change with every election.

The object `SecurityParams` holds all security relevant parameters and is injected as an additional function argument to all algorithms. Several different `SecurityParams` objects are created initially, which contain all the parameters according to the recommendations in the CHVote specification document ("level 0" for testing purposes and "level 1" through "level 3" for actual use of the protocol). This approach allows us to use different levels of security during development of the algorithms and protocols. For simple unit testing we used "level 0" in order to inject the security parameters recommended for testing puposes. For actual test runs of the project the security parameters from "level 2" were used.

The public election parameters are defined in a JSON file and simply read as an object which is directly accessed by the protocol. If an algorithm needs to know certain election parameters (like the size of the electorate N_E), these values are typically derived from vectors that they have access to, so they do not require specific knowledge of these parameters.

The following is a exmaple for the contents of a JSON file containing all the parameters:

```json
{
  "autoGenerateVoters" : true,
  "numberOfVotersToGenerate" : 50,
  "voters" : [
    {
      "name" : "Voter1",
      "selection" : "1,5"
    },
    {
      "name" : "Voter2",
      "selection" : "0,5"
    },
    {
      "name": "Voter3",
      "selection": "0,5"
    }
  ],
  "t" : 2,
  "n" : [5, 3],
  "c" : ["Hillary Clinton", "Donald Trump", "Vladimir Putin", "Marine Le Pen", "May", "Yes", "No", "Empty"],
  "k" : [1, 1],
  "E" : [[1, 1], [1, 1], [1, 1]],
  "securityLevel" : 2,
  "deterministicRandomGeneration" : false
}
```

5.4.3 Coding style

The following source code sample shows a typical implemation of an algorithm (in this exmaple, algorithm 7.18 according to the CHVote specification).

```python
import unittest
import os, sys
from gmpy2 import mpz
import gmpy2

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from Utils.Utils                    import AssertMpz, AssertList, AssertClass, AssertString
from Crypto.SecurityParams          import SecurityParams, secparams_l0
from Utils.ToInteger                import ToInteger
from VotingClient.GetSelectedPrimes import GetSelectedPrimes
from VotingClient.GenQuery          import GenQuery
from VotingClient.GenBallotProof    import GenBallotProof
from UnitTestParams                 import unittestparams
from Types                          import Ballot
from Utils.StringToInteger          import StringToInteger

```

```
def GenBallot(X_bold, s, pk, secparams):
    """
    Algorithm 7.18: Generates a ballot based on the selection s and the voting code X. The
    ballot includes an OT query a and a proof pi. The algorithm also returns the random
    values used to generate the OT query. These random values are required in Alg. 7.27
    to derive the transferred messages from the OT response, which itself is generated by
↪   Alg. 7.25.

    Args:
        X_bold (str):                       Voting Code X ∈ A_X^l_X
        s (list of int):                    Selection s = (s_1, ... , s_k), 1 <= s_1 < ... <
↪   s_k
        pk (mpz):                           ElGamal key pk ∈ G_p \ {1}
        secparams (SecurityParams):         Collection of public security parameters

    Returns:
        tuple:                              alpha = (r, Ballot) = (r, (x_hat, a, b, pi))
    """

    AssertMpz(pk)
    AssertList(s)
    AssertClass(secparams, SecurityParams)

    x = mpz(StringToInteger(X_bold, secparams.A_X))
    x_hat = gmpy2.powmod(secparams.g_hat, x, secparams.p_hat)

    q_bold = GetSelectedPrimes(s, secparams)                    # q = (q_1, ... , q_k)
    m = mpz(1)

    for i in range(len(q_bold)):
        m = m * q_bold[i]

    if m >= secparams.p:
        return None

    (a_bold, r_bold) = GenQuery(q_bold, pk, secparams)
    a = mpz(1)
    r = mpz(0)

    for i in range(len(a_bold)):
        a = (a * a_bold[i]) % secparams.p
        r = (r + r_bold[i]) % secparams.q

    b = gmpy2.powmod(secparams.g,r, secparams.p)
    pi = GenBallotProof(x,m,r,x_hat,a,b,pk, secparams)
    alpha = Ballot(x_hat,a_bold,b,pi)

    return (alpha, r_bold)

class GenBallotTest(unittest.TestCase):
    def testGenBallot(self):
        selection = [1,4]       # select candidates with indices 1,4
        (ballot, r) = GenBallot(unittestparams.X, selection, unittestparams.pk,
         ↪  secparams_l0)
        print(ballot)
        print(r)
```

```python

if __name__ == '__main__':
    unittest.main()
```

All algorithms contain a short description, which was taken as-is from the specification document, as well as a comment (Google-style documentation string), which can be used to automatically generate code documentation. The algorithm itself is implemented as close to the specification as possible, using the same variable names and (as far as the language supports it) similar control structures:

- The suffix `_bold` for emphasized (bold) variables, e.g. `p_bold` for $\mathbf{p}$
- The suffix `_hat` for variables with a hat, e.g. `a_hat` for $\hat{a}$
- The suffix `_prime` for variables with a prime, e.g. `a_prime` for a'
- etc.

Each file also contains unit test relevant to the specific algorithm (if unit testing was considered useful for the particular algorithm).

The following example shows the similarities between the algorithm pseudo code and the actual implmentation in Python:

```python
x = mpz(StringToInteger(X_bold, secparams.A_X))
x_hat = gmpy2.powmod(secparams.g_hat, x, secparams.p_hat)
q_bold = GetSelectedPrimes(s, secparams)

m = mpz(1)
for i in range(len(q_bold)):
    m = m * q_bold[i]

if m >= secparams.p:
    return None

(a_bold, r_bold) = GenQuery(q_bold, pk, secparams)
```

```python
a = mpz(1)
r = mpz(0)

for i in range(len(a_bold)):
    a = (a * a_bold[i]) % secparams.p
    r = (r + r_bold[i]) % secparams.q

b = gmpy2.powmod(secparams.g,r, secparams.p)
pi = GenBallotProof(x,m,r,x_hat,a,b,pk, secparams)
alpha = Ballot(x_hat,a_bold,b,pi)

return (alpha, r_bold)
```

5.4.4 Return types

In most cases, when an algorithm returns more than a scalar datatype, tuples are used. Tuples allow to return multiple values from a function:

```python
def foo():
   return (1, 2)

def main():
   a, b = foo()
```

This way a lot of the source code looked very similar to the pseudo code in the CHVote specification. For more complex data types or return values that are used more often, named tuples were used. The data type "namedtuple" is like a lightweight class and allows access to named properties.

```python
Ballot = namedtuple("Ballot", "x_hat, a_bold, b, pi")

def main():
   Ballot b = getBallot()
   x_hat = b.x_hat
```

By following this approach we can avoid having hundreds of classes only used to pass data structures between the algorithms.

5.4.5 Protocol

Upon completion of the algorithm implmentation we have built a protocol layer according to the specification. For that purpose we created a seperate entity for every protocol participant, namely the following ones:

- **VotingClient**
- **ElectionAuthority**
- **BulletinBoard**
- **PrintingAuthority**

The following example shows the first step of protocol 6.5:

```python
def castVote(self, s, autoInput, secparams):
    self.pk = GetPublicKey(self.bulletinBoard.pk_bold,secparams)

    X = input('Enter your voting code: ')
    (self.alpha, self.r) = GenBallot(X, s, self.pk, secparams)

    return (self.alpha, self.r)
```

Finally, within a **VoteSimulation**, all these entities are instantiated and perform their protocol steps in order. The following example illustrates the vote casting phase:

```python
votingClients = [VotingClient(i, self.voters[i], self.rawSheetData[i], self.bulletinBoard) for i in range(len(self.voters))]
for votingClient in votingClients:
    # Get selection (protocol 6.4)
    s = votingClient.candidateSelection(autoInput, self.secparams)

    # Generate ballot & send oblivious transfer query (protocol 6.5)
    (ballot, r) = votingClient.castVote(s, autoInput, self.secparams)

    # Generate oblivious transfer response & check ballot (protocol 6.5)
    responses = [(authority.name, authority.runCheckBallot(votingClient.i, ballot, self.secparams)) for authority in self.authorities]
    for res in responses: print("Ballot validity checked by authority %s: %r" % (res[0], res[1]))
```

5.5 Time schedule

Project tasks		38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
Documentation																		
Update documentation and journal	soll	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦◆
	ist	🟩	🟩	🟩			🟩											
Concept																		
Working out goals / requirements	soll	🟦	🟦	🟦														
	ist	🟩	🟩	🟩	🟩	🟩												
Concept	soll		🟦	🟦														
	ist		🟩	🟩	🟩													
Implementation																		
Update CHVote cryptolibrary to the lastest specification	soll	🟦	🟦◆															
	ist	🟩	🟩															
Prototyping (proof of concept)	soll			🟦	🟦◆													
	ist			🟩	🟩													
Iteration 1																		
Implement backend API	soll					🟦	🟦	🟦	🟦	🟦	🟦							
	ist					🟩	🟩											
Implement frontend: Pre-election	soll							🟦	🟦									
	ist							🟩	🟩									
Implement frontend: Election	soll								🟦	🟦								
	ist								🟩	🟩								
Implement frontend: Post-election	soll									🟦	🟦◆							
	ist																	
Iteration 2																		
Improve backend API	soll											🟦	🟦	🟦	🟦	🟦	🟦	
	ist																	
Improve frontend and implement can-criteria: Pre-election	soll											🟦	🟦					
	ist																	
Improve frontend and implement can-criteria: Election	soll													🟦	🟦			
	ist																	
Improve frontend and implement can-criteria: Post-election	soll															🟦	🟦◆	
	ist																	
Project completion																		
Poster, Final Day, Presentation	soll																🟦	🟦
	ist																	

Table 5.1: Project time schedule

6 Journal

6.1 Week 1

6.1.1 Reflexion

During our kickoff meeting we discussed the several possibilities of our bachelor thesis based on the spadework of the previous "project 2" module and broke them down into two options: A realistic prototype of the whole chVote system that includes everything a real implementation would need, like signatures, channel security, distributed election authorities with docker etc., or an interative prototype ("demonstrator") that allows to demonstrate the functionality of the chVote protocol in a more visual manner.

6.1.2 Next steps

6.2 Week 2

6.2.1 Reflexion

The following week we have made the decision to build the demonstrator mainly because the final product would potentially be more attractive visually than a prototype where the main work is hidden behind the scences. We have also started thinking about what technologies and frameworks to use and to build a few sketches and mockups to have some basis for discussion for our next meeting. At this stage we have yet been very unsure about how the application should look like, what audience we should have as our main target and what functionality the application should offer.

During our second meeting we ellaborated the goals and some more technical details.

- In essence, the application should allow to demonstrate a chVote election from the view of every party participating in the election process.
- The system should manage multiple elections
- The application should be a realtime webapp that updates the views automatically as soon as something changes and without having to reload the page

6.2.2 Next steps

6.3 Week 4

6.3.1 Reflexion

In the third week we finished describing the goals and further worked on the system architecture. We also made some first experiences with the envisaged frameworks and technologies (VueJS, socket.io, Flask, MongoDB).

- working with socket.io and VueJS has been very intuitive and looked very promising and suitable for our project
- We were not yet sure whether or not mongoDB is the right technology for our needs.

6.3.2 Next steps

7 Conclusion

During the first few weeks we felt as if we have been thrown into cold water. Reading and understanding the protocol wasn't easy at first, because we had to get used to the notation and memorize a large amount of variables used by the many algorithms. While some of the cryptographic primitives were taught in previous courses, most of them were new and unknown to us. We focused on getting a good understanding of the protocol on a higher level rather than learning about each and every algorithm in detail, as this was sufficient for implementing and understanding the protocol.

Additionally, programming algorithms isn't something we are doing on a daily basis. Therefore, the first few algorithms took us quite some time to implement. After a few weeks, we could greatly increase our productivity and in the end, we could implement even the larger algorithms in not much more time than the simple ones in the beginning of the project.

From our perspective, the project has been extremely interesting and we are still impressed by the ideas presented and specified in the CHVote specification. From simply implementing the protocol we could learn a lot about the CHVote protocol and E-Voting in general and could improve both our knowledge of more advanced cryptographic topics and get practise in implementing cryptographic algorithms.

7.1 Python drawbacks

During the project we have experienced a few issues with the programming language that we used to implement the specification in, Python. In particular, we have observed the following issues:

- Performance issues due to Python being an interpreted language
- Function overhead: function calls in Python seem to be quite slow
- Strongly dynamic typing vs. static typing: the Python interpreter needs to inspect every single object during run time (be it an integer or a more complex object)
- The *BigInteger* library surprisingly isn't as fast as using directly the GMP library
- Larger projects tend to turn out messy
- Little to no standard documentation regarding project structure
- No real standard for unit testing, documentation generation etc.

For detailed information regarding the performance issues that we have experienced see [2] and [3]. Based on the reasons above we would not recommend to use Python for the use in similar or larger project. Python is indeed a very handy language to write quick prototypes and proof of concepts, but issues become more frequent in larger projects.

Bibliography

[1] "CHVote System Specification", by Rolf Haenni, Reto E. Koenig, Philipp Locher and Eric Dubuis, April 11, 2017.

[2] "Why Python is Slow: Looking Under the Hood", by Jake VanderPlas, see http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/

[3] "Python speed: performance tips", from the official Python wiki, see https://wiki.python.org/moin/PythonSpeed/PerformanceTips