Formally Verified Electronic Voting Scheme : A Case Study

Mukesh Tiwari

A thesis submitted for the degree of
YOUR DEGREE NAME
The Australian National University

August 2019

Except where otherwise indicated, this thesis is my own original work.

Mukesh Tiwari
6 August 2019

to my xxx, yyy (yyy is the people you want to dedicated this thesis to.)

Acknowledgments

This thesis could not have been possible without the support of my supervisor, Dirk Pattinson. I really admire his ability to understand the problem, and his intuition to to make sure that I stay clear from many dead ends which I would have happily spent months. I wish I could incorporate more of his qualities, but I believe I have less optimism about my chance.

Abstract

Put your abstract here.

Contents

Acknowledgments **vii**

Abstract **ix**

1 Introduction **1**
1.1 Why Schulze ? . . . 2
1.2 Why Coq . . . 2

2 Background **3**
2.1 Coq: Interactive Theorem prover . . . 3
2.1.1 Calculus of Inductive Construction . . . 3
2.1.2 Dependent Types . . . 4
2.1.2.1 Correct by Construction . . . 4
2.1.2.2 Types vs. Prop . . . 4
2.1.2.3 Gallina : The Specification Language . . . 4
2.1.3 Trusting Coq proofs . . . 4
2.2 Cryptography . . . 4
2.2.1 Homomorphic Encryption . . . 4
2.2.1.1 El-Gamal Encryption Scheme . . . 5
2.2.1.2 Pallier Encryption Scheme . . . 5
2.2.2 Commitment Schemes . . . 5
2.2.2.1 Hash Based Commitment Scheme . . . 5
2.2.2.2 Discrete Logarithm Based Commitment Scheme . . . 5
2.2.3 Zero Knowledge Proof . . . 5
2.2.4 Sigma Protocol : Efficient Zero Knowledge Proof . . . 5
2.3 Summary . . . 5

3 Schulze Method : Evidence Carrying Computation **9**
3.1 Schulze Algorithm . . . 9
3.1.1 An Example . . . 9
3.2 Formal Specification . . . 9
3.2.1 Scrutiny Sheet . . . 9
3.2.2 Vote Counting as Inductive Type . . . 9
3.2.3 All Schulze Election Have Winners . . . 9
3.3 Experimental Result . . . 9
3.4 Summary . . . 10

4 Homomorphic Schulze Algorithm : Axiomatic Approach 11
4.1 Verifiable Homomorphic Tallying 11
4.1.1 Ballot Representation 11
4.1.1.1 Validity of Ballots 11
4.1.2 Cryptographic Primitives 11
4.1.2.1 Construction Primitive 11
4.1.2.2 Verification Primitive 11
4.2 Realization in Theorem Prover 11
4.3 Correctness by Construction 11
4.4 Implementation : A Experience Report 12
4.4.1 Extraction 12
4.4.2 Unicrypt : Java Library for Realizing Cryptographic Primitives . 12
4.5 Experimental Result 12
4.6 summary 12

5 Scrutiny Sheet : Software Independence 13
5.1 Verification and Verifiability 13
5.2 Certificate : Ingredient for Verification 13
5.2.1 Plaintext Ballot Certificate 13
5.2.2 Encrypted Ballot Certificate 13
5.3 Proof Checker 13
5.3.1 A Verified Proof Checker : IACR 2018 13
5.4 Summary 14

6 Machine Checked Schulze Properties 15
6.1 Properties 15
6.1.1 Condercet Winner 15
6.1.1.1 Reversal Symmetry 15
6.1.2 Monotonicity 15
6.1.3 Schwartz set 15

7 Conclusion 17
7.1 Related Work 17
7.2 Future Work 17
7.2.1 Formalization of Cryptography 17
7.2.2 Integration with Web System : Helios 17

List of Figures

List of Tables

Chapter 1

Introduction

Leave this chapter till end

This place is why formalizing Electronic voting scheme ?

Before I start diving deep into explaining bits and pieces of this thesis (Formal verification of Schulze Method), I still need to provide persuasive argument that why did I choose to verifying Schulze algorithm in Coq theorem prover given the fact that Schulze is not used in any democratic election, and Coq is not serious business in development of mathematical theories or software artefact. Well, the honest answer is that I want to graduate (hopefully), but it's still not convincing argument because I could have chosen some other project and graduate. On the serious note, this thesis started as a quest to find the answer of question "Can we afford bug or bugs in software used for vote counting ?". Given that I am computer scientist by profession, I would not try to justify my decisions by excessive use of philosophical arguments, but at this point it seems very apt to first investigate this question from philosophical point.

"People shouldn't be afraid of their government. Governments should be afraid of their people." âĂŢ Alan Moore, V for Vendetta

"Those who cast the vote decides nothing. Those who count the vote decide everything." âĂŢ Joseph Stalin

"The best weapon of a dictatorship is secrecy, but the best weapon of a democracy should be the weapon of openness. " âĂŢ Niels Bohr

The answer depends on how you perceive democracy. For a dictator, probably bug in the vote software would be a natural choice, among many others, to rig the election. If you firmly believe in democracy and democratic values, then among many other things, transparency and bug freeness in vote counting software would also be in your agenda. There is no doubt that technology can play a critical role in maintaining the democratic values, but assuming that it is the only factor would be a gross mistake. In Azerbaijan's 2013 election, the running president Ilham Aliyev launched a iPhone app, to boost the credibility of election, which enabled the citizens of Azerbaijan to track the tallies as counting took place. There was just one problem. The app already showed that Ilham Aliyev elected before even a ballot was counted. In this particular case, technology merely helped in surfacing the problem, but it did not do any other thing. More often technology can be used to hide the transparency of system than making it evident specially in corrupt society for personal gain.

Democracy is a complex system of different actors interacting with each other in certain fashion. How to make these interaction more productive and better for society, I leave this to political scientists and social scientist to figure out, and I stick to my job as a technology enabler. This thesis is my journey (with my supervisor) about finding a way to make vote counting software more robust and transparent.

1.1 Why Schulze ?

Even though Schulze method is not used in any democratic election, we settled down with it because it is interesting and at the same time, non-trivial. Schulze's method [cite Schulze] elects a single winner based on preferential votes. At the same time, Arrow's impossibility theorem [cite Arrow] states that no preferential voting scheme can have all the desired properties established by social choice theorist, the Schulze's method offers a good balance. Many of these properties are already established in his original paper. These properties are Non-dictatorship, Pareto, Monotonicity, Resolvability, Independence of Clones

I will discuss some of its properties in next chapter.

A quantitative comparison of voting methods [cite An Optimal Single-Winner Preferential Voting System Based onGame Theory] also shows that Schulze voting is better (in a game theoretic sense) than other, more established, systems. The Schulze Method is rapidly gaining popularity in the open software community, and It is one of the most popular voting protocol over internet to elect candidates. At of 22 July 2019, Wikipedia entry on Schulze method [cite Wiki entry] shows at least 70 users, and some of notable users among them are Gentoo Foundation, Debian, GNU Privacy Guard (GnuPG), Ubuntu, and Pirate Party in Australia, Austria, Belgium, Brazil, Germany, Iceland, Italy, Netherlands, New Zealand, Sweden, Switzerland, and United States.

1.2 Why Coq

How many chapters you have? You may have Chapter 2, Chapter **??**, Chapter **??**, Chapter **??**, and Chapter **??**.

Chapter 2

Background

At the begging of each chapter, please introduce the motivation and high-level picture of the chapter. You also have to introduce sections in the chapter.

Write about Hilbert's idea of mathematical formalism

A proof assistant is a computer program which assists users in development of mathematical proofs. The idea of developing mathematical proofs using computer goes back to Automath (automating mathematics) [cite Automath] and LCF [cite Logic for computation] project. The Automath project (1967 until the early 80's) was initiative of De Bruijn, and the aim of the project was to develop a language for expressing mathematical theories which can be verified by aid of computer. Automath was first practical project to exploit the Curry-Howard isomorphism (proofs-as-programs and formulas-as-types) [reference here]. DeBruijn was likely unaware of this correspondence, and he almost re-invented it ([Wiki entry on Curry-Howard]). Many researchers refers Curry-Howard isomorphism as Curry-Howard-DeBruijn isomorphism. Automath project can be seen as the precursor of proof assistants NuPrl [cite here] and Coq [cite coq]. Some other notable proof assistants are LCF (Logic for Computable Functions) [cite Milner?], Mizar [cite], Nqthm/ACl2 [cite], PVS [cite], HOL (a family of tools derived from LCF theorem prover), Agda [cite], and Lean [cite].

2.1 Coq: Interactive Theorem prover

explain here a about Coq. What is Coq ?

The Coq proof assistant is an interactive theorem prover based on underlying theory of Calculus of Inductive Construction [Cite Pualine Mohring] which itself is an augmentation of Calculus of Construction [cite Huet and Coquand] with inductive data-type.

2.1.1 Calculus of Inductive Construction

Flesh out the details of Calculus of construction and Inductive construction

Write here

2.1.2 Dependent Types

2.1.2.1 Correct by Construction

Well typed program can't go wrong. Give a example of dependent type lambda calculus(Dirk's white board) Hello World is dependent type Vector

Following from previous subsection, write description about dependent types. Show that how dependent types help in correct by construction

2.1.2.2 Types vs. Prop

It's good starting point to tell the reader that we have two definitions, one in type and other in prop. Why ? Because Type computes, but it's not very intuitive for human inspection while Prop does not compute, but it's very intuitive for human inspection. We have connected that the definition expressed in Type is equivalent to Prop definition.

Combing program with proofs leads to one stop solution, mainly correct by construction

2.1.2.3 Gallina : The Specification Language

Coq provides a highly expressive specification language Gallina for development of mathematical theories and proving the theorems about these theories. Even though Gallina is very expressive, writing proofs in Gallina is very tedious and cumbersome. In order to ease the proof development, Coq also provides tactics. The user interacting with Coq applies these tactics to build the Gallina term which otherwise would be very laborious.

explain here the difference between Prop and Types. How it affects the code extraction

2.1.3 Trusting Coq proofs

The fundamental question for trusting the Coq proofs is two fold: i) is the logic (CIC) sound ?, ii) is the implementation correct ?. The logic has already been reviewed by many peers and proved correct using some meta-logic. The Coq implementation itself can be partitioned into two parts: i) Validity Checker (Small kernel), ii) Tactic language to build the proofs. We lay our trust in validity checker, because it's small kernel. If there is bug in tactic language which often is the case then build proof would not pass the validity checker.

Try to write here how Fuzzer failed to find bugs in Compcert.

2.2 Cryptography

Write some basic crypto stuff

2.2.1 Homomorphic Encryption

Add the details of homomorphic encryption

2.2.1.1 El-Gamal Encryption Scheme

Give both additive and multiplicative

2.2.1.2 Pallier Encryption Scheme

Write some description

2.2.2 Commitment Schemes

2.2.2.1 Hash Based Commitment Scheme

2.2.2.2 Discrete Logarithm Based Commitment Scheme

Pedersen's Commitment Scheme

2.2.3 Zero Knowledge Proof

Details from

2.2.4 Sigma Protocol : Efficient Zero Knowledge Proof

2.3 Summary

Now I give a small example which defines natural number, addition of two natural numbers, and proof that addition over natural number is commutative. We can define natural number in Coq using inductive data type (listing 1.1), addition of the natural numbers (listing 1.2), and proof that addition of natural numbers is commutative written in Gallina (listing 1.3).

```
Inductive Natural : Type :=
 | O : Natural
 | Succ : Natural -> Natural
```

Listing 2.1: Inductive Data Type for Natural Numbers

More precisely, the interpretation is that Natural is a inductive type with two constructors: i) O representing zero, and ii) Succ representing successor which takes a Natural number and gives next Natural number.

Change the Addition into infix symbol + and use + in proofs. It will convey the idea more clearly

```
Fixpoint Addition (n m : Natural) : Natural :=
  match n with
  | O => m
  | Succ n' => Succ (Addition n' m)
  end.
```

```
(* Notation for Addition. Now we can use + instead of
   writing Addition *)
Notation "x␣+␣y" := (Addition x y)
            (at level 50, left associativity).
```

Listing 2.2: Addition function for Natural Numbers

We define the addition by pattern matching on first argument **n**. When **n** is O (zero), then we sum is **m**, and if **n** is **Succ n'**, then sum is successor of **n' + m**.

```
Theorem Addition_by_zero : forall (n : Natural), n + O = n.
  refine (fix IHa (n : Natural) : n + O = n :=
            match n as nz return (nz + O = nz) with
            | O => eq_refl
            | Succ n' =>
              let IHn' := IHa n' in
              eq_ind_r (fun m => Succ m = Succ n') eq_refl IHn'
            end).
Qed.


Lemma Successor_addition : forall (n m : Natural),
    Succ (n + m) = n + (Succ m).
  refine
    (fix IHn (n : Natural) : forall m : Natural,
        Succ (n + m) = n + (Succ m) :=
       match n as nz return (forall m : Natural,
                                 Succ (nz + m) =
                                 nz + (Succ m)) with
       | O => fun m : Natural => eq_refl
       | Succ n' =>
         fun m : Natural =>
           eq_ind (Succ (n' + m))
                  (fun t => Succ (Succ (n' + m)) = Succ t)
                  eq_refl (n' + (Succ m)) (IHn n' m)
       end).
Qed.


Theorem Addition_is_commutative :
  forall (n m : Natural), n +  m = m + n.
  refine
    (fix IHn (n : Natural) : forall m : Natural,
```

```
         n + m = m + n :=
        match n as nz return (forall m : Natural,
                                 nz + m =
                                 m + nz) with
        | O => fun m : Natural => eq_ind_r (fun t => m = t)
                                       eq_refl
                                       (Addition_by_zero m)
        | Succ n' =>
          fun m  =>
            eq_ind (Succ (m + n'))
                   (fun t => Succ (n' + m) = t)
                   (eq_ind_r (fun t => Succ t = Succ (m + n'))
                             eq_refl (IHn n' m))
                   (m + (Succ n'))
                   (Successor_addition m n')
        end).
Qed.
```

Listing 2.3: Addition function for Natural Numbers

We need two additional Lemma: i) *Addition_by_zero*, a proof of n + 0 = 0, and ii) *Successor_addition*, a proof of Succ (n + m) = n + (Succ m) to prove that addition on Natural is commutative (*Addition_is_commutative*),

One thing that can't escape from the reader's eyes is that the proofs written in Gallina is verbose, and they don't appear anywhere compared to a proof that would have been written by a mathematician. Well, we can lift the burden of verbosity by using tactics provided by Coq; however, there is no universally accepted solution in Coq community for second problem. There has been some research in declarative style proof writing [1], but it is not widely practised in Coq community. The proof of addition on Natural is commutative re-written using tactics (Listing 1.4)

```
Lemma Addition_by_zero : forall (n : Natural), n + O = n.
  induction n; cbn; [auto | rewrite IHn; auto].
Qed.


Lemma Successor_addition : forall (n m : Natural),
    Succ (n + m) = n + (Succ m).
Proof.
  induction n; intros m;
    cbn; [auto | rewrite <- IHn; auto].
Qed.
```

[1]www-verimag.imag.fr/ corbinea/ftp/publis/bricks-poster.pdf

```
Theorem Addition_is_commutative :
  forall (n m : Natural), n +  m = m + n.
Proof.
  induction n; intro m;
    [rewrite Addition_by_zero |
     rewrite <- Successor_addition;
     rewrite <- IHn]; auto.
Qed.
```

Listing 2.4: Addition function for Natural Numbers

Section **??** xxxx.

Section **??** yyyy.

Chapter 3

Schulze Method : Evidence Carrying Computation

Same as the last chapter, introduce the motivation and the high-level picture to readers, and introduce the sections in this chapter.

3.1 Schulze Algorithm

Write Schulze Algorithm in general

3.1.1 An Example

3.2 Formal Specification

Copy paste here ITP section 2

3.2.1 Scrutiny Sheet

paste ITP section 3

3.2.2 Vote Counting as Inductive Type

paste ITP section 4

3.2.3 All Schulze Election Have Winners

paste ITP section 5

3.3 Experimental Result

It's slow.

3.4 Summary

Same as the last chapter, summary what you discussed in this chapter and be the bridge to next chapter. Start a story here about how scrutiny sheet can help in auditing the election.

Chapter 4

Homomorphic Schulze Algorithm : Axiomatic Approach

Same as the last chapter, introduce the motivation and the high-level picture to readers, and introduce the sections in this chapter.

The problem with the previous methods is it's clearly exposes the ballots in plaintext possible leading to coercion.

4.1 Verifiable Homomorphic Tallying

4.1.1 Ballot Representation

Argue here why do we changed the ballot representation to encrypted matrix

4.1.1.1 Validity of Ballots

4.1.2 Cryptographic Primitives

4.1.2.1 Construction Primitive

Write here about how you construct the data

4.1.2.2 Verification Primitive

Write here about how you verify the constructed data

4.2 Realization in Theorem Prover

Discuss the primitives and axioms

4.3 Correctness by Construction

Describe inductive data type and assertion in the system to make sure you can not construct any illegal term

4.4 Implementation : A Experience Report

4.4.1 Extraction

4.4.2 Unicrypt : Java Library for Realizing Cryptographic Primitives

4.5 Experimental Result

Here goes the result and certificate Also point here the weakness of the system is

4.6 summary

Discuss here the weakness of system and pave the path to next chapter that how software independence can help in verifying the election

Chapter 5

Scrutiny Sheet : Software Independence

Same as the last chapter, introduce the motivation and the high-level picture to readers, and introduce the sections in this chapter.

One of the crucial aspect in electronic voting is ability to detect change in outcome of election because of software bugs.

A voting system is software independent if an undetected change or error in its software can not cause an undetected change or error in an election outcome.

5.1 Verification and Verifiability

Paste Evote section 2

5.2 Certificate : Ingredient for Verification

There are two notion of verification. Software verification and election verification. A electronic voting scheme implemented in Coq is verified implementation, but it does not imply that method itself is verifiable. Explain the software independence

5.2.1 Plaintext Ballot Certificate

Flesh out the details needed here for writing proof checker

5.2.2 Encrypted Ballot Certificate

Flesh out the details needed here for writing proof checker

5.3 Proof Checker

Write the details of proof checker for both certificates

5.3.1 A Verified Proof Checker : IACR 2018

Write some details about proof checker.

5.4 Summary

Write some advantage of proof checker for certificates. To create the mass scrutineers, all we need is a simple proof checker which would take proof certificate as input and spit true or false. If it's true then we accept the outcome of election otherwise something wrong.

Chapter 6

Machine Checked Schulze Properties

Not planned but hopefully it will done

6.1 Properties

List some properties which it follows with pictures ?

6.1.1 Condercet Winner

6.1.1.1 Reversal Symmetry

6.1.2 Monotonicity

6.1.3 Schwartz set

Chapter 7

Conclusion

Same as the last chapter, introduce the motivation and the high-level picture to readers, and introduce the sections in this chapter.

7.1 Related Work

Here goes the details for related work

7.2 Future Work

Possibility of future work

7.2.1 Formalization of Cryptography

7.2.2 Integration with Web System : Helios