



Norwegian University of
Science and Technology

Realizing Secure Multiparty Computations

Håvard Vegge

Master of Science in Communication Technology

Submission date: June 2009

Supervisor: Stig Frode Mjølsnes, ITEM

Co-supervisor: Tord I. Reistad, ITEM

Problem Description

A multiparty computation is where three or more parties compute a commonly agreed function with secret input and public output. Using this technique, virtually any cryptographic protocol problem involving a trusted party can be solved without the need for such a party. Typical problems are electronic voting, various types of auctions, privacy-preserving benchmarking, etc. Although there exists a lot of research in this field, and multiparty computations are said to have great potential, very few practical applications seem to have been developed.

This master thesis work will be to understand the basic theory of multiparty computations, and select some interesting multiparty computation problem to implement and do experiments. The aim will be to create a practical and useful program, and possibly set it up as a web application. The implementation will be based on VIFF (viff.dk), a framework for creating efficient and secure multiparty computations.

Assignment given: 15. January 2009
Supervisor: Stig Frode Mjølunes, ITEM

Abstract

The general theory of multiparty computation (MPC) was founded in the late 80-ties. Since then the concept has evolved in several ways in different scientific papers. Much of the work has been put into making MPC faster, as the first protocols were quite inefficient. Generally, MPC has been predicted a great future potential as it is possible to simulate a trusted third party, but without actually using one. Despite this very nice property, virtually no practical applications have been implemented.

Recently, a new trend within the field of MPC has emerged. Several research projects have turned their focus towards building frameworks that encourage implementation of the so far theoretical protocols and applications. The Virtual Ideal Functionality Framework (VIFF) has been chosen for this master's thesis with the goal of creating visually and user-friendly applications.

Two applications were implemented, both with focus on how MPC can provide practical solutions to various voting problems. The first application was a tool for scientists to decide who to be the 1th author on some scientific paper. This solution had several problems, including scalability, and thus led to the development of another application. A secure web voting scheme was set up, now allowing participants to visit a normal web page and vote through their browser.

The methodology of using VIFF proved effective for creating MPC applications. Although the applications are mainly proof-of-concept, they show that multiparty computations not only have a great potential, but in practice are able to solve various interesting problems even today.

Preface

This report serves as a master's thesis in Information Security in the 10th semester of the Master's Programme in Communication Technology at The Norwegian University of Science and Technology (NTNU). The assignment was given by PhD student Tord I. Reistad and Professor Stig F. Mjølsnes at the Department of Telematics.

This thesis has been a challenging and inspirational task in an emerging field of cryptology. Working with the Virtual Ideal Functionality Framework (VIFF), which very few others have used, has been motivating and exciting. I really enjoyed the practical aspect of the work.

I would like to thank my supervisor Tord I. Reistad for his valuable input and frequent feedback. His enthusiasm and great mathematical skills have been an inspiration throughout the thesis. Also thanks to Stig F. Mjølsnes, my professor, for helpful comments and constructive feedback.

Finally I would like to thank Atle Mauland for the cooperation regarding the mathematics of MPC addition and multiplication in Section 3.4.3.

Trondheim, June 11, 2009

Håvard Vegge

Abbreviations

CSS	Cascading Style Sheets
DRE	Direct Recording Electronic
GF	Galois Field
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IF	Ideal Functionality
IP	Internet Protocol
MPC	Multipart Computation
PHP	Hypertext Preprocessor
PRNG	Pseudo-Random Number Generator
RPC	Remote Procedure Call
SSL	Secure Sockets Layer
TTP	Trusted Third Party
URL	Uniform Resource Locator
VIFF	Virtual Ideal Functionality Framework
XML	Extensible Markup Language

Contents

Abstract	i
Preface	iii
Abbreviations	v
List of Figures	xi
List of Tables	xiii
List of Listings	xv
1 Introduction	1
1.1 Secure Multiparty Computation	1
1.2 Related Work	2
1.3 Motivation	3
1.4 Objectives	3
1.5 Limitations	4
1.6 Method	4
1.7 Document Structure	5
2 Secret Sharing	7
2.1 Secret Splitting	7
2.2 Additive Secret Sharing	8
2.2.1 Creating the Shares	8
2.2.2 Reconstructing the Secret	9
2.3 Shamir Secret Sharing Scheme	9
2.3.1 Overview	9
2.3.2 Finite Fields	11
2.3.3 Creating the Shares	11
2.3.4 Reconstructing the Secret	12

Contents

3	Secure Multiparty Computation	13
3.1	Adversaries	13
3.2	Network Assumptions	14
3.3	Security	14
3.4	Computation Stages	15
3.4.1	The Input Stage	16
3.4.2	The Final Stage	16
3.4.3	The Computation Stage	16
3.4.4	Multiplication Example	19
3.5	MPC Frameworks	21
4	Virtual Ideal Functionality Framework	23
4.1	History	23
4.2	Overview	23
4.3	Features	24
4.4	Architecture	24
4.4.1	Runtime	25
4.4.2	Finite Fields	26
4.5	Asynchronous Design	26
4.5.1	Expression Tree	26
4.5.2	Twisted	27
4.6	Example VIFF Program	28
4.7	Security Assumptions	31
4.8	Benchmarking	31
5	Voting	35
5.1	Overview	35
5.2	Motivation	36
5.3	The Voting Process	36
5.4	Challenges	37
5.5	Security Requirements	37
6	Choice of Applications	39
6.1	Motivation and Properties	39
6.2	The Author Application	40
6.3	The Voting Application	40
7	Application 1: Rank the Authors	41
7.1	Design and Implementation	41
7.1.1	Screen Shots	41
7.1.2	Architecture	44
7.1.3	Libraries	45
7.1.4	Code	45
7.2	Benchmarks	47

7.3	Possible Improvements	49
7.3.1	Calculation of Points	49
7.3.2	Complicated Application Launch	49
8	Application 2: Secure Web Voting	51
8.1	Design and Implementation	51
8.1.1	Screen Shots	52
8.2	Architecture	56
8.2.1	Web Server	56
8.2.2	Computation Servers	60
8.3	Libraries	60
8.3.1	SecureRandom	60
8.3.2	XML-RPC	61
8.4	Security Analysis	62
8.4.1	Voter Privacy	62
8.4.2	Eligibility	62
8.4.3	Uniqueness	63
8.4.4	Fairness	63
8.4.5	Uncoercibility	64
8.4.6	Receipt-freeness	64
8.4.7	Accuracy	64
8.4.8	Individual Vote Check	65
8.5	Possible Improvements	65
9	Discussion	67
9.1	Secure Web Voting Application	67
9.1.1	Location of the Computation Servers	67
9.1.2	Storing of Shares	68
9.2	VIFF	68
9.2.1	Necessary Steps	69
9.2.2	Development	69
9.2.3	Required Background Theory	69
9.2.4	Future Potential	70
9.3	The Potential of Multiparty Computations	70
9.4	Further Work	71
9.4.1	Other Applications	71
9.4.2	Improve and Deploy a Web Voting Application	71
9.4.3	Large-scale E-voting	72
10	Conclusion	73
	Bibliography	75
	Web References	79

Contents

A	Multiplication Mathematics	81
A.1	Linear System Approach	81
A.2	Vandermonde Matrix	82
B	VIFF Installation Guide	85
B.1	Installation Steps	85
B.2	Troubleshooting	86
B.3	Generation of Configuration Files	86
B.4	Additional Components	86
C	Source Code Author Application	87
C.1	author.py	87
D	Source Code Web Application	95
D.1	Vote.java	95
D.2	server1.py	100
E	Attachment/ZIP file	105
E.1	Rank the Authors	105
E.2	Secure Web Voting	105
E.3	Secure Position Determination	106

List of Figures

1.1	A trusted party scenario to the left and the idea of MPC (simulation of a trusted party) to the right.	1
2.1	The linear curve corresponding to a (2,n)-threshold scheme [5].	10
2.2	The quadratic curve corresponding to a (3,n)-threshold scheme [5].	10
3.1	Illustration of how $f(x)$, $g(x)$ and $h(x)$ can be added together and form a new polynomial marked <i>total</i>	18
3.2	Graph with the points (1,4), (2,2) and (3,0) which implies that the secret is 6 (the line intersects the y-axis).	20
4.1	The protocol stack which VIFF is built upon.	24
4.2	Relations between class instances at runtime [GDP09].	25
4.3	Expression tree [DGKN08].	27
4.4	Parallel multiplication benchmarks, average time per multiplication [Gei08].	32
4.5	Parallel comparison benchmarks, average time per comparison [Gei08].	32
7.1	The main window of the author application. Atle cannot vote for himself.	42
7.2	Atle tries to give all the votes for Håvard, but gets an error message.	42
7.3	Atle has voted and 25% of the players have given their inputs.	43
7.4	The result (order of authors) is securely calculated. Nothing else is revealed.	43
7.5	Player 1, 2 and 3 are sharing x, y and z [6].	45
7.6	Actual number of comparisons, the upper bound growth rate, and the average computation time used are plotted for various values of n	48
8.1	The architecture of the web application.	51
8.2	Step 1 for creating a poll on the web page.	53

List of Figures

8.3	Step 2 for creating a poll on the web page.	54
8.4	An example e-mail received by a person participating in a vote.	54
8.5	Java applet for delivering of a vote.	55
8.6	The console of one computation server after it has calculated the result of a vote.	56
8.7	Classes and files of the web application.	56
8.8	The three database tables where information about polls, users and results are stored.	59
8.9	The MySQL table <i>result</i> , containing the encrypted shares, as shown in phpMyAdmin.	60
8.10	XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism [10].	61
9.1	Web application scheme which stores the shares at different locations, that is, at P1, P2 and P3.	68

List of Tables

3.1	The threshold obtainable for various qualities of security, where all results are for adaptive security.	15
3.2	Example matrix for secret shared multiplication.	20
3.3	The players' shares of the total polynomial.	20
5.1	Classification of voting types.	36
7.1	List of files required for executing the author program.	44
7.2	Benchmark results for the author application.	48
8.1	List of PHP files in the secure web voting application.	57

List of Tables

List of Listings

4.1	The use of callbacks in Twisted.	27
4.2	A simple VIFF example program.	29
4.3	VIFF configuration file for player 1. The pubkey and seckey have been abbreviated.	30
7.1	Distribution of points for the votes given in the author application.	46
7.2	The votes/points are given as inputs to the shamir share function.	46
7.3	The points, represented as shares, are summed up.	47
8.1	The essential part of the makesShares function in the Java applet.	58
8.2	The RSA encryption of shares in the Java applet. The public keys have been abbreviated.	58
8.3	Java code demonstrating how to create random integers.	61
8.4	The XML-RPC request message sent from the web server to computation server 1.	62
C.1	Source Code author.py	87
D.1	Source Code Vote.java	95
D.2	Source Code server1.py	100

List of Listings

Chapter 1

Introduction

A multiparty computation (MPC) is where three or more parties compute a commonly agreed function, but with the restriction that none of the participants can learn anything more than their own input and the public output. 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?”

1.1 Secure Multiparty Computation

In a multiparty computation (MPC) there are three or more distrusting parties. With the assistance of a (fictive) trusted party, they can securely evaluate functions of their inputs. There exist of course no trusted party, but a multiparty computation will be equivalent to one. Figure 1.1 illustrates the idea of simulation.

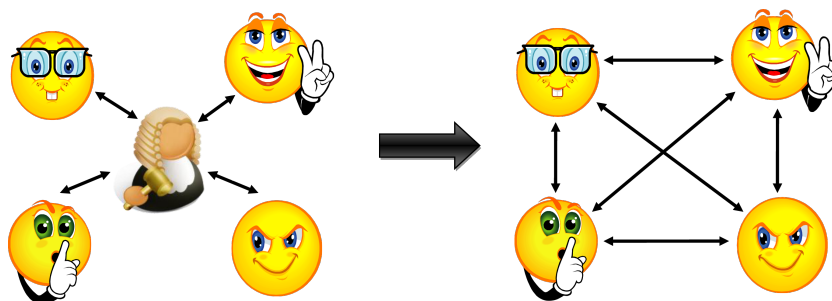


Figure 1.1: A trusted party scenario to the left and the idea of MPC (simulation of a trusted party) to the right.

Chapter 1. Introduction

To the left in the figure is a judge who all participants trust to give their secret inputs. This trusted party can then compute the outcome of the process (for instance the sum of all inputs) and reveal the output. To the right in the figure is how the equivalent MPC would work. 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. This is what MPC is all about: finding cryptographic protocols that can be used to simulate (and hence replace) trusted parties [Opp05].

Besides determining who is the richer of some number of people, multiparty computation has many other practical applications. Some of the most well-known are listed below:

- Electronic voting
- Elections over the Internet
- Private bidding and auctions
- Privacy-preserving data mining
- Privacy-preserving database query
- Sharing of signature or decryption functions

1.2 Related Work

The problem of multiparty computation was first introduced by Yao [Yao82]. Following the results of Ben-Or, Goldwasser and Widgerson [BOGW88] and also Chaum, Crépeau and Damgård [CCD88] any given function may be computed securely if $t < n/3$, where n is the number of players and t is the number of corrupted players. If the adversary is passive, the bound is $t < n/2$. Goldreich, Micali and Widgerson [GMW87] presented a protocol that tolerated an active adversary corrupting any $t < n/2$ of the players under computational assumptions.

An essential part of multiparty computations is secret sharing. This method for distributing a secret amongst a group of participants by allocating *shares* of the secret, was invented by both Adi Shamir [Sha79] and George Blakley [Bla79] independently in 1979. The shamir module in VIFF is obviously based on Shamir's paper.

Most importantly MPC provide solutions to various problems such as distributed voting, private auctions, etc. Although having a great potential,

very few practical applications are known to be in use. One example, however, is a large-scale MPC application used for trading contracts in the Danish sugar beet market [BCD⁺08].

1.3 Motivation

As already mentioned, multiparty computations are said to have great potential, but very few practical applications seem to have been developed. This can partly be explained by the lack of or slow protocols for doing such computations. Several MPC frameworks have been developed in recent years and by using the Virtual Ideal Functionality Framework (VIFF) it should be fairly easy to develop applications that use multiparty computations.

Multiparty computation is a powerful tool, but not widely used in practice to solve cryptographic protocol problems. Recently I have come over a few simple problems that I have thought would be interesting to implement using MPC:

- A project I wrote in cooperation with SINTEF ICT in the second half of 2008 resulted in a scientific paper submitted to the conference ICIMP 2009¹. In total we were five persons that had contributed to the paper and we had to decide the order of the authors. Nobody dared to express their preferences loudly, and the decision was finally made by a simple and unintelligent method of drawing lots.
- A blog post by the main author behind VIFF, Martin Geisler, suggested a list of names as candidates for the new framework in late 2007 [13]. Anyone could vote by leaving a comment. One of the users then replied with the following amusing line:

“Is it possible to vote without revealing any information?
...and if so, can you prove it safe to do so?”

1.4 Objectives

This project consists of two main parts. First a theoretical background study in the field of secure multiparty computation is to be performed. Secondly, several applications will be implemented using MPC and VIFF. In general, this master’s thesis aspires to achieve the following objectives:

- Describe the basic theory of multiparty computations. The aim is to describe the concept clearly and with examples such that one understands how secret sharing and multiparty computations really works.

¹The Fourth International Conference on Internet Monitoring and Protection, ICIMP 2009, in Venice, Italy.

Chapter 1. Introduction

- Give a thorough introduction to VIFF and show what steps are necessary to create applications with this framework.
- Create one or more practical and useful programs, and possibly set it up as a web application.
- Evaluate VIFF and its future potential.

1.5 Limitations

The following limitations have been identified:

- Multiparty computation is a broad term and a lot of different protocols are described in scientific papers. This thesis is focused towards the main concepts needed to understand how MPC and programming in VIFF works.
- The two applications developed throughout this master's thesis are basically just proof-of-concept applications. It would have been interesting to do a more comprehensive analysis of scalability, but this has been considered out of the scope.

1.6 Method

The method used throughout this thesis can be divided into three main phases:

- Systematic study of scientific research, both on MPC in general and on applications of MPC.
- Experimenting with the possibilities of VIFF and creation of a simple test program.
- Design and implementation of one or more applications.

The most extensive phase was to design and implement the applications. The first application, *Rank the Authors* in Chapter 7, was focusing on providing a user-friendly GUI. Although fulfilling this property, the evaluation clearly showed aspects that could be improved. These aspects were transformed into new properties that the next MPC application should possess.

The *Secure Web Voting* application in Chapter 8 was then designed and implemented using a totally different and more complete scheme. It is a more general voting application available from the world wide web, making it both user-friendly, easy to understand and accessible as long as one has

an Internet connection.

For the actual implementation of code in the MPC programs, an *incremental and iterative development* process was chosen [Coc08]. The alternative strategy would have been to plan everything right the first time and develop the entire system with a “big-bang” at the end. This would not have been optimal as the time to create applications with the new VIFF framework was very uncertain. By using an incremental and iterative process, it was possible to add more functionality and requirements to the application during the implementation.

1.7 Document Structure

The remainder of this report is organized as follows:

- **Chapter 2: Secret Sharing**
This chapter presents theory related to secret sharing which is an essential part of multiparty computations.
- **Chapter 3: Secure Multiparty Computation**
This chapter presents multiparty computation more thoroughly and includes detailed description of how addition and multiplication are done in MPC.
- **Chapter 4: VIFF**
The chapter about VIFF describes the features and mode of operation of the framework. A simple example program is also included.
- **Chapter 5: Voting**
This chapter reviews basic theory related to voting as the two implemented applications are voting programs.
- **Chapter 6: Choice of Applications**
This chapter describes the high-level motivation for the two implemented applications.
- **Chapter 7: Application 1: Rank the Authors**
This chapter is a review of the first application; a GUI application which lets participants of a scientific paper vote on their choice of 1th, 2th, 3th, etc. author.
- **Chapter 8: Application 2: Secure Web Voting**
This chapter describes a web application for secure voting. The system requires a more extensive setup and includes a web page (HTML and PHP), a database (MySQL) together with three computation servers (VIFF).

Chapter 1. Introduction

- **Chapter 9: Discussion**

This chapter evaluates the two applications and VIFF as a framework. Finally, some ideas for future work are presented.

- **Chapter 10: Conclusion**

The last chapter summarizes the major results and experiences, and concludes the thesis.

Additionally, the following appendices are included:

- **Appendix A: Multiplication Mathematics**

The first appendix contains detailed mathematics required when performing multiplication of shares in a MPC protocol.

- **Appendix B: VIFF Installation Guide**

This appendix explains the installation steps required for setting up VIFF on MS Windows.

- **Appendix C: Source Code Author Application**

This appendix shows the source code of the author program.

- **Appendix D: Source Code Web Application**

This appendix shows the source code of two essential files of the web voting application, namely the Java applet and one of the computation servers.

- **Appendix E: Attachment/ZIP file**

This appendix lists the contents of the attached ZIP file.

Chapter 2

Secret Sharing

In this chapter theory related to secret sharing is presented. Secret sharing is closely related to MPC and a few such schemes are thus described. Shamir secret sharing will be the most important as it is used in VIFF.

Imagine setting up a launch program for a nuclear missile. You do not want to give a single person this responsibility. In order to launch the missile it is desirable that two people will need to enter a password and turn their keys at the same time. This example is comparable to the concept of secret sharing. More formally:

Definition 1 *A secret sharing scheme refers to a method which distributes shares of a secret among a group of participants in such a way that the secret can only be reconstructed when the shares are combined together.*

2.1 Secret Splitting

A special case of secret sharing is in some literature referred to as *secret splitting*. In its simplest form this scheme is flawed, but it clearly shows what to avoid.

Consider the naive scheme of splitting the secret phrase *password* between two parties. The secret phrase consists of 8 characters, each which can be selected from a set of 100 possible characters, thus the number of possible passwords are 100^8 . Assuming the generation and check of one such password takes 1 microsecond, it would take $100^8 \cdot 10^{-6}$ seconds ≈ 300 years to check them all. On average it would only take half the time, but over 100 years is still quite some time. Now assuming an adversary has one share, that is, four of the eight characters are known. This would reduce the problem to 100^4 possible passwords to check against. Generation and checking of all passwords would now only take $100^4 \cdot 10^{-6} = 100$ seconds.

Chapter 2. Secret Sharing

It is obvious that this scheme cause partial information disclosure [5]. To avoid such problems, two requirements common to all unconditionally secure secret sharing schemes can be set up:

- Each share of the secret must be at least as large as the secret itself. This way, no information about the secret can be determined, if one has less than threshold t shares.
- All secret sharing schemes must use random bits.

2.2 Additive Secret Sharing

There exist several secret sharing schemes which differ in theory and application. The most trivial ones require all n participants in order to reconstruct the secret. Additive secret sharing is an example of such a scheme and is described next.

Assume we wish to split the secret s among n people, then $n - 1$ random numbers r_1, \dots, r_{n-1} need to be selected. It is impossible to choose a random integer in a way that all integers are equally likely (the sum of the infinitely many equal probabilities, one for each integer, cannot be 1). Therefore an integer p larger than all possible messages that might occur is selected. Then s and r are regarded as numbers mod p [TW06]. The share for player P_n is computed:

$$s_n = s - \sum_{i=1}^{n-1} r_i \text{ mod } p$$

The rest of the players $P_i, 1 \leq i \leq n - 1$, get the shares $s_i = r_i$. In order to reconstruct the secret, the sum of all shares needs to be computed:

$$s = \sum_{i=1}^n s_i \text{ mod } p$$

The above technique is absolutely secure if done properly. Each piece by itself is totally worthless, but together the players can reconstruct the secret. A scheme characterized by this property is also the definition of a *perfect secret sharing scheme*.

2.2.1 Creating the Shares

Below is a simple example to better understand how additive secret sharing really works. Assume we have a *secret* = 13 which should be secret shared among $n = 3$ participants. We define an integer $p = 20$ to work over, together with $n - 1$ random numbers $r_1 = 3$ and $r_2 = 8$. The following calculations are then possible:

2.3. Shamir Secret Sharing Scheme

$$\begin{aligned}s_1 &= r_1 \bmod p = 3 \bmod 20 = 3 \\s_2 &= r_2 \bmod p = 8 \bmod 20 = 8 \\s_3 &= s - \sum_{i=1}^{n-1} r_i \bmod p = 13 - (3 + 8) \bmod 20 = 2\end{aligned}$$

2.2.2 Reconstructing the Secret

Reconstructing the secret is easy, just add all shares together and we see that the sum 13 is equal to the original secret.

$$s = \sum_{i=1}^n s_i \bmod p = 3 + 8 + 2 \bmod p = 13$$

One problem with this protocol is that with a missing share, no one can reproduce the secret. Schemes that solve this issue are called threshold schemes and are described next.

2.3 Shamir Secret Sharing Scheme

In difference to additive secret sharing, threshold schemes allow a subset of the players to reconstruct the secret. More formally a (t, n) -threshold scheme is defined as follows:

Definition 2 *A (t, n) -threshold scheme is a method of sharing a message M among a set of n participants such that any subset consisting of t participants can reconstruct the message M , but no subset of smaller size can reconstruct M . [TW06]*

In 1979, both Adi Shamir [Sha79] and George Blakley [Bla79] independently presented such threshold schemes. Shamir's method is using polynomial interpolation, while Blackley works on the intersection of hyperplanes. Only Shamir's secret sharing will be described here, as this is the scheme implemented in VIFF.

2.3.1 Overview

In order to illustrate Shamir secret sharing, a $(2, n)$ scheme is designed. Assume a secret is to be shared among n participants. In Figure 2.1 the point $(0, s)$ on the y -axis, which corresponds to the secret, is selected. Drawing a random line through this point gives n points on that line. Each point $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ represents a share.

Since two points determine a line, it is clear that two participants can discover the secret by drawing a line between their points, and from there check

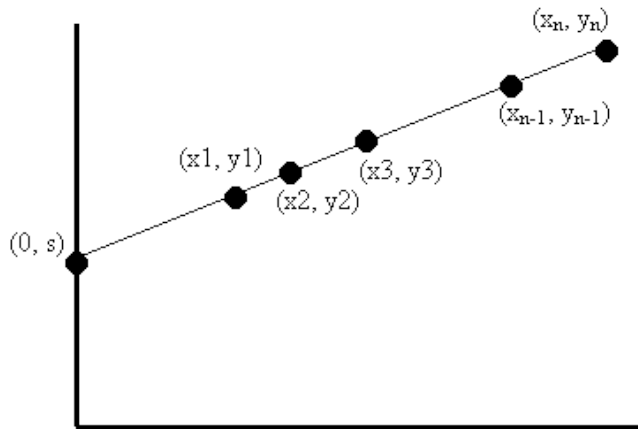


Figure 2.1: The linear curve corresponding to a $(2, n)$ -threshold scheme [5].

where the line intersects the y axis. If holding only one share, infinite possibilities would exist for a line through this point. This means that with one point, no information about the secret is revealed.

Extending the idea to a $(3, n)$ scheme, a curve determined by three points is needed. This corresponds to the quadratic function $y = a_0 + a_1 \cdot x + a_2 \cdot x^2$. Again the secret is located on the y-axis before drawing a random curve corresponding to a quadratic function that goes through the point. Finally n points are selected as seen in Figure 2.2 which all represent a share.

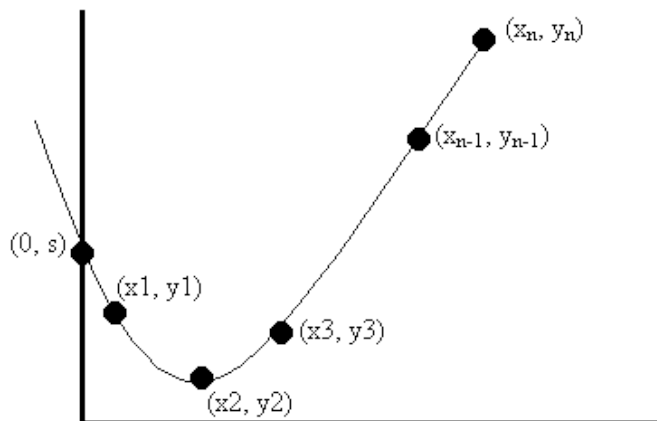


Figure 2.2: The quadratic curve corresponding to a $(3, n)$ -threshold scheme [5].

2.3.2 Finite Fields

Finite fields are used in Shamir secret sharing and in VIFF, thus a brief description is given. A finite field, also called a *Galois field*, is a field with a finite number of elements. The order of a finite field is always a prime or a power of a prime.

$GF(p)$, where p is a prime number, is simply the ring of integers modulo p . This means one can perform operations (addition, subtraction, multiplication) as usually done on integers, followed by reduction modulo p . For instance, in $GF(7)$, $5 + 4 = 9$ is reduced to 2 modulo 7.

The reason for working modulo p is to achieve *perfect* secrecy, that is, given fewer than threshold t shares, the secret can be any element in the field and thus those shares do not supply any additional information about the secret. Simply working over all the reals makes it impossible to state the same qualities, because the shares cannot be uniformly distributed over the reals.

2.3.3 Creating the Shares

The essential idea of Shamir is that two points are sufficient to define a line, three points to define a parabola, etc. To define a polynomial of degree $t - 1$, it then requires t points:

$$f(x) = s + r_1x + \dots + r_{t-1}x^{t-1} \pmod{p}$$

The prime p must be larger than all possible messages and also larger than the number of n players. The coefficients r_1, \dots, r_{t-1} in $f(x)$ are randomly chosen from a uniform distribution over the integers in $[0, p)$.

To illustrate this mathematically, suppose that we have a secret 1337 ($s = 1337$). It is desirable to divide this secret into 6 shares ($n = 6$) and where any subset of 3 players ($t = 3$) can reconstruct the secret. As $t = 3$, we select $t - 1 = 2$ random integers $r_1 = 122$ and $r_2 = 51$. The polynomial is then:

$$f(x) = 1337 + 122x + 51x^2$$

and the shares can be calculated as shown below. Note, however, that we assume to be working over a very large field in the following calculations and thus the problem of overflow is avoided.

Chapter 2. Secret Sharing

$$\begin{aligned}s_1 &= f(1) = 1337 + 122 \cdot 1 + 51 \cdot 1^2 = 1510 \\s_2 &= f(2) = 1337 + 122 \cdot 2 + 51 \cdot 2^2 = 1785 \\s_3 &= f(3) = 1337 + 122 \cdot 3 + 51 \cdot 3^2 = 2162 \\s_4 &= f(4) = 2641 \\s_5 &= f(5) = 3222 \\s_6 &= f(6) = 3905\end{aligned}$$

2.3.4 Reconstructing the Secret

In order to reconstruct the polynomial and hence the secret message, we will use interpolation. The requirement is that we obtain t of the polynomial's values $(x, f(x) = s_x)$. According to Lagrange's formula, the following expression gives the polynomial $f(x)$:

$$f(x) = \sum_{i=1}^t s_i \prod_{\substack{j=1 \\ j \neq i}}^t \frac{x - x_j}{x_i - x_j} \pmod{p}$$

Because we want to evaluate the polynomial at $x = 0$, the equation can be further simplified. This gives the following formula to calculate the secret s :

$$s = f(0) = \sum_{i=1}^t s_i \prod_{\substack{j=1 \\ j \neq i}}^t \frac{-x_j}{x_i - x_j} \pmod{p}$$

Now suppose player 2, 4 and 5 want to collaborate to determine the secret. Using the last equation and the tuples $(2, 1785)$, $(4, 2641)$ and $(5, 3222)$ the reconstruction of the secret is possible.

$$\begin{aligned}s &= 1785 \cdot \left(\frac{-4}{2-4} \cdot \frac{-5}{2-5}\right) + 2641 \cdot \left(\frac{-2}{4-2} \cdot \frac{-5}{4-5}\right) + 3222 \cdot \left(\frac{-2}{5-2} \cdot \frac{-4}{5-4}\right) \\&= 5950 - 13205 + 8592 \\&= 1337\end{aligned}$$

Chapter 3

Secure Multiparty Computation

This chapter addresses secure multiparty computation (MPC) which is the problem of how mutually distrusting parties can compute a function without revealing their individual input values to each other. More formally the problem of multiparty computation is defined as follows [Dam06]:

Definition 3 *Secure multiparty computation (MPC) can be defined as the problem where n participants P_1, \dots, P_n agree on a function f and wish to compute and reveal to each participant $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$, where each input x_i is a secret input provided by participant P_i . Player P_i will only learn y_i , nothing more.*

Security in this context means guaranteeing the correctness of the output as well as the privacy of the participants' inputs, even if some participants try to cheat.

3.1 Adversaries

Participants in a MPC protocol do not necessarily behave as intended. Corrupt players may occur and they are divided into the following two main groups [Opp05]:

- A *passive adversary* is where players perform the protocol correctly, but collaborate in information gathering and sharing with the goal of getting access to sensitive information of other players. Such players are sometimes called *semihonest*.
- An *active adversary* is a group of players deviating from the protocol in order to disrupt the computation. The goal is to produce incorrect results and/or violate the privacy of other players. Such players are sometimes called *byzantine*.

Chapter 3. Secure Multiparty Computation

Both types can be *static* or *adaptive*. A *static adversary* means that the set of corrupted players is chosen before the protocol starts. An *adaptive adversary* on the other hand, can choose which players to corrupt during the execution of the protocol.

3.2 Network Assumptions

Traditionally secure multiparty computations were done with the assumption of a *synchronous* network, with a known maximum drift rate. In this setting it is easy to divide a protocol into logical units called *rounds*. In each round, each player may send a message to each other and the messages are delivered before the next round begins.

Modern networks, like the Internet, are though mostly *asynchronous*. Data is transmitted without the use of an external clock signal and problems like congestion, delay, lost packets and other transmission errors might occur. It is worth noticing that VIFF with its Twisted integration is designed to be used on asynchronous networks. Refer to Section 4.5 for a description.

In addition the players communicate with each other over channels. Many MPC protocols assume the existence of *pairwise* channels among the players. For a MPC protocol to be information-theoretically secure, it is assumed that there exists an unconditionally secure channel between each of the participants.

Other protocols assume broadcasting, which is similar to the Byzantine Agreement [LSP82]. A Byzantine fault tolerant system will be able to reach the same group decision assuming there are not too many Byzantine players (also known as active adversaries). A message “broadcast” by a faulty user will result in all honest users agreeing on a single value for that message [GL02].

3.3 Security

This section describes some classical results on security and threshold adversaries. Generally, the security can be divided into two groups [KLR06]:

1. *Information-theoretically secure MPC* is unconditionally secure, i.e., it is not possible to cheat or violate the security even if an adversary has unrestricted computing power. It can be further classified into the following levels:
 - (a) *Perfect security*: The result of a real execution of the protocol with a real adversary must be exactly the same as the result of

3.4. Computation Stages

an ideal execution with a trusted party and an ideal adversary.

- (b) *Statistical security*: The result of the real protocol execution need “only” be statistically close to the result of an ideal execution.

2. *Cryptographically secure MPC* is based on unproven cryptographic primitives that are assumed to be computationally infeasible.

As mentioned in Section 1.2 the classical results for the information-theoretic model due to Ben-Or, Goldwasser and Widgerson [BOGW88] and Chaum, Crépeau and Damgård [CCD88] state that every function can be securely computed with perfect security in presence of an adaptive, passive adversary, if the adversary corrupts less than $n/2$ players. The threshold for an adaptive, active adversary is $n/3$.

If a physical broadcast channel is available, then every function can be computed securely with statistical security, if the adaptive, active adversary corrupts less than $t < n/2$ players [RBO89].

For the cryptographic model, Goldreich, Micali and Widgerson showed that any function can be securely computed with computational security in presence of a static, active adversary corrupting less than $n/2$ players [GMW87]. These results are summarized in Table 3.1, refer to [CDN08] for more details.

	Passive	Active with broadcast	Active without broadcast
Perfect	$n/2$	$n/3$	$n/3$
Statistical	$n/2$	$n/2$	$n/3$
Computational	$n/2$	$n/2$	$n/2$

Table 3.1: The threshold obtainable for various qualities of security, where all results are for adaptive security.

3.4 Computation Stages

According to a paper by Ben-Or, Goldwasser and Widgerson [BOGW88] a multiparty computation can be divided into three stages:

1. *The input stage*, where each player enters some value(s) which are shared according to a secret sharing scheme.
2. *The computation stage*, where the specified arithmetic circuit is evaluated with the shared values.
3. *The final stage*, where the output value(s) are reconstructed.

Chapter 3. Secure Multiparty Computation

Each stage is described next. Stage 2 is the most complicated, as it involves both addition and multiplication of shared values, and the details of the computation stage is thus explained last.

3.4.1 The Input Stage

Basically players give some input which is secret shared among the participants. As described in Section 2.3 this implies that it is possible to compute a function, while preserving the secrecy of the player's inputs.

Let P_0, \dots, P_{n-1} be a set of players that want function F to be computed. Assume that all inputs are elements from the finite field \mathbf{E} , with $|\mathbf{E}| > n$, and that F is some polynomial over \mathbf{E} .

Let $\alpha_0, \dots, \alpha_{n-1}$ be some n distinct non-zero element in our field \mathbf{E} . Each player then selects t random elements $a_i \in \mathbf{E}$, for $i = 1, \dots, t$, setting

$$f(x) = s + a_1x + \dots + a_tx^t$$

The value $s_i = f(\alpha_i)$ is sent to each player P_i . The sequence (s_0, \dots, s_{n-1}) consists of random variables uniformly distributed over \mathbf{E} and the value of the input is thus completely independent from the shares s_i that are distributed to the other players.

3.4.2 The Final Stage

In order for a player to reconstruct the secret value s he gathers the shares from all players and computes the interpolation polynomial $f(x)$ of degree t . The free coefficient is then the result and can be revealed.

All coefficients of $f(x)$, except the free coefficient, are uniform random variables independent of the inputs. This implies that the set of shares does not contain any information about the inputs except from what follows from the value of $f(0)$.

3.4.3 The Computation Stage

Addition and multiplication are described next as these are the fundamental operations in MPC (and VIFF).

Addition

Let s_f and s_g be two secrets that are shared with Shamir's secret sharing scheme using the polynomials $f(x)$ and $g(x)$, respectively. Every player has a share of both secrets denoted by $s_{i,j}$ where i is the polynomial and j is the player. The addition of s_f and s_g can be done locally by each player simply

3.4. Computation Stages

by adding its own shares of the secrets s_f and s_g resulting in a new share for each player. For three players the calculations are:

$$\begin{aligned}s_{new,1} &= s_{f,1} + s_{g,1} \\ s_{new,2} &= s_{f,2} + s_{g,2} \\ s_{new,3} &= s_{f,3} + s_{g,3}\end{aligned}$$

This is possible due to the following calculations. Let f and g be the two polynomials:

$$\begin{aligned}f(x) &= s_f + r_{1_f}x + r_{2_f}x^2 + \dots + r_{t-1_f}x^{t-1} \\ g(x) &= s_g + r_{1_g}x + r_{2_g}x^2 + \dots + r_{t-1_g}x^{t-1}\end{aligned}$$

Let $h(x)$ be the sum of $f(x)$ and $g(x)$:

$$\begin{aligned}h(x) &= f(x) + g(x) \\ h(x) &= (s_f + s_g) + (r_{1_f} + r_{1_g})x + (r_{2_f} + r_{2_g})x^2 + \dots + (r_{t-1_f} + r_{t-1_g})x^{t-1} \\ h(x) &= (s_f + s_g) + r_1x + r_2x^2 + \dots + r_{t-1}x^{t-1}\end{aligned}$$

The result is a new polynomial with the same degree as $f(x)$ and $g(x)$, where the coefficients in each term of $h(x)$ is the sum of the coefficients in the corresponding terms of $f(x)$ and $g(x)$. The polynomial $h(x)$ intersects the y-axis in the same point as the addition of $f(x) + g(x)$.

Addition Example

To illustrate why addition can be done locally, assume we have three secrets (0, 2 and 2). The polynomials are set to:

$$\begin{aligned}f(x) &= 2x \\ g(x) &= x + 2 \\ h(x) &= -x + 2\end{aligned}$$

We plot these functions for x values between 0 and 3, in addition to the sum of the three polynomials ($2x + 4$) which is marked *total* in Figure 3.1. The relation between the functions is clearly illustrated. Both $f(x)$, $g(x)$ and $h(x)$ were defined as functions of degree 1, and the *total* function is obviously of the same degree. Remember that the original secrets were 0, 2 and 2. The sum is 4 and corresponds to where the *total* polynomial intersects the y-axis.

Multiplication

Multiplication is a bit more complicated than addition. Again, let s_f and s_g be two secrets that are shared using the polynomials $f(x)$ and $g(x)$, respectively, which are of degree $t - 1$. The multiplication of two polynomials

Chapter 3. Secure Multiparty Computation

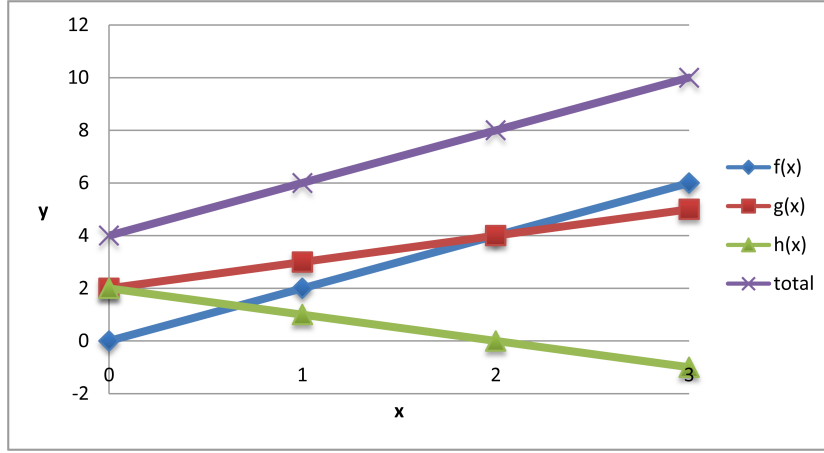


Figure 3.1: Illustration of how $f(x)$, $g(x)$ and $h(x)$ can be added together and form a new polynomial marked *total*.

of degree $t - 1$ will result in a new polynomial $h(x)$ with degree $2t - 2$. This would require more points for the interpolation used to reconstruct the secret, meaning that more players have to participate in the reconstruction. Additional multiplications will raise the degree even further, eventually rendering the interpolation impossible due to the lack of participating players. To overcome this problem, $h(x)$ needs to be reduced to the original degree $t - 1$. Let f and g be the two polynomials:

$$\begin{aligned} f(x) &= s_f + r_{1_f}x + r_{2_f}x^2 + \dots + r_{t-1_f}x^{t-1} \\ g(x) &= s_g + r_{1_g}x + r_{2_g}x^2 + \dots + r_{t-1_g}x^{t-1} \end{aligned}$$

The multiplication of $f(x) \cdot g(x)$ results in a new polynomial $h(x)$:

$$\begin{aligned} h(x) &= f(x) \cdot g(x) \\ h(x) &= s_f g(x) + r_{1_f} x g(x) + r_{2_f} x^2 g(x) + \dots + r_{t-1_f} x^{t-1} g(x) \\ h(x) &= s_f s_g + s_f r_{1_g} x + s_f r_{2_g} x^2 + \dots + r_{1_f} s_g x + \dots + r_{t-1_f} x^{t-1} r_{t-1_g} x^{t-1} \end{aligned}$$

To clarify $h(x)$ can be written in the following form:

$$h(x) = s_f s_g + r_1 x + r_2 x^2 + \dots + r_{2t-2} x^{2t-2}$$

Each player now holds a “share” of $h(x)$, a polynomial of degree $2t - 2$, which needs to be reduced to a polynomial of degree $t - 1$. These $h(x)$ outputs are then used as input to a new round of sharing, which results in a new set of shares in new random polynomials on the form of $i(y)$:

$$i(y) = h(x, y) = h(x) + r_1 y + r_2 y^2 + \dots + r_{t-1} y^{t-1}$$

Solving this results in a polynomial of the correct degree with $s_f s_g$ as the free coefficient:

$$i(y) = s_f s_g + r_1 y + r_2 y^2 + \dots + r_{t-1} y^{t-1}$$

3.4.4 Multiplication Example

In order to explain the secret shared multiplication, an example with small numbers is included below. Two secrets are defined (3 and 2), and the two polynomials are set to:

$$\begin{aligned} f(x) &= 3 - 2x \\ g(x) &= 2 + x \end{aligned}$$

Each player has a share in $f(x)$ and $g(x)$ where $x = 1, 2, 3$ for player 1, player 2 and player 3, respectively.

$$\begin{aligned} \text{Player 1: } & f(1) = 1 \quad g(1) = 3 \\ \text{Player 2: } & f(2) = -1 \quad g(2) = 4 \\ \text{Player 3: } & f(3) = -3 \quad g(3) = 5 \end{aligned}$$

By multiplying the shares from $f(x)$ and $g(x)$ each player obtain a “share” in $h(x)$. The players share these values with a new random polynomial as shown in row 4 in Table 3.2. The rest of the table is calculated by each player inputting $x = 1, 2, 3$ in its own polynomial and distributes a share to each of the other players. As an example, player 1 calculates:

$$\begin{aligned} \text{Share 1: } & 3 + 2 \cdot 1 = 5 \\ \text{Share 2: } & 3 + 2 \cdot 2 = 7 \\ \text{Share 3: } & 3 + 2 \cdot 3 = 9 \end{aligned}$$

Player 1 then distributes share 2 to player 2 and share 3 to player 3. Both player 2 and player 3 calculate their column in Table 3.2 and distribute the shares to the other players.

Now each player holds its secret polynomial (player 1 holds $3 + 2x$ etc.) together with a single point from each of the other players’ polynomials (player 1 receives -1 from player 2 and -14 from player 3). With this information, each player can calculate its share of the total polynomial using one of two methods:

- Linear system approach (Appendix A.1)
- Vandermonde matrix (Appendix A.2)

Chapter 3. Secure Multiparty Computation

	Player 1	Player 2	Player 3	
$f(x)$	1	-1	-3	
$g(x)$	3	4	5	
$h(x)$	3	-4	-15	
	$3 + 2x$	$-4 + 3x$	$-15 + x$	S_h
Player 1	5	-1	-14	4
Player 2	7	2	-13	2
Player 3	9	5	-12	0

Table 3.2: Example matrix for secret shared multiplication.

These calculations give the players the values seen in Table 3.3, which can also be found in the s_h column in Table 3.2:

Player	Share value
1	4
2	2
3	0

Table 3.3: The players' shares of the total polynomial.

When a subset of at least two players exchanges shares the secret can be reconstructed. By plotting the values as shown in Figure 3.2, the secret is found where the line intersects the y-axis. The revealed number is 6, which corresponds with the multiplication of the initial secrets 3 and 2.

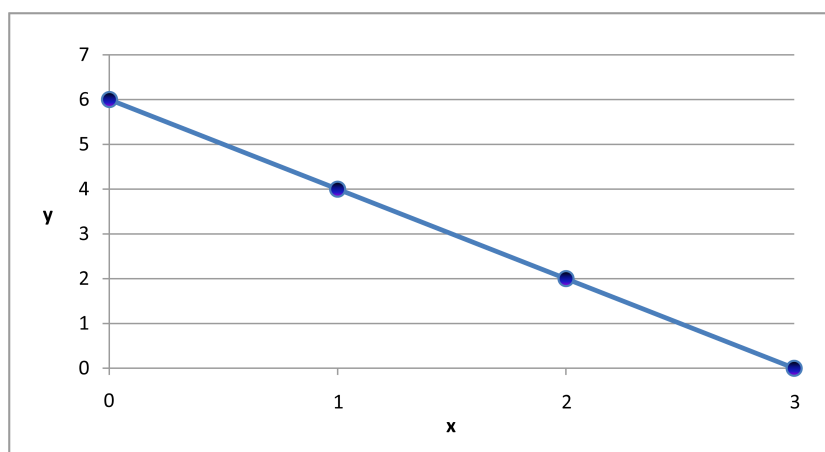


Figure 3.2: Graph with the points (1,4), (2,2) and (3,0) which implies that the secret is 6 (the line intersects the y-axis).

3.5 MPC Frameworks

In order to make the creation of applications based on multiparty computations simpler, several MPC frameworks have been developed. They are all listed below together with a brief description. VIFF is further described in Chapter 4 and is also the framework used for developing the applications in this thesis.

- *The Virtual Ideal Functionality Framework (VIFF)* allows you to specify secure multiparty computations in a clean and easy way [14]. It was started by Martin Geisler and grew out of the SIMAP project mentioned below.
- *The SIMAP project* is conducted at the University of Aarhus, Denmark in collaboration with the University of Copenhagen and industry partners [4]. The tools are a set of efficient cryptographic protocols and a domain-specific language named Secure Multiparty Computation Language (SMCL) [7].
- *The FairPlay project* is an initiative from the Hebrew University of Jerusalem and the University of Haifa in Israel. Fairplay [MNPS04] is a system for secure two-party computation, and FairplayMP [BDNP08] is a different system for secure computation by more than two parties.
- *Sharemind* is another project aiming to be an efficient and easily programmable platform for developing privacy-preserving computations [BLW08]. It is currently developed by people in the University of Tartu in Estonia and AS Cybernetica.

Chapter 4

Virtual Ideal Functionality Framework

The Virtual Ideal Functionality Framework (VIFF) provides a Python library for creating SMPC protocols. It allows three or more parties to execute a cryptographic protocol to do some joint computation, without the players revealing anything about their inputs. This chapter gives an introduction to VIFF including the history of the project, the architecture, current features and an example VIFF program.

4.1 History

VIFF was started by the PhD student Martin Geisler in March 2007. It grew out of a research project called Secure Information Management and Processing (SIMAP) [4] carried out at the University of Aarhus, Denmark. SIMAP, which again is the successor to the Secure Computing Economy and Trust (SCET) project [3], was the first project to create and run a large-scale application of multiparty computation. This took place in January 2008 and was a double auction directed at Danish farmers trading sugar beet contracts [BCD⁺08].

The sugar beet auction was programmed in Java in the SIMAP project. This Java implementation was big (about 8,500 lines of code for some 130 classes and interfaces) and a number of problems with its design were detected. The new VIFF implementation in Python supports the same protocols as the SIMAP project and more, with considerably fewer lines of code [13].

4.2 Overview

VIFF hides the difficult cryptographic details and as a developer it works like a high-level API for writing multiparty computations. Figure 4.1 illustrates

Chapter 4. Virtual Ideal Functionality Framework

how VIFF takes care of network communication, secret sharing and operations on shares in MPC. Instead of writing for instance `add(add(a, b), c)` one can just type `a + b + c` which then is compiled to VIFF library calls interpreted by the Python Virtual Machine. The upper row named *Development* is where the work of this thesis is concentrated.

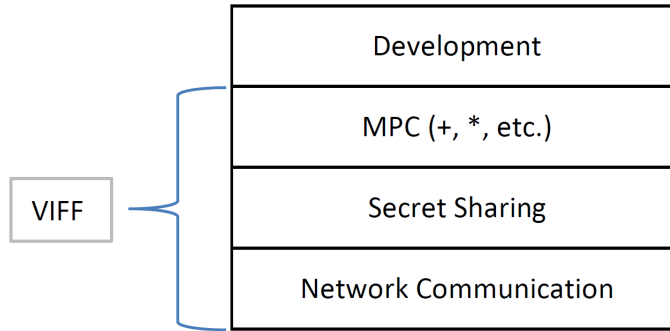


Figure 4.1: The protocol stack which VIFF is built upon.

4.3 Features

Current features include:

- Arithmetic with shares from \mathbf{Z}_p or $\mathbf{GF}(2^8)$. Further described in Section 4.4.2.
- Secret sharing based on Shamir and pseudo-random secret sharing (PRSS).
- Secure addition, multiplication, and exclusive-or of shares.
- Comparison of secret shared \mathbf{Z}_p inputs, with secret \mathbf{Z}_p or $\mathbf{GF}(2^8)$ output.
- Automatic parallel (asynchronous) execution. More information is given in Section 4.5.
- Secure communication using SSL.

4.4 Architecture

VIFF consists of several modules. The main functionality is implemented in `viff.runtime` module, while the `viff.field` module contains implementations of finite fields. These modules are further explained next.

4.4.1 Runtime

The `viff.runtime` module is where the virtual ideal functionality is hiding. It is responsible for sharing inputs, handling communication and running calculations. The module contains the `Runtime` and `Share` classes. `Runtime` offers methods to do addition, multiplication, etc. and these methods operate on `Share` instances.

Figure 4.2 illustrates how `Runtime` objects interconnect with `Share` objects. A number of `Share` objects can exist on each party (represented by a circle) and they use the `Runtime` object when asked to perform calculations like addition, multiplication, etc. The `Runtime` objects are again connected to each other via `ShareExchanger` objects, which maintain SSL connections between the parties [GDP09].

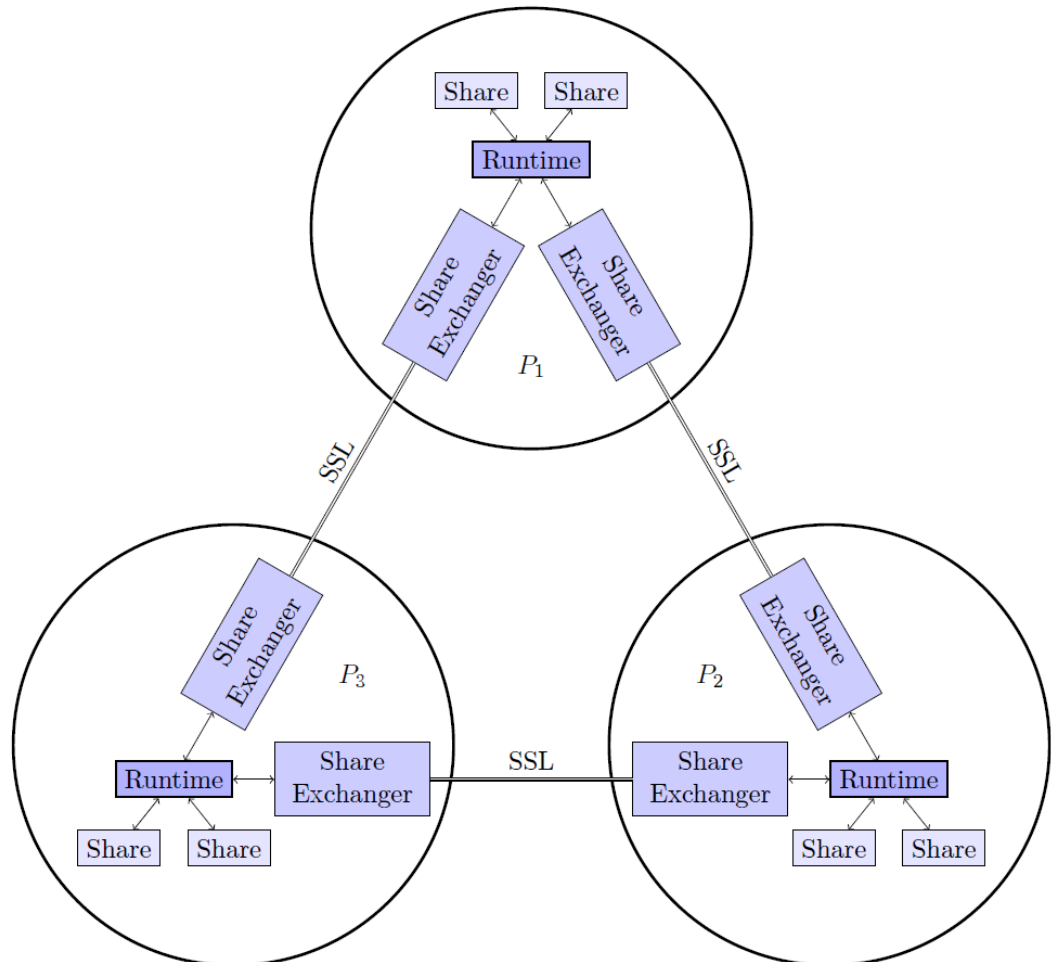


Figure 4.2: Relations between class instances at runtime [GDP09].

4.4.2 Finite Fields

VIFF provides classes for modeling Galois fields. All fields work the same, that is, one instantiates an object from a field to get hold of an element of that field. Normal arithmetic like addition, multiplication, etc. is provided via overloaded operators. An example demonstrating how to define a field, create field elements and do multiplication and addition (with modulo reduction), is showed below.

```
Zp = GF(19)
a = Zp(10)
b = Zp(5)
c = 2 * a + b
```

Because of the GF(19), the `FieldElement` object `c` will wrap around and hold the value of 6.

4.5 Asynchronous Design

VIFF aims to be usable by parties connected by real world networks, like the Internet, where a message is likely to go through many hops which introduce an unpredictable delay. In a synchronous setting all parties wait for each other at the end of each round, but VIFF has no concept of rounds. Instead VIFF provide *asynchronous* communication by using a Python framework called Twisted. This makes it possible to create a function and arrange for this to be called when the data is available.

4.5.1 Expression Tree

As an example, consider the expression tree in Figure 4.3. It illustrates the calculation: $z = x + y = (a * b) + (c * d)$. The variables a , b , c and d all represent secret shared values, while the arrows denote dependencies between the expressions. The two variables x and y are mutually independent and may be calculated in parallel.

Two factors determine the execution time of a multiparty computation, that is, the speed of the CPUs engaged in the local computations and the delay through the network. Normally the network latency will dominate as it can reach several hundred milliseconds. In this context parallel means that when the calculation of x waits on network communication from other parties, then the calculation of y must get a chance to begin its network communication. This will put maximum load on both the CPU and the network [DGKN08].

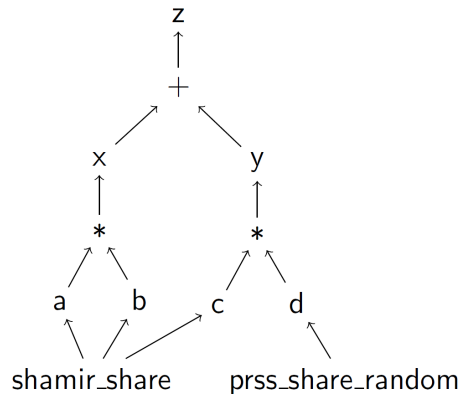


Figure 4.3: Expression tree [DGKN08].

4.5.2 Twisted

As mentioned, VIFF uses the event-driven networking engine Twisted in order to build efficient network applications with asynchronous communication. In contrast to a synchronous model where system calls may be blocked, Twisted is using non-blocking calls. Operations in VIFF may be executed in a random order because *deferred* objects are used and never the results themselves. A *deferred* result will eventually contain some data, but in the meanwhile it is just added as a *callback* function. This function will be called when the data is ready [16].

```

from twisted.web.client import getPage
from twisted.internet import reactor

def printContents(contents):

    print "The Deferred has called printContents with the
        following contents:"
    print contents

    reactor.stop()

deferred = getPage('http://twistedmatrix.com/')
deferred.addCallback(printContents)

# Start the Twisted event loop.
reactor.run()
  
```

Listing 4.1: The use of callbacks in Twisted.

Listing 4.1 shows a call to `getPage`, which returns a *Deferred*. A callback is attached in order to handle the contents of the page once the data is available.

4.6 Example VIFF Program

As mentioned, VIFF hides much of the difficult cryptographic concepts. Next follows a review of maybe the simplest VIFF program one can create. Listing 4.2 shows a program for three players who each provide a private input (in this case a random integer between 1 and 200). They want to calculate the sum of their total financial wealth, but do not want to reveal its own number. The following steps occur in the program:

- Each player executes `python example.py --no-ssl player-i.ini` where `i` is the player number.
- The first lines import standard and VIFF modules.
- Each player has a configuration file with information about the other players. An example is given in Listing 4.3.
- On line 37 the `create_runtime` function is asked to run the protocol when ready. Among others it schedules the opening of TCP connections between the parties. In order to start the associated events, the Twisted reactor needs to be started (`reactor.run()` on line 39).
- On line 13 the private inputs are created. Each player generates a random number between 1 and 200.
- On line 15 the finite field \mathbf{Z}_p is defined. The calculations will be done over this field, so it is important that it is bigger than the largest values that can occur.
- Line 16 invokes the `shamir_share` method. All players contribute their inputs which are secret shared and distributed to the other players. Each player now holds three variables: `m1` is the player's share of the input number from player 1, `m2` is the share from player 2 and `m3` is the share from player 3.
- Two secure addition operations are done in line 17. The variables are `Share` instances, so the `+` operator call the `runtime.add` method.
- On line 20 the sum is opened, meaning that the shares are sent to the designated receivers and Shamir recombines them. Note that the `open_sum` is also a `Share` and it is not possible to directly print its value.
- A callback to `results_ready` is done in line 21 which prints the result.
- The `runtime.shutdown()` callback on line 22 will make the players synchronize, close the TCP connections and stop the reactor.

4.6. Example VIFF Program

```
1 from optparse import OptionParser
2 from twisted.internet import reactor
3
4 from viff.field import GF
5 from viff.runtime import Runtime, create_runtime, gather_shares
6 from viff.config import load_config
7 from viff.util import rand
8
9 class Protocol:
10
11     def __init__(self, runtime):
12         self.runtime = runtime
13         self.millions = rand.randint(1, 200)
14
15         Zp = GF(1031)
16         m1, m2, m3 = runtime.shamir_share([1, 2, 3], Zp, self.
17             millions)
18         sum = m1 + m2 + m3
19         open_sum = runtime.open(sum)
20
21         results = gather_shares([open_sum])
22         results.addCallback(self.results_ready)
23         results.addCallback(lambda _: runtime.shutdown())
24
25     def results_ready(self, results):
26         sum = results[0].value
27         print sum
28
29 parser = OptionParser()
30 Runtime.add_options(parser)
31 options, args = parser.parse_args()
32
33 if len(args) == 0:
34     parser.error("You must specify a config file")
35 else:
36     id, players = load_config(args[0])
37
38 pre_runtime = create_runtime(id, players, 1, options)
39 pre_runtime.addCallback(Protocol)
40 reactor.run()
```

Listing 4.2: A simple VIFF example program.

Chapter 4. Virtual Ideal Functionality Framework

As mentioned, each player has a configuration file with information about the other players. The host address and port number are present in order to know who to contact. The `pubkey` and `seckey` hold Paillier keys [Pai99] used for homomorphic encryptions and the `prss_keys` hold shared keys used for pseudo-random secret sharing [CDI05] among players. Note that player 1 has full information about its own keys, but only the public keys of the other players.

```
1 # VIFF config file for Player 1
2
3 [Player 1]
4   host = localhost
5   port = 9001
6   pubkey = 6495900803...29, 4135810115...52
7   seckey = 6495900803...29, 4135810115...52, 5413250669...32
8   [[prss_keys]]
9     1 3 = 0xc882b63ebd7ab023cda1b4c9ba18787a3563f21dL
10    1 2 = 0x93a05fd6b6c682bd543a8cc4d016df1b37d5a157L
11   [[prss_dealer_keys]]
12     [[[Dealer 1]]]
13       1 3 = 0x3a74bc9292ab8332ee4939c29cc8e05e668b2fa4L
14       1 2 = 0xdbaf5936f3a904272454ba85c86912837afb6359L
15       2 3 = 0x234c4eff1e228e14beb1e0c36edf787189269469L
16     [[[Dealer 2]]]
17       1 3 = 0x92731178e72dab4367a4238bfe0a6654451cc59aL
18       1 2 = 0xc4f35dda3552573d11b69c408faf401d3ff67fL
19     [[[Dealer 3]]]
20       1 3 = 0xf8a8115e3b4a694c49bdef3bdb37fa6b5b573c1cL
21       1 2 = 0x652b4be0c1afb12c5b1d860689e177db181deaceL
22
23 [Player 2]
24   host = localhost
25   port = 9002
26   pubkey = 1228140364...19, 1504025515...44
27
28 [Player 3]
29   host = localhost
30   port = 9003
31   pubkey = 3149769946...77, 1472902211...82
32
33 # End of config
```

Listing 4.3: VIFF configuration file for player 1. The `pubkey` and `seckey` have been abbreviated.

4.7 Security Assumptions

VIFF is short for Virtual Ideal Functionality Framework and programs will implement a virtual ideal functionality (IF). The Universally Composable (UC) security framework by Canetti [Can01] defines security as follows:

Definition 4 *A protocol is secure if an outside observer cannot distinguish between an execution of the real world protocol and an ideal world protocol.*

The real world is where the actual protocol and attacks on it take place. The ideal world protocol is a specification of what we would like the protocol to do. In order to prove a protocol secure, one takes the protocol which is known to be secure and prove that the new protocol is indistinguishable from the ideal scenario.

There is no IF in the real world, but VIFF makes it possible to implement a virtual ideal functionality. A VIFF program is indistinguishable from a real world program in the sense that everything that can happen in a real world protocol could happen in the ideal world too. Since no successful attacks can occur in a ideal world, no successful attacks can occur in the real world either [15].

Any cryptographic system has security assumptions that need to be fulfilled in order to be considered secure. The creators of VIFF include three security assumptions in their documentation:

1. There is a threshold of adversaries that can be corrupted. Only 1/2 of the players may be corrupted, that is, there must be an honest majority. In the case of three players, only one player can be corrupted.
2. An adversary is computationally bounded. The protocols used by VIFF rely on certain computational hardness assumptions, and therefore only polynomial time adversaries are allowed.
3. VIFF currently only supports passive adversaries as described in Section 3.1, meaning that the adversary can monitor network traffic, but still follows the protocol. Support for active (Byzantine) adversaries is planned in a future version.

4.8 Benchmarking

Evaluating the performance of programs implemented in VIFF is dependent on several parameters, like number of operations and number of players. The type of operation is also very important. As described in Section 3.4.3 addition and multiplication have different requirements. Addition can be

Chapter 4. Virtual Ideal Functionality Framework

done locally while multiplication requires an extra step of re-sharing and thus network traffic. As a consequence, comparison is also a lot more time-consuming than addition, since it involves several multiplication operations.

As a part of his PhD progress report [Gei08], Martin Geisler provides benchmark results done with VIFF version 0.4 (released March 12th 2008). The benchmark was run over the Internet using full TLS encryption and by using three computers located in Denmark, Norway and USA. The main results for multiplication and comparison are given below to indicate the efficiency.

Figure 4.4 shows how the average time per multiplication drops rapidly at first until it stabilizes at around 1.3 ms. Remember that VIFF has a parallel execution and by default many multiplications will be started before the first one finishes.

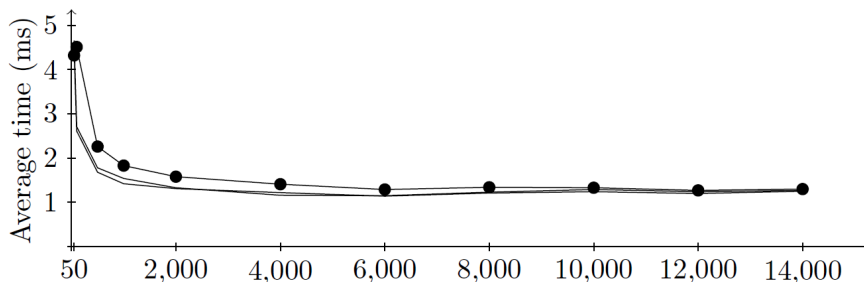


Figure 4.4: Parallel multiplication benchmarks, average time per multiplication [Gei08].

Figure 4.5 shows that the average time of a comparison falls to around 800 ms, which is much slower than multiplication. Note that the comparison protocol by Toft [Tof05] in this benchmark has been improved, but comparisons are still an efficiency problem, not only in VIFF, but in MPC in general.

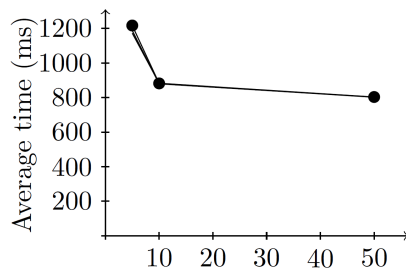


Figure 4.5: Parallel comparison benchmarks, average time per comparison [Gei08].

4.8. Benchmarking

Other results from the same report indicate that time spent on local computations (CPU time) is negligible compared to the communication cost between the computers.

Chapter 5

Voting

This chapter gives a brief overview of electronic voting with emphasis on different electronic voting types and typical security requirements. Note that the voting application implemented in Section 8 is not a full-scale election; however, it will be analyzed according to the same requirements.

5.1 Overview

Electronic voting refers to the use of computers or computerized voting equipment to cast ballots in an election [Cet08]. There are several types of voting and based on voting equipment and voting location, electronic voting can be classified into five different types [Cet09]. A description is given below, while Table 5.1 summarizes them all including paper based voting.

- *DRE voting* (Direct Recording Electronic) is physically hardened electronic equipment with special purpose voting software. The votes are cast inside a voting booth at a polling site, however, cast votes are recorded in electronic ballot boxes.
- *Poll-site voting* does not use voting booths, but public computers at a polling-site. The computers are connected over a closed and controlled network.
- *Poll-site kiosk voting* typically contains electronic voting terminals inside a voting booth at a polling site (as in DRE voting). The terminals are connected with a closed and controlled network.
- *Poll-site Internet voting* provides a polling-site where users cast their votes by using public computers. The computers at the site are online over an uncontrolled network.

Chapter 5. Voting

- *Remote Internet voting* only requires Internet access and can be done from your home computer. For authentication, the credentials of voters are verified prior to the voting period through the use of a password or some type of authentication token.

	Stand-alone Voting	Networked Voting	
		<i>Controlled Network</i>	<i>Uncontrolled Network</i>
Paper Voting	Paper-based Voting	N/A	N/A
Electronic Voting	DRE Voting	Poll-site kiosk voting	Poll-site Internet voting
		Poll-site voting	Remote Internet voting

Table 5.1: Classification of voting types.

5.2 Motivation

Most aspects of our lives have online components, ranging from e-mails, music and other types of entertainment. Still the old fashioned paper ballots are used in polling (or voting), but we have seen an increase in the use of electronic voting systems in the recent years. Some countries are pioneers in the use of electronic voting, even over the Internet. In the United States 7.7% of the voters used electronic voting systems even in 1996. This use was mainly through DRE machines, which also have been used in Brazil, Venezuela, India and Netherlands. Internet voting has been used in United States, United Kingdom, Ireland, Switzerland, Canada, France and Estonia. However, most countries have their own approach for Internet voting. As an example, voters in Switzerland receive their access passwords through mail, while Estonia uses advanced national identity cards equipped with electronic chips. Also in Norway there are plans for e-voting, but not until 2011 [Usc08].

5.3 The Voting Process

There exists a wide variety of electronic voting systems and protocols, but the main process is almost standard. The typical actors participating in any voting system is described below [Cet09]:

- *Voter*: A voter is entitled to vote in the election, and has some private information that identifies it.
- *Registration authority*: The registration authority ensures that only registered voters can vote (and only once).
- *Collection authority*: The collection authority collects all cast votes.

- *Tallying authority*: The tallying authority computes the result of the election and publishes it.

5.4 Challenges

Incorporating electronic voting, especially for use in nation-wide elections, requires the fulfillment of several security requirements. The traditional paper-based voting is open and easy to understand, while electronic voting involves a more complex system. Internet is open and reachable by anyone, thus the e-voting results can in theory be manipulated at any step in the voting process. Sufficient security and cryptography is thus crucial to ensure the trustworthiness of the system.

In their paper Damgård, Groth and Gorm outline three important challenges that need to be solved [DGS02]:

- *Privacy*: Only the final result should be made public, no information about the votes must be leaked.
- *Robustness*: The result will reflect all submitted and well-formed ballots correctly, even if some voters or entities running the election cheat.
- *Universal verifiability*: After the election, the result can be verified by anyone. In other words, any party should be able to convince himself that the election was fair in the sense that the published tally was correctly computed.

When implementing a voting scheme it could be defined analogously to the security definition of MPC, that is, with help of a fictive trusted party which is simulated by a protocol. Each voter would then send its vote to the trusted party, who selects the valid votes, adds them up and publishes the tally. A voting protocol would thus be secure if an adversary cannot achieve more than what he could in this specification. However, this is not the standard approach for defining security of a voting scheme. Usually a list of properties that must be satisfied is given. In conformance with most literature, such security requirements are listed next.

5.5 Security Requirements

In his paper, *Analysis of Security Requirements for Cryptographic Voting Protocols*, O. Cetinkaya lists security requirements that a secure and complete cryptographic voting protocol should satisfy [Cet08]:

- *Voter privacy*: The prevention of associating a voter with a vote.

Chapter 5. Voting

- *Eligibility*: Only eligible voters who are registered can cast votes.
- *Uniqueness*: Only one vote per voter should be counted.
- *Fairness*: No partial tally is revealed before the end of the voting period to ensure that all candidates/choices are given a fair decision.
- *Uncoercibility*: Any voter must be able to vote freely and no coercer should be able to extract the value of the vote.
- *Receipt-freeness*: The system should not provide a confirmation of the receipt of the vote which may yield its content.
- *Accuracy*: The published tally should be correctly computed from correctly cast votes.
- *Individual vote check*: The voter should be able to check that his encrypted vote was counted and tabulated correctly in the final tally.

Chapter 6

Choice of Applications

The first chapters about MPC, secret sharing, VIFF and voting have served as an introduction to what this thesis aspires to achieve: the development of applications using multiparty computation. This chapter will first describe the high-level motivation for the applications, before explaining the difference between the two.

6.1 Motivation and Properties

As already mentioned, multiparty computations have great potential. However, most practical applications exist only in scientific papers. As an example Du and Atallah [DA01] have studied a number of applications, but only aim to stimulate researchers into coming up with new theoretical MPC problems. In a publication by CACE¹ a long list of applications together with relevant properties that needs to be implemented are listed [Pin08]. Are some of these applications implemented and publicly available? If the potential of MPC is so great, why are there not more practical applications out there?

One counter-example is the double auction in which Danish farmers can trade sugar beet contracts using MPC [BCD⁺08]. This master's thesis aims to implement applications that are practical and could be useful for a large audience. In order to make a practical application, several properties are defined:

- The application must be easy to understand, which implies a self-explanatory GUI. A user should never have to write code into a command line window to make it work.
- The application must be user-friendly. It should not be necessary to install a lot of other packages and programs to make it work.

¹Computer Aided Cryptography Engineering

Chapter 6. Choice of Applications

- The application must be easy accessible, which in practice mean it should be available on the world wide web.

Not only would an application fulfilling these properties be interesting to implement, but I believe a public deployment of more MPC applications would increase the interest for this promising field of security. Next follows a brief introduction to each of the two implemented applications with focus on which properties they hold and to what extent they fulfill needs existing among the public.

6.2 The Author Application

Chapter 7 describes a simple and specific voting program where authors on a scientific paper can vote for their choice of 1th, 2th, 3th, etc. author. This is done with the help of a self-explanatory GUI and is thus easy to understand. The usefulness is more arguable as the problem of selecting authors is quite far-fetched and not anything that would interest ordinary non-scientific people.

The biggest drawback, however, is the extensive setup needed to make the application run. Each participant would have to install VIFF and a lot of required components. Each participant would also need to create a configuration file containing, among others, a public and a private key. This procedure is clearly too extensive.

6.3 The Voting Application

Chapter 8 describes a more general voting application and solves the problems present in the first application. The actual computation is now delegated to three computation servers, making the requirements of each participating user much less. A web browser with Java support is all that is needed.

With its graphical interface, both for creating a poll and for giving a vote, the application is easy to understand. It is also very user-friendly since most people know how to use a web browser. Finally, and perhaps most importantly, it is available on the Internet. It is a web application offering easy and practical voting, but at the same time provides a strong level of privacy.

Chapter 7

Application 1: Rank the Authors

As mentioned in Chapter 1.3 I wrote a project in cooperation with SINTEF ICT in the second half of 2008. The work resulted in a scientific paper and we had no tools for deciding whom to be the first author, other than drawing lots. This chapter describes an application addressing this problem.

The author program lets all participants on a scientific paper vote on their choice of 1th, 2th, 3th, etc. author. This is done via a graphical interface where it is not possible to vote for yourself. The votes are secret shared and finally added up before revealing only the ranking of the authors.

Section 7.1 contains information about the functionality and implementation of the application. A few benchmark tests are given in Section 7.2 while some possible improvements is listed in Section 7.3.

7.1 Design and Implementation

The implementation is based on VIFF, which is a Python framework for specifying secure multiparty computations. First the application in action is demonstrated through a series of screen shots. Next follows a description of the architecture and libraries used, before examining the code and important choices more in detail.

7.1.1 Screen Shots

Assume four participants (Atle, Håvard, Tord and Martin) want to decide the order of author names on a scientific paper. Using the author application each player will see the window in Figure 7.1 when running the program. The figure shows Atle's view and it is obvious that he is not able to vote for

Chapter 7. Application 1: Rank the Authors

himself.

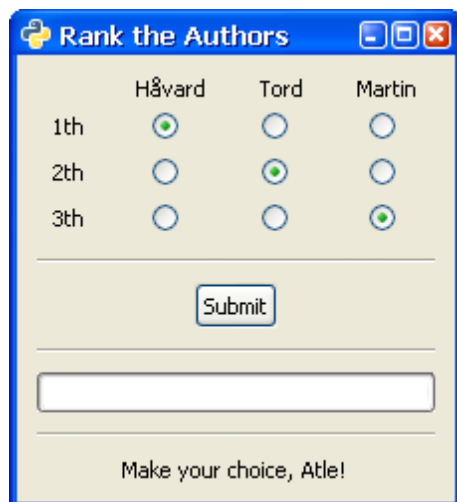


Figure 7.1: The main window of the author application. Atle cannot vote for himself.

One could imagine that Atle would try to give all votes to Håvard, in order to outsmart the two last players, but as seen in Figure 7.2, this would result in an error message.

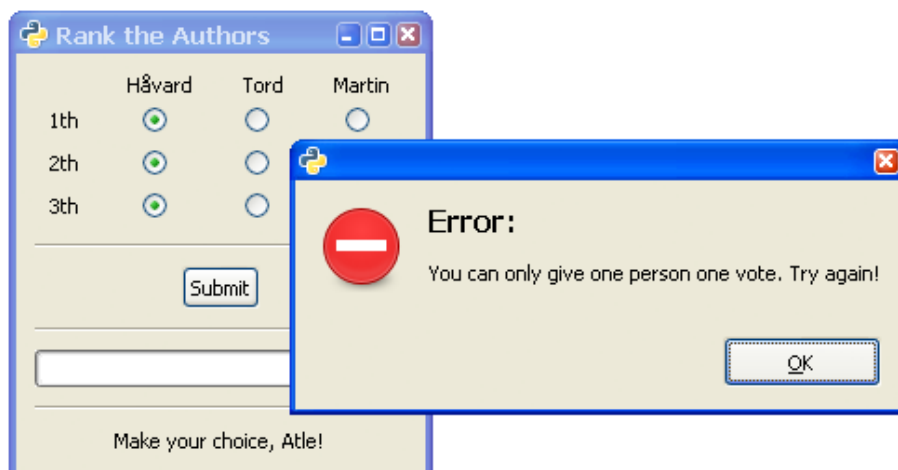


Figure 7.2: Atle tries to give all the votes for Håvard, but gets an error message.

Following the correct procedure as indicated in Figure 7.3, Atle selects Håvard as 1th author, Tord as 2th author and Martin as 3th author. Pushing the submit button will initiate a secret sharing procedure of Atle's vote and the GUI is updated. Atle is the only player so far that has given his vote,

7.1. Design and Implementation

which is reflected in the progress bar labeled 25%.

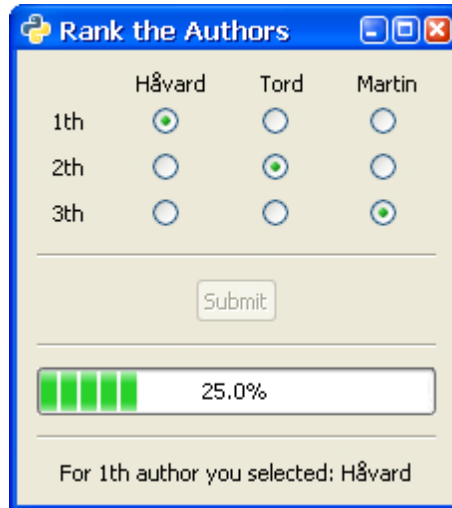


Figure 7.3: Atle has voted and 25% of the players have given their inputs.

When all players have given their votes, the progress bar reaches 100% and VIFF will calculate the result. It takes a few seconds, mainly due to comparison of shares, which is a time-consuming task. As displayed in the bottom of Figure 7.4 the only output is the sequence of authors, nothing else is revealed.



Figure 7.4: The result (order of authors) is securely calculated. Nothing else is revealed.

Chapter 7. Application 1: Rank the Authors

7.1.2 Architecture

The author program is a relatively simple application. The main VIFF code is located in one class which each player has to execute. Table 7.1 lists the three files needed to start the program. In addition one would need to install VIFF together with the GUI toolkit PyGTK. Installation instructions can be found in Appendix B.

File	Description
player-x.ini	Config file containing information like hostname and port number of the other players. See an example in Listing 4.3 in Section 4.6.
author_config.txt	Config file containing the participating players' names. This file is imported into the main VIFF program.
author.py	The main VIFF program which initiates the GUI, receives votes, secret shares the votes and finally outputs the ranking of the authors. A more detailed review is given in Section 7.1.4.

Table 7.1: List of files required for executing the author program.

When distributing the config files (.ini), note that `player-1.ini` corresponds to the first name in `author_config.txt`, `player-2.ini` corresponds to the second name in `author_config.txt`, etc. In a real world scenario each player would probably build up the config file itself, including its own private key and the others player's public key. When developing small applications it is though much easier to generate all files and run the program on the local host.

After starting the application, each participating player will deliver its vote. Figure 7.5 shows the basic idea if three players were to execute the program and had one number each to share. Then player 1 (P_1) has the variable x which is secret shared into x_1 , x_2 and x_3 . Player 1 will keep share x_1 for itself, while sending share x_2 to player 2 and share x_3 to player 3.

Note that in the author program there is not only one number for each player, but n numbers, where n is the number of players. If three persons are executing the program, player 1 would for instance have 3 numbers, that is, the amount of points for the selected 1th author, some points for the 2th author and 0 points for the 3th author (which would be the player itself). When secret sharing these three numbers among all players, each numbers will result in three shares, meaning quite a lot of shares are created.

All the shares are handled by VIFF and the only output from the application will be the ranking of the authors.

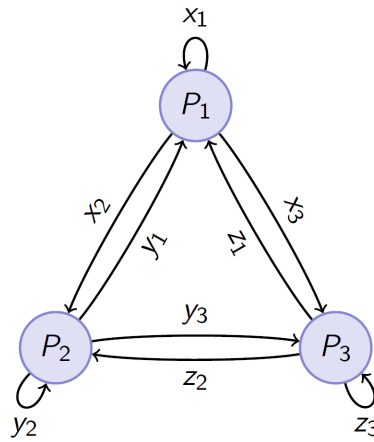


Figure 7.5: Player 1, 2 and 3 are sharing x , y and z [6].

7.1.3 Libraries

One of the main goals with this program was to create an application with an easy graphical interface. To achieve this, a GUI toolkit for Python was needed. Several toolkits, like PyQt, Tkinter and wxPython were considered, but PyGTK¹ was chosen due to its integration with Twisted [2]. This is important in order to run both the network and graphical event loops.

In order to sort arrays secret shared, an implementation of a secure sorting algorithm made by the VIFF Development Team is used. The algorithm used is bitonic sort [9] and was chosen in order to maximize the amount of work (comparisons) done in parallel.

7.1.4 Code

This section will explain the essential parts of the author program, focusing on how the players get their points and which numbers that are shared. The complete source code is included in Appendix C.

Assume four players are running the author program. When for instance player 1 clicks the submit button, an array of this player's ranking could be as indicated below, meaning player 2 is ranked as 1th author, player 4 as 2th author and player 3 as 3th author.

```
self.list[0] = 2
self.list[1] = 4
self.list[2] = 3
```

¹GTK+ for Python <http://www.pygtk.org>

Chapter 7. Application 1: Rank the Authors

Since player 1 cannot vote for itself, it is left out of the array. The code in Listing 7.1 then transforms the ranking array into an array containing points.

```
self.points = range(self.number_of_players)
pts = self.number_of_choices
for i in range(0, self.number_of_choices):
    x = self.list[i]
    self.points[x-1] = pts
    pts -= 1
# Give yourself 0 points
self.points[self.runtime.id-1] = 0
```

Listing 7.1: Distribution of points for the votes given in the author application.

First author results in $n - 1$ points, second author in $n - 2$ points, etc. As indicated in the new array below, player 1 receives 0 points, player 2 receives 3 points, player 3 receives 1 point and player 4 receives 2 points.

```
self.points[0] = 0
self.points[1] = 3
self.points[2] = 1
self.points[3] = 2
```

It is these n variables which are player 1's inputs to the shamir share function in the VIFF code. In the example with four players, a total of 16 ($n \cdot n$) variables/shares are created.

```
var_num = 0
for i in range(0, self.number_of_players):
    player = 1
    for j in range(0, self.number_of_players):
        if self.runtime.id == player:
            exec 'a%s = self.runtime.shamir_share([player], Zp,
            self.points[i])' % var_num in globals(), locals()
        else:
            exec 'a%s = self.runtime.shamir_share([player], Zp)'
            % var_num in globals(), locals()
    var_num += 1
    player += 1
```

Listing 7.2: The votes/points are given as inputs to the shamir share function.

The most important line in Listing 7.2 is `self.runtime.shamir_share([player], Zp, self.points[i])`. A variable containing some points is secret shared into four different representations of this number and then each player gets one share. In order to calculate the player with the most points, each player

executes the code in Listing 7.3.

```
# Calculate the sum of points for each player
array = []
var_num = 0
for i in range(0, self.number_of_players):
    exec 'sum%s = 0' % i in globals(), locals()
    for j in range(0, self.number_of_players):
        exec 'sum%s += a%s' % (i, var_num) in globals(), locals()
        var_num += 1
    exec 'array.append(sum%s)' % i in globals(), locals()
```

Listing 7.3: The points, represented as shares, are summed up.

Notice that these sums, in accordance with the description of VIFF in Chapter 4, do not contain any understandable numbers yet. Not until shares from the other players are received, and the VIFF framework starts computing the sums, can one make any sense of the information. The array of sums is also sorted secret shared and the only output which is revealed is the ranking of the authors.

7.2 Benchmarks

To evaluate the performance and scalability of the author application, a number of benchmark tests were run over a fast local area network (ping times < 1 ms) using full SSL encryption. The computers had Intel Pentium CPUs with clock speeds ranging from 1.5 GHz to 2.60 GHz, about 1 GB of RAM and were running Windows XP Professional, Python 2.5.4 and VIFF 0.7.1. This would be a typical setup if all authors on a scientific paper were from the same institution.

The benchmarks were run with $n = 3, 4, \dots, 8$ while the threshold was kept on $t = 1$. This was due to the included sorting algorithm not working properly with other threshold values. Although this “bug” would be a security problem, it is not important for the benchmark tests. 8 participants were considered appropriate, as more authors on a scientific paper is rare. Another problem for me was to get hold of this amount of computers.

Table 7.2 shows the average computation time of the author application. The computation consists of adding up a sum of points for each player before finally sorting an array list with n elements. The sorting function has a total number of comparisons that is $O(n \log^2 n)$, which is what causes most of the time consumed.

From the column *Time/comp* we see that the time per comparison drops

Chapter 7. Application 1: Rank the Authors

rapidly at first, and is still decreasing for $n = 8$. This corresponds to the performance results given in Section 4.8 where the time per comparison approached 800 ms for $n > 10$. Notice that for $n = 4$, the time per comparison is notably faster than for $n = 5$. The reason for this is not clear, and would require a more thorough analysis of the comparison protocol.

(n, t)	Average time (s)	Comparisons	Time/comp (s)
(3,1)	7,03	3	2,34
(4,1)	7,91	6	1,32
(5,1)	13,27	9	1,47
(6,1)	16,58	13	1,28
(7,1)	20,38	18	1,13
(8,1)	24,52	24	1,02

Table 7.2: Benchmark results for the author application.

In Figure 7.6 the actual number of comparisons, the upper bound (big O) growth rate of the number of comparisons, and the average time for various values of n are plotted. We see that the the *Time* would probably cross the *Comparison* line if higher values of n were used.

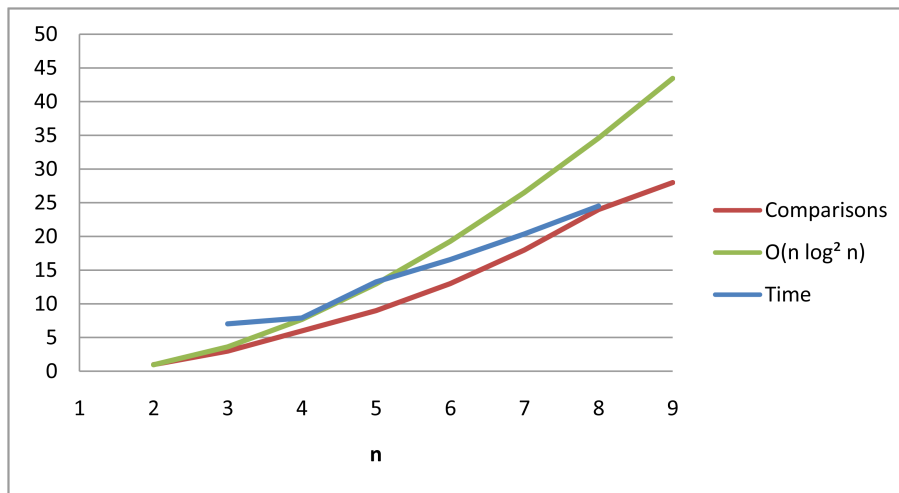


Figure 7.6: Actual number of comparisons, the upper bound growth rate, and the average computation time used are plotted for various values of n .

To conclude, it is obvious that comparison of shared values is a time-consuming task. For this specific author application it would probably not cause any problems. However, using such a scheme for bigger applications would be impossible. This does not scale very well!

7.3 Possible Improvements

The author application works for its intended purpose and could easily have been used to decide the order of authors in a scientific paper. Still there are a few aspects which could be improved.

7.3.1 Calculation of Points

With the current point rating system, where 1th author results in $n - 1$ points, 2th author in $n - 2$ points etc., people could end up with the same amount of points. The application does not handle this incident specifically, but the bitonic sort algorithm implemented will provide a default solution. If two persons receive the same amount of points, the person nearest the bottom of the `author_config.txt` file will be ranked first.

Since this is mostly a proof-of-concept application, no further solutions have been implemented, but one could imagine the following solutions:

- Decide one player who's vote will count extra if two players receive the same amount of points.
- Introduce a second round of voting between those players with equal amount of points.

7.3.2 Complicated Application Launch

Although the graphical interface makes it easy to understand how to vote, the process of installing Python and VIFF, distributing configuration files and finally execute the program through the command line is overly complicated. In addition, all participants have to execute the application and give their vote in about the same space of time, as there are no concept for storing the votes.

One could imagine setting up the author program as a web application, where one player puts up the vote. Then the participating players can visit a specific URL and vote through their web browser, without the need to install a lot of other programs. Such a scheme was designed through the *Secure Web Voting* application (see Chapter 8), but it is more a general poll than a tool for paper authors.

Chapter 8

Application 2: Secure Web Voting

The author application in Chapter 7 worked as intended, but required a lot of programs and packages to be installed in order to run. To improve these complications a new voting scheme was developed, focusing on making it easiest possible to deliver a vote. Instead of each player running VIFF and executing the multiparty computation protocol, these tasks are delegated to three computation servers. The only required step for each participant is to visit a unique URL from a normal web browser.

8.1 Design and Implementation

Figure 8.1 illustrates the main idea behind the web application. The participants connect to the web server and deliver their votes via a Java applet. The web server will receive the votes and save them to a database. When all the participants have answered, a message will initiate the secure computation among the three servers P1, P2 and P3.

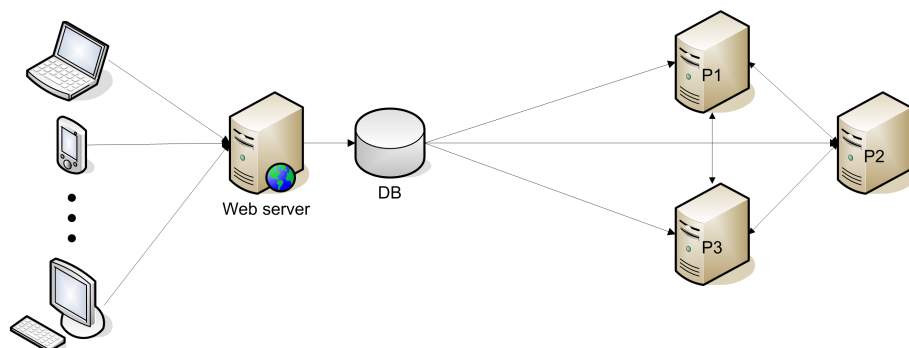


Figure 8.1: The architecture of the web application.

Chapter 8. Application 2: Secure Web Voting

Note that the three computation servers each have a public/private key pair. When a participant delivers a vote, it will be secret shared and encrypted with the public key of the computation servers. At last the entire set of ciphertexts is stored in a database.

The vote of the last participant will trigger a message to all the computation servers. They will connect to the database, fetch their respective entries, decrypt the ciphertexts with its private key, and finally compute the result from the shares.

8.1.1 Screen Shots

Next follows some screen shots of the sequence of events needed to carry out a vote. Note that the system should be available and possible to check out at any time. The vote setup page at <http://folk.ntnu.no/havardv/smpc/> is guaranteed to work as long as NTNU's web systems are up and running. The Java applet, where the actual voting takes place, is located at the same place.

The uncertainty lies within the computation servers, since they need to be online and running at the workplace at NTNU. I cannot guarantee for these computers to be available in the future.

Setting up the Vote

The first step is to create the vote itself. By visiting the above URL, a standard web page containing different form elements will appear. As seen in Figure 8.2 the first step include options like:

- Title
- Description
- Your name (the author of the specific vote)
- Your e-mail
- Number of participants, possibly up to 10
- Voting options, possibly up to 9 choices

– Poll options (step 1 of 2) —————

Provide a meaningful title and more detailed information in the description. In step 2 (press Next) you will fill in the e-mail addresses of the chosen number of participants.

Title:

Description:

Your name:

Your e-mail:

Number of participants:

Voting options/candidates:

Option 1:

Option 2:

Option 3:

Option 4:

Option 5:

Option 6:

Option 7:

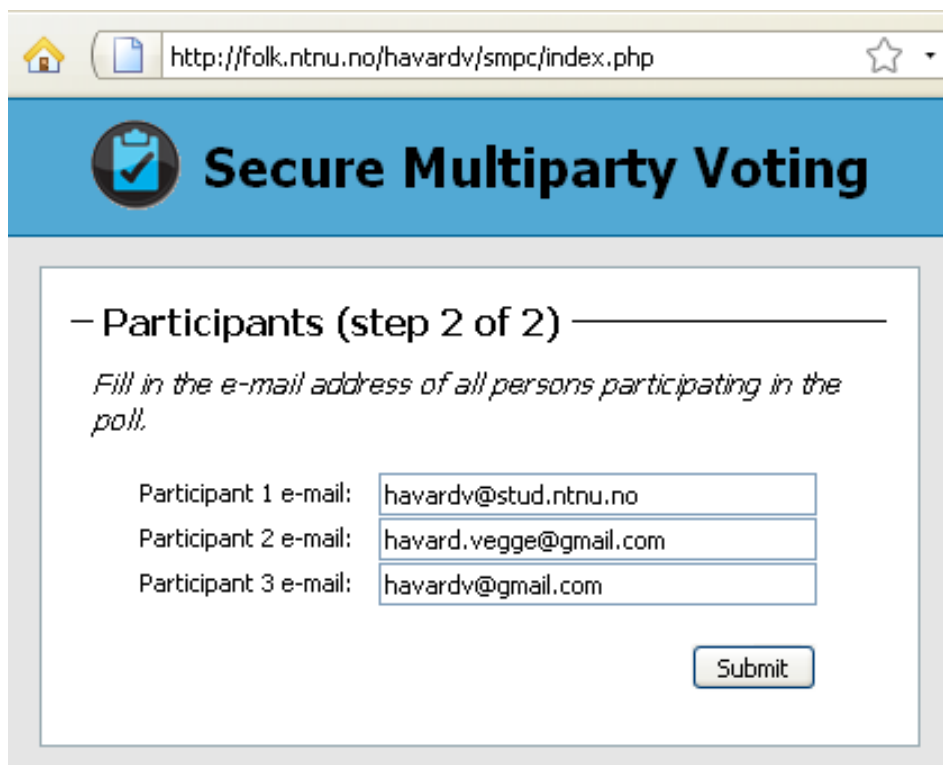
Option 8:

Option 9:

Figure 8.2: Step 1 for creating a poll on the web page.

Chapter 8. Application 2: Secure Web Voting

The next screen (step 2) asks for the e-mail addresses to the number of persons participating in the vote. If three participants were selected, the page would look like the one in Figure 8.3. After pressing the submit button, each participant will receive an e-mail like the one listed in Figure 8.4.



— Participants (step 2 of 2) —

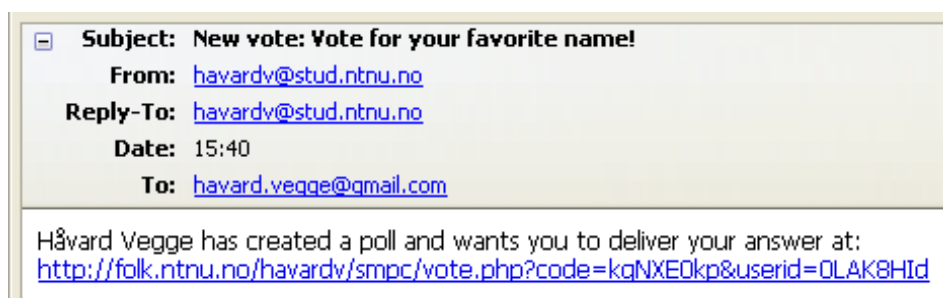
Fill in the e-mail address of all persons participating in the poll.

Participant 1 e-mail:

Participant 2 e-mail:

Participant 3 e-mail:

Figure 8.3: Step 2 for creating a poll on the web page.



Subject: New vote: Vote for your favorite name!
From: havardv@stud.ntnu.no
Reply-To: havardv@stud.ntnu.no
Date: 15:40
To: havard.vegge@gmail.com

Håvard Vegge has created a poll and wants you to deliver your answer at:
<http://folk.ntnu.no/havardv/smpc/vote.php?code=kqNXE0kp&userid=OLAK8HId>

Figure 8.4: An example e-mail received by a person participating in a vote.

The author of the vote will in addition receive a confirmation e-mail reporting that the vote was created and that e-mails have been sent to all participants.

Deliver a Vote

As mentioned each participant will receive an e-mail with a unique link. Several “random” sequences of letters and numbers are created to identify the poll and each of the participants. The Java applet where the actual voting takes place is illustrated in Figure 8.5.

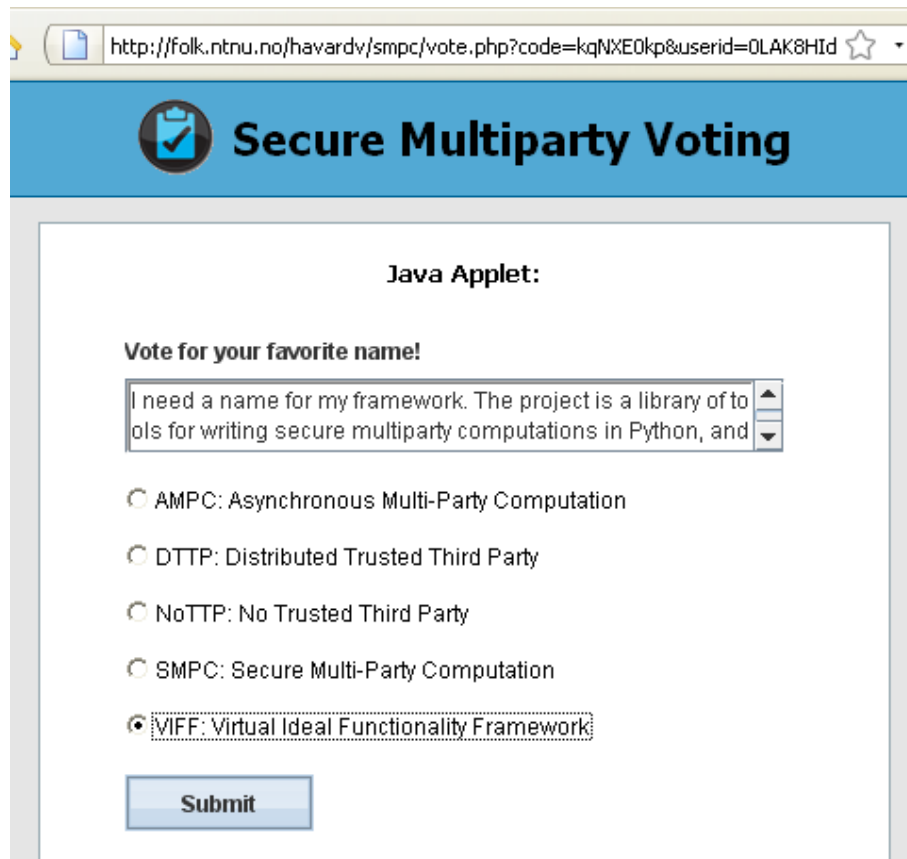


Figure 8.5: Java applet for delivering of a vote.

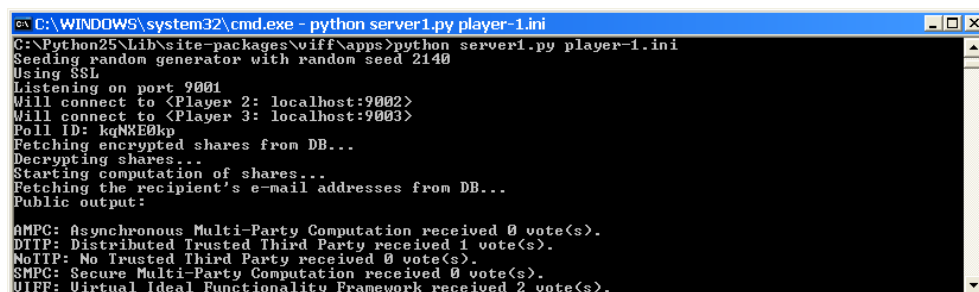
When pressing the submit button in the Java applet, the chosen value will be secret shared and encrypted. These values are then sent via HTTP POST to a PHP script on the web server which stores the representations of the vote in a database.

Calculate the Result

When all participants have voted, an XML-RPC message is automatically sent to all of the three computation servers initiating the computation. The result is then e-mailed to all participants, while nothing else is revealed. Figure 8.6 shows how one of the servers acts during the session. Also note

Chapter 8. Application 2: Secure Web Voting

that it is immediately ready for new XML-RPC messages and computation of votes from different polls.



```
C:\WINDOWS\system32\cmd.exe - python server1.py player-1.ini
C:\Python25\Lib\site-packages\uiff\apps>python server1.py player-1.ini
Seeding random generator with random seed 2140
Using SSL
Listening on port 9001
Will connect to <Player 2: localhost:9002>
Will connect to <Player 3: localhost:9003>
Poll ID: kqNKE0kp
Fetching encrypted shares from DB...
Decrypting shares...
Starting computation of shares...
Fetching the recipient's e-mail addresses from DB...
Public output:
AMPC: Asynchronous Multi-Party Computation received 0 vote(s).
DTTP: Distributed Trusted Third Party received 1 vote(s).
NoTTP: No Trusted Third Party received 0 vote(s).
SMPG: Secure Multi-Party Computation received 0 vote(s).
UIFF: Virtual Ideal Functionality Framework received 2 vote(s).
```

Figure 8.6: The console of one computation server after it has calculated the result of a vote.

8.2 Architecture

As the voting system consists of several entities, the different source files and database tables are on various locations. Note that Figure 8.7 is not a correct UML diagram, but it illustrates where the different classes and files belong.

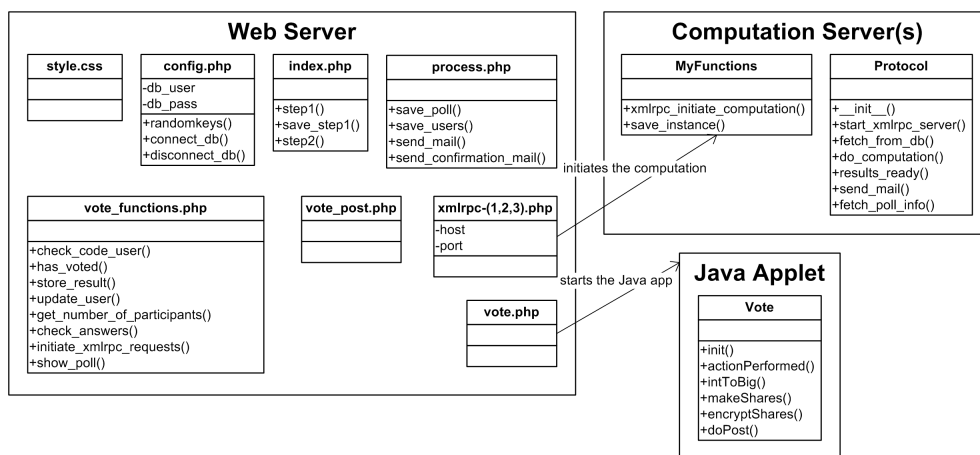


Figure 8.7: Classes and files of the web application.

8.2.1 Web Server

The web server is where all users direct their communication in order to set up or deliver a vote. Although the files that constitutes the web server is placed in the same location, they can be divided into three different entities.

The PHP scripts are run by the server, while the Java applet executes locally on the clients' machines. The MySQL database is located on some other server at NTNU.

PHP Script

The web page where the voting takes place is built up with HTML and PHP. See Table 8.1 for a brief explanation.

File	Description
style.css	Style sheet for controlling the style and layout of multiple web pages at once.
config.php	The config file contains the username and password to the MySQL database. Note that this file is only accessible by the web server. In addition there are functions for connecting and disconnecting the database together with a function for creating "random" sequences of letters and numbers.
index.php	The index file holds all the forms for creating a poll. When all information is gathered, the values are dispatched to <code>process.php</code> .
process.php	The process file connects to the database and stores the poll together with the participating users. E-mails are also sent out.
vote.php	The vote file checks the incoming variables. If a valid user asks for an existing poll, the Java applet is showed. The steps performed by the Java applet are further described in the next section.
vote_functions.php	This file contains a lot of functions regarding the voting, <code>show_poll()</code> and <code>store_result()</code> to mention a few. These functions are accessible to the other PHP scripts.
vote_post.php	The <code>vote_post</code> file receives the encrypted shares from the Java applet through an HTTP POST request. It verifies that the answer is valid and stores the result in the database. If all participants have voted, the three files <code>xmlrpc-(1,2,3).php</code> will be executed.
xmlrpc-(1,2,3).php	These three files are identical except for the <code>\$host</code> and <code>\$port</code> variables. Each file will send an XML-RPC request to the corresponding computation server initiating the computation of the poll.

Table 8.1: List of PHP files in the secure web voting application.

Chapter 8. Application 2: Secure Web Voting

Java Applet

The Java applet takes in parameters like title, description and the options one can vote for. A GUI is then built and showed to the user. When the user submits his vote, the selected option is secret shared. The calculation of the shares is shown in Listing 8.1. The variable `id` is just the index of one of the radio buttons, for instance would the first button yield the number 0. The variable `random` is a big integer between 0 and $2^{160} - 1$.

```
public void makeShares(int id) {  
  
    ...  
  
    share1 = id + random * 1 (mod p);  
    share2 = id + random * 2 (mod p);  
    share3 = id + random * 3 (mod p);  
}
```

Listing 8.1: The essential part of the `makeShares` function in the Java applet.

Next the three shares are encrypted using RSA and one of the computation servers' public keys. Share 1 will be encrypted using computation server 1's public key, share 2 with computation server 2's public key and finally share 3 with computation server 3's public key. The encryption shown in Listing 8.2 is done with 1024 bits RSA keys.

```
public void encryptShares(BigInteger share1, BigInteger share2,  
    BigInteger share3) {  
    BigInteger n1 = new BigInteger("9681521607...93");  
    BigInteger n2 = new BigInteger("9186864592...29");  
    BigInteger n3 = new BigInteger("1159172490...09");  
    BigInteger e = new BigInteger("65537"); // 2^16+1  
  
    BigInteger c1 = share1.modPow(e, n1);  
    BigInteger c2 = share2.modPow(e, n2);  
    BigInteger c3 = share3.modPow(e, n3);  
}
```

Listing 8.2: The RSA encryption of shares in the Java applet. The public keys have been abbreviated.

When the encryption is finished, the values are transmitted back to the PHP script which stores the three values in the database. These values make no sense even if they had been captured on the way to the web server.

The reason why this operation (i.e. secret sharing and encryption) is performed through a Java applet and not just in the original PHP script is

based on security. A PHP script is executed server-side and a vote would thus be transferred in clear text over the Internet from the client to the web server before it is secret shared. By using a Java applet, this problem is bypassed, since it is run locally by the voter. The vote is first secret shared, and then each share is encrypted using RSA, before transmitting the content.

Another question is, assuming a Java applet is used, why not store the encrypted shares directly from the applet through Java's JDBC¹ instead of returning the values to the PHP script? If the Java applet itself were to connect to the database and store the results, it would have needed the database user and password. Preferably user id and password should remain safely on the server, not in a client-side application. This is the reason why the Java applet does not use JDBC, but instead transfers the encrypted shares back to the PHP script, and it is the PHP server that connects to the MySQL database and stores the values.

Database

Three tables are set up in a database to keep track of the polls, users and encrypted shares. Figure 8.8 shows the columns of each table. *PK* is short for primary key while the *I* is an abbreviation for index. A database index improves the speed of data selection. Especially when the database query contains a *WHERE* clause an index on that column would lessen the workload the server must endure to perform the search.

poll		users		result	
PK	<u>id</u>	PK	<u>id</u>	PK	<u>id</u>
I1	code title description author author_email participants participants_email options	I1	code	I1	code
		I2	user_id has_voted		share1 share2 share3

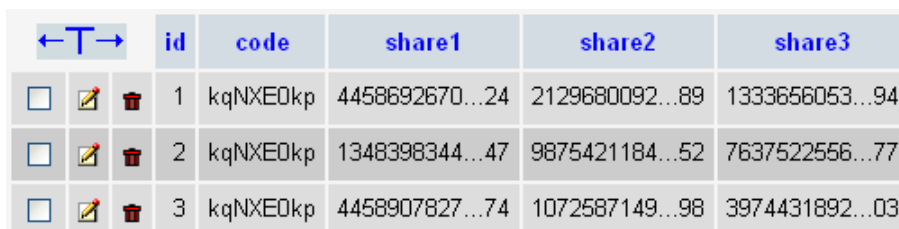
Figure 8.8: The three database tables where information about polls, users and results are stored.

To illustrate what is actually stored in the database, a snapshot of the table *result* is given in Figure 8.9. The values in the column *code* are identical

¹JDBC is an API for the Java programming language that defines how a client may access a database.

Chapter 8. Application 2: Secure Web Voting

meaning all rows belong to the same poll. The three shares is a representation of a vote, but needs to be decrypted and reconstructed in order to mean anything.









	id	code	share1	share2	share3
<input type="checkbox"/>  	1	kqNXE0kp	4458692670...24	2129680092...89	1333656053...94
<input type="checkbox"/>  	2	kqNXE0kp	1348398344...47	9875421184...52	7637522556...77
<input type="checkbox"/>  	3	kqNXE0kp	4458907827...74	1072587149...98	3974431892...03

Figure 8.9: The MySQL table *result*, containing the encrypted shares, as shown in phpMyAdmin.

8.2.2 Computation Servers

Each of the three computation servers holds a file `server-i.py`, where *i* is the number 1, 2 or 3. This file contains the VIFF code and also provides the functionality of an XML-RPC server.

When a message is received from the web server reporting that all users in a vote have answered, each computation server connects to the database and fetches the belonging shares. Computation server 1 will fetch all values from the *share1* column, computation server 2 will fetch all values from the *share2* column and finally computation server 3 from the *share3* column. Each of the servers has a unique secret key and thus is able to decrypt its shares. By executing the VIFF protocol the three servers calculate the result of the poll and they also notifies all participants by e-mail.

8.3 Libraries

In addition to VIFF there are several libraries and modules helping to build all functionality of the system. This section will outline two of the most important, namely *SecureRandom* and *XML-RPC*.

8.3.1 SecureRandom

SecureRandom is a Java class which provides a cryptographically strong pseudo-random number generator (PRNG) [11]. When the Java applet creates the shares, a big random integer is needed, and *SecureRandom* helps in this process. Listing 8.3 demonstrates how the class is used to create a random number between 0 and $2^{160} - 1$. This number is again used in the

creation of the shares.

```

BigInteger random = BigInteger.ZERO;

SecureRandom secRandom;
try {
    secRandom = SecureRandom.getInstance("SHA1PRNG");
    random = new BigInteger(160, secRandom); // 160 bits
}
catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}

```

Listing 8.3: Java code demonstrating how to create random integers.

8.3.2 XML-RPC

XML-RPC is a specification and a set of implementations that allow software running on dissimilar operating systems or different environments to make procedure calls over the Internet [17]. It uses XML to encode its calls and HTTP as a transport mechanism. XML-RPC has evolved into what is now called SOAP, but many people prefer XML-RPC because of its simplicity and ease of use. The concept is illustrated in Figure 8.10.

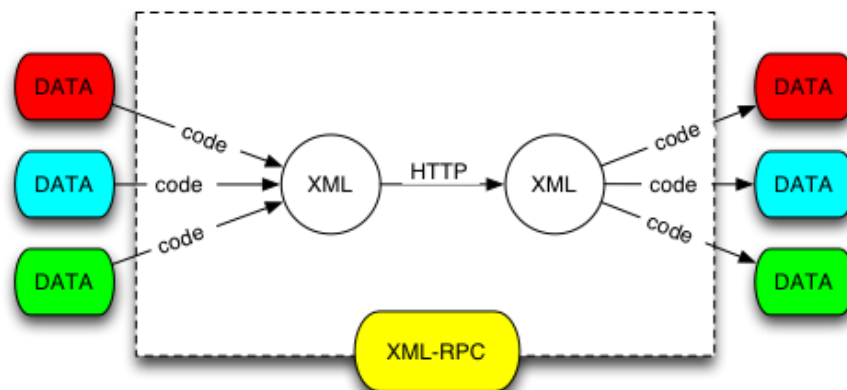


Figure 8.10: XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism [10].

In the voting application, XML-RPC is used for the communication between the PHP script at the web server and the Python/VIFF script at the computation servers. When all participants of a vote have given their answers, an XML-RPC message will be transmitted. Listing 8.4 shows the request message sent from the web server to the computations servers. It contains

Chapter 8. Application 2: Secure Web Voting

only a single method `initiate_computation` with a code variable.

```
POST / HTTP/1.0
User-Agent: XML-RPC for PHP 2.2.2
Host: 129.241.209.179:8001
Accept-Encoding: gzip, deflate
Accept-Charset: UTF-8,ISO-8859-1,US-ASCII
Content-Type: text/xml
Content-Length: 170

<?xml version="1.0"?>
<methodCall>
  <methodName>initiate_computation</methodName>
  <params>
    <param>
      <value><string>XbIRNvMd</string></value>
    </param>
  </params>
</methodCall>
```

Listing 8.4: The XML-RPC request message sent from the web server to computation server 1.

8.4 Security Analysis

Section 5.5 listed several security requirements that a secure and complete cryptographic voting protocol should satisfy. An e-voting system for use in a national election obviously has stronger requirements than this web application. Nevertheless the security analysis is performed with respect to these eight requirements.

8.4.1 Voter Privacy

The authentication is made very easy in the web voting system. Each participant receives a unique identifier consisting of “random” letters and numbers through their e-mail. The identifier and the e-mail are though not linked together in the database, thus nothing can be discovered about a participant. Information about IP address etc. are not stored.

The actual vote will consist of the tuple (`code`, `share1`, `share2`, `share3`) and even if someone were to break into the database, they would not find out which of the participants delivered the vote.

8.4.2 Eligibility

The requirement of non-eligible voters not being able to vote is satisfied through the same “random sequences”. One could, using brute force, try to

guess a *poll_code* and a *user_code*, but this would be a time-consuming task. The random sequence consists of 8 characters, each which have 62 possible values. As a result one would have to try $8^{62} = 9,807971461541689e + 55$ values (187 bits), or on average the half of this, to find a specific poll. In addition one would also have to guess the *user_code* which is of the same length.

A possible improvement would be to let each user generate a pair of private/public keys and this way be verified against a registrar or similar. Such a setup would be a lot more complex and not very practical for a simple and easy web voting such as this one.

8.4.3 Uniqueness

Uniqueness is about making sure only one vote is counted for each eligible voter. As already mentioned, each participant receives an unique identifier which is stored in the database. When this user delivers its vote, the table `users` in the database will set the belonging column `has_voted` to 1. If the same user visits the poll once more, it will be notified that *“It is not allowed to vote more than once!”*

8.4.4 Fairness

To ensure all candidates/choices get a fair decision, no partial tally must be revealed before the end of the voting. This is the case in the web application since nothing is calculated before the last participant has voted. Not until then will a message be sent to the computation servers which start the process of calculating the result.

As mentioned each vote is secret shared into three shares. These are again encrypted with the public key of one of the computation servers. As a consequence, no one except the computation server with the corresponding private key is able to decrypt the share.

Security of the Shares

The shares in the Java applet are made with the following equation:

$$share = s + random * x(\text{mod } p)$$

The variable x lies in the range $[1,3]$, s in the range $[0,9]$ (number of choices/candidates) and $random$ in the range $[0,p)$ where p is 2^{160} . The modulus operation provides perfect security in the sense that the share can be any element from 0 to p and thus this share alone does not supply any additional information about the secret s . A brute force attack would now be in the order of 2^{160} .

RSA Encryption

In the web voting application all shares are stored in the same database. If someone had managed to break into the database, all shares would be revealed and the system proven useless. That is why the shares are encrypted with RSA. As of today, the largest known number factored by a general-purpose factoring algorithm is 663 bits long [8]. The current recommendation is that n should be at least 1024-2048 bits long. The Java applet is using 1024 bits which RSA Security claims that are equivalent in strength to 80-bit symmetric keys.

8.4.5 Uncoercibility

Participating voters are able to vote freely, without anyone able to coerce them into casting a vote in a particular way.

In addition, no authority should be able to extract the value of a vote. Assume that someone were to get hold of the private key of one computation server. It would then be possible to decrypt one share, but that would not be enough to reconstruct the vote. In fact, knowing one share is not helpful at all.

Also remember from Section 8.2.1 that the Java applet is executed locally. The creation of shares and the subsequent encryption is only “visible” for that user. When the values are sent to the web server, they have no meaning except for the set of cooperative computation servers, thus confidentiality is achieved.

8.4.6 Receipt-freeness

The application has no concept of confirmation or receipt of the votes other than a string reporting “*Answer stored in database... you will receive an e-mail when the results are ready!*” Leaking of information regarding the vote because of receipts is thus non-existent, thus discouraging vote-buying or coercing.

8.4.7 Accuracy

Assuming the simple authentication scheme based on “random” letters and numbers as described in Section 8.4.2 about eligibility holds, the final output of the vote should be correctly computed.

Countermeasures against attacks like SQL-injection etc. have been implemented, but other security precautions regarding the MySQL database are not employed. If an adversary had gained access to the database, he could

have deleted or modified the encrypted shares. He would however, not manage to derive the content of the votes.

The multiparty computation itself is secure against passive adversaries for up to $1/2$ of the computation servers.

8.4.8 Individual Vote Check

It would be desirable if a voter could be able to check that his encrypted vote was counted and tabulated correctly in the final tally. The problem is that the combination of receipt-freeness and individual verifiability requirements obviously conflict with each other. One would need some kind of receipt in order to achieve verifiability, but then the same receipt could be used for vote buying or selling. As a consequence this voting application has given preference to receipt-freeness and thus no functionality for verifying votes later in the process are implemented.

8.5 Possible Improvements

The *Secure Web Voting* application was successfully implemented and illustrates a more practical way of using multiparty computation. However, it is still just a proof-of-concept application and several aspects could have been improved. Below are some of the security parameters that could have been adjusted, further discussion is given in Section 9.

- The “random” sequence of letters and numbers are of size 8. They could easily be increased from just 8 characters to a bigger number if desired.
- The size of the RSA keys used for encryption in the Java applet is 1024 bits. According to security recommendations, it should in practice be increased to at least 2048 bits, as 1024 bits RSA may become breakable in the near future.
- Investigate the possibility of fulfilling both receipt-freeness and verifiability.
- In the current application it is only computation server 1 that sends out e-mails of the voting results. Functionality that distributes and synchronizes this responsibility between the servers should be done. Such an important task should not be performed by a single entity alone.

Chapter 9

Discussion

This chapter will evaluate the *Secure Web Voting* application and the required steps in order to build an application in VIFF. Section 9.3 discusses the potential of secure multiparty computations, while some ideas for further work are given in Section 9.4.

9.1 Secure Web Voting Application

While the *Rank the Authors* application was overly difficult to execute, the *Secure Web Voting* application was successfully implemented to solve this problem. Instead of each participant executing the secure multiparty computation, this task was delegated to three computation servers. The only required step for a participant is to visit a web page (the poll) with a normal web browser.

The application illustrates how MPC can be used to simulate a trusted third party, but some issues would have to be solved in order to trust the application. All computation servers should not be run by one authority and this issue is addressed in Section 9.1.1 below. Another scheme for storing the shares is described in Section 9.1.2.

9.1.1 Location of the Computation Servers

In the proof-of-concept application, all computation servers are run locally on one machine. This eases the development and testing, but in a real-world scenario, these computers should be independent and run by different authorities. One could for instance be run at NTNU, one with the VIFF development guys in Denmark and one at a university in the USA. That way no party would have access to all the information and nobody could be able to cheat.

9.1.2 Storing of Shares

In the current application all encrypted shares are stored in the same database. Another scheme could be to let each computation server hold its own database. The share for computation server 1 would thus only be stored in the respective database and impossible for the other computation servers to access.

One could still encrypt the shares, but it would not be absolutely necessary, if the transfer is assumed to take place through a secure channel. The main reason why the shares are encrypted in the current application is to avoid all three shares from being stored in cleartext at the same location. Then the choice is to encrypt the shares or store them at separate locations. The latter is illustrated in Figure 9.1.

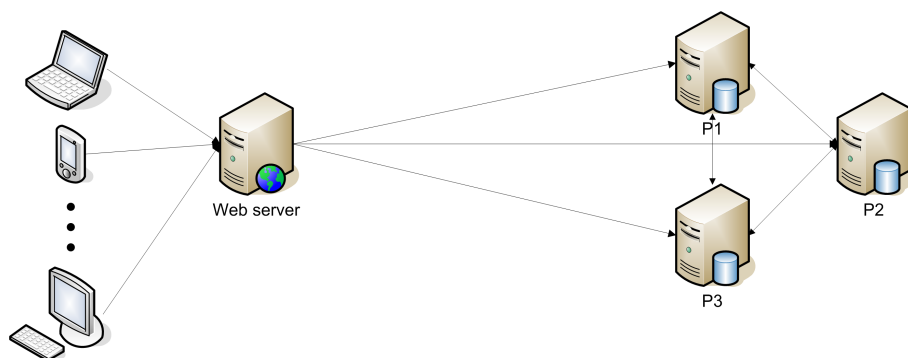


Figure 9.1: Web application scheme which stores the shares at different locations, that is, at P1, P2 and P3.

Note that the three shares should never be at the same location, other than at the client-side. In the figure above it seems like the web server will get all the shares, but in an implementation it should be a Java applet (or another client-side application) that transmits the shares to the computation servers directly.

9.2 VIFF

The author's of VIFF claim that the framework allows you to specify secure multiparty computations in a clean and easy way. However, it is a relatively new framework and except for the developers, very few people have built applications using VIFF so far. As a consequence, there exists little objective review of the framework, in contrast to existing and widespread frameworks. This section describes which steps and amount of background theory that are needed to build applications using VIFF.

9.2.1 Necessary Steps

In order to develop applications using VIFF one needs to install VIFF and several belonging packages. The procedure for MS Windows is described in detail in Appendix B, refer to the installation guide [12] to install on Mac OS X and GNU/Linux.

On Windows at least 6 different components¹ need to be installed. This is overly complicated, especially if the goal is to make VIFF a popular tool. A single file which could install all the dependencies needed by VIFF automatically would have been great. This is an issue which the developers are aware of, but yet a solution is missing, and for the time being the installation process is not very user-friendly to newcomers.

9.2.2 Development

Once VIFF is installed, development is fairly easy. By inspecting one of the enclosed example applications (for instance the VIFF implementation of the millionaire problem), one understands how to structure the code elements. Creating similar and simple applications is thus trivial (at least if you know the Python programming language from before).

The problem occurs when deviating from the example applications, because as a new VIFF developer you probably have difficulties understanding what functions like `runtime.shamir_share()`, `runtime.open()` and `gather_shares()` really is. There is a documentation section on VIFF's web page, but I would love to see a more comprehensive and easy explained documentation guide, at least to invite other than MPC experts to try it out.

9.2.3 Required Background Theory

As mentioned, one could easily create a simple VIFF application that does some computation on a number of input values. Another question is how much one understands of the program. If someone is eager enough to develop an application in VIFF, they will probably want to understand why it is secure and how the calculations are done.

As VIFF uses multiparty computations and Shamir secret sharing, these two main theory pillars are essential to understand the framework. As seen from Chapter 2 and Chapter 3 the background theory is pretty extensive, especially when diving into the mathematics of addition and multiplication.

¹Python, Twisted, GMPY, OpenSSL Win32 Installer, PyOpenSSL and VIFF.

Also in VIFF there are aspects which are not self-explanatory. An essential building block is for instance the event-driven network engine Twisted and its use of deferred objects. In order to create more complex applications all this needs to be understood.

9.2.4 Future Potential

In spite of the somewhat troublesome installation process, VIFF is very promising. As pointed out in the introduction in Section 1.3, multiparty computations are said to have great potential, but very few applications have emerged except for in scientific papers. This could change due to frameworks like VIFF. Especially for people with some knowledge of multiparty computations, this framework makes it trivial and fast to develop applications.

The first large-scale and practical application of multiparty computation, which took place in January 2008, was run by the SIMAP project [BCD⁺08]. The double auction system for trading sugar beets among Danish farmers was a success and is still in use. As VIFF grew out of the same project and have implemented the same functionality (and more), there is no reason why VIFF will be less successful. VIFF is still in its starting line, and several efficiency issues are constantly under development, but it has the possibility of becoming very good. Of course, it depends on the developers continuing their good work. Also, multiparty computations are not a well-known concept to the general public, and thus there will be a challenge to inform about the potential.

9.3 The Potential of Multiparty Computations

Secure multiparty computations have many potential applications, for instance secure auctions, privacy-preserving benchmarking, e-voting and privacy-preserving data mining [CK08]. However, the research has been mainly focused towards cryptographic methods for building secure multiparty computation protocols and formally proving the security properties of those protocols. As it turns out, few of these protocols have emerged as practical applications. Two of the main problems can be identified as:

1. Poor performance, functionality and scalability.
2. Lack of understanding about the potential of MPC among the general public.

If a trusted third party (TTP) is available, it can be very simple and efficient: the TTP privately collects the secret inputs, performs the calculation and

delivers the output to each party. Such a protocol will fulfill privacy and correctness requirements if followed strictly. In real-life a provider offering this kind of guarantees might not be available or is too expensive. In such cases, multiparty computation could replace the TTP. There is still much work to be done regarding the aspect of practical implementations, but there is no doubt that multiparty computations have potential to be used in a lot more applications.

9.4 Further Work

This section lists some ideas for future work.

9.4.1 Other Applications

In the first phase of this master's thesis several possible applications were identified. I chose to focus on making a practical application available to the public via the World Wide Web. Some of the other ideas are listed below and could perhaps be interesting to implement in the future:

- Distributed RSA (Atle Mauland implemented this).
- Shortest path problems (Dijkstra).
- Privacy-preserving database queries.
- Calculation of bad mortgage in Norwegian banks or similar problems.
- Position determination without GPS, see Tord Ingolf Reistad's paper *Multi-party Secure Position Determination* [Rei06]. This was actually implemented as a test program and the code is included in the attached ZIP file (see Appendix E).
- See the article by CACE [Pin08] for more application ideas.

9.4.2 Improve and Deploy a Web Voting Application

The *Secure Web Voting* application was successfully implemented, but since it was mainly a proof-of-concept application, some issues were identified in Section 8.5. These issues could be fixed or one could make a completely new voting application, based on the same ideas. However, the application should be deployed and ran by three different and independent computations servers. It could give multiparty computations more focus and good publicity.

Imagine the creation of a web site similar to Doodle [1], just that the new service would provide better privacy through the use of multiparty computations.

9.4.3 Large-scale E-voting

As Norway are planning the introduction of e-voting in 2011, it would be interesting to evaluate how a voting scheme in such a large scale could be done using MPC and VIFF. One challenge would of course be that all the security requirements listed in Section 5.5 and most likely some additional nation-specific requirements would have to be provided.

Another issue which is not addressed in this thesis is how such a voting scheme would perform in a bigger scale. The current database, VIFF program and the connection between them should handle 10,000 votes, maybe 1,000,000. Also, the current application does not perform any complex arithmetic, something that should be implemented and further investigated.

Chapter 10

Conclusion

In this thesis the Virtual Ideal Functionality Framework (VIFF) has been used for implementing practical applications based on multiparty computation (MPC). VIFF has proven to be a feasible and easy framework, thus accelerating the change from just theoretical protocols into hopefully a lot more implemented programs.

The work started with a theoretical study of secret sharing and multiparty computation with emphasize on giving illustrative figures and clarifying examples. VIFF, its functionality and mode of operation were also thoroughly investigated. In addition, an overview of electronic voting, challenges and typical security requirements have been briefly examined.

The background study led to the development of the first application, a small GUI program in which participants on some scientific paper can vote for their choice of 1th, 2th, 3th, etc. author. Although successfully implemented, several issues were identified. In order to execute the application, several additional components would have to be installed. This is overly complicated for a program that originally should solve an easy problem. A second problem was that it did not scale very well. Also, all participating voters would have to run the application at the same time, a requisition that is very impractical.

After evaluating the author application, a different scheme for doing MPC voting was designed. It solved all the problems present in the first application, and also took a step further into becoming a realistic way of using MPC in practical applications. A *Secure Web Voting* application was implemented, but instead of each player running VIFF and executing the MPC protocol, these tasks were now assigned to three computation servers. For a participating voter's point of view, the only requirement is to visit a web page (URL) from a normal web browser. This is also a scheme which could

Chapter 10. Conclusion

be used for bigger applications, like nation-wide elections, but this would probably require even stronger security requirements. If used in this context, a more thorough analysis of scalability would have to be performed, but this has been considered out of the scope of this thesis. Looking at the double auction application developed through the SIMAP project, in which over 1000 Danish farmers each submitted 4000 bids, an indication of the potential is given [BCD⁺08].

As described above, SIMAP, which is the predecessor of VIFF, obviously worked and had its uses. VIFF has implemented the same functionality and more, thus the potential for becoming a great framework is certainly present. For the development of this thesis' applications VIFF proved as an easy and intuitive tool. However, for the framework to stay alive, continued improvements and development are needed.

The classical theory of MPC shows great potential, but efficiency has been a problematic issue. Addition of shares is very fast, multiplications a bit slower. Comparisons are still very inefficient though, as experienced in the *Rank the Authors* application. An estimate of 800 ms per comparison makes this impractical for applications that cannot afford some delay. As a consequence, one must evaluate what a specific application needs of arithmetic operations, in order to discover how feasible a successful implementation is.

To conclude, I hope this thesis' work can stimulate other people into creating more practical applications, and hopefully put some of them on the web. The presence of a popular web application based on MPC, accessible for the general public, would certainly demonstrate the power of MPC to a bigger audience.

Bibliography

- [BCD⁺08] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. *Cryptology ePrint Archive, Report 2008/068*, 2008.
- [BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266, New York, NY, USA, 2008. ACM.
- [Bla79] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of National Computer Conference*, 48:313–317, 1979.
- [BLW08] Dan Bogdanov, Sven Laur, and Jan Willemsen. Sharemind: a framework for fast privacy-preserving computations. *Cryptology ePrint Archive, Report 2008/289*, 2008.
- [BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.
- [Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136, Washington, DC, USA, 2001. IEEE Computer Society.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM.

Bibliography

- [CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. *Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation*, pages 342–362. SpringerLink, 2005.
- [CDN08] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation, an introduction. Technical report, Computer science department of Aarhus University, May 25, 2008. <http://www.daimi.au.dk/~ivan/mpc.pdf>.
- [Cet08] O. Cetinkaya. Analysis of security requirements for cryptographic voting protocols (extended abstract). pages 1451–1456, March 2008.
- [Cet09] O. Cetinkaya. Cryptography in electronic voting systems. In *International Conference on eGovernment and eGovernance*, pages 297–310, Ankara, Turkey, March 2009.
- [CHE87] David E. Penney C. H. Edwards. *Elementary Linear Algebra*. Prentice Hall, 1987.
- [CK08] O. Catrina and F. Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. pages 693–700, March 2008.
- [Coc08] Dr. Alistair Cockburn. Using both incremental and iterative development. *CrossTalk - The Journal of Defense Software Engineering*, 21(5):27–30, May 2008.
- [DA01] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, New York, NY, USA, 2001. ACM.
- [Dam06] Ivan Damgård. *Theory and Practice of Multiparty Computation*, pages 360–364. Lecture Notes in Computer Science. Springer-Link, Berlin / Heidelberg, August 2006.
- [DGKN08] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. *Cryptology ePrint Archive, Report 2008/415*, 2008.
- [DGS02] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In *Secure Electronic Voting*, pages 77–100. Kluwer Academic Publishers, 2002.

- [GDP09] Martin Geisler, Ivan Damgård, and Benny Pinkas. Mpc virtual machine specification. Technical report, Computer Aided Cryptography Engineering, January 9, 2009. http://www.cace-project.eu/downloads/deliverables-y1/CACE_D4.3_MPC_Virtual_Machine_Specification.pdf.
- [Gei08] Martin Geisler. Implementing asynchronous multi-party computation - phd progress report. Technical report, University of Aarhus, Denmark, January 2008.
- [GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without a broadcast channel. In *In 16th International Symposium on Distributed Computing (DISC)*, 2002.
- [GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.
- [KLR06] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 109–118, New York, NY, USA, 2006. ACM.
- [LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
- [MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - a secure two-party computation system. In *In USENIX Security Symposium*, pages 287–302, 2004.
- [Opp05] Rolf Oppliger. *Contemporary Cryptography (Artech House Computer Security Library)*. Artech House, Inc., Norwood, MA, USA, 2005.
- [Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. pages 223–238. 1999.
- [Pin08] Benny Pinkas. Applications of secure computation. Technical report, Computer Aided Cryptography Engineering, July 30, 2008. http://www.cace-project.eu/downloads/deliverables-y1/CACE_D4.1_ApplicationsofMPC.pdf.

Bibliography

- [RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multi-party protocols with honest majority. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, New York, NY, USA, 1989. ACM.
- [Rei06] Tord Ingolf Reistad. Multi-party secure position determination. *Presented at the NIK-2006 conference*, 2006.
<http://www.nik.no>.
- [Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [Tof05] Tomas Toft. Secure integer computation with applications in economics - phd progress report. Technical report, University of Aarhus, Denmark, July 2005.
- [Tur66] L. Richard Turner. *Inverse of the Vandermonde Matrix With Applications*. Lewis Research Center, NASA, Cleveland, Ohio, 1966.
- [TW06] Wade Trappe and Lawrence Washington. *Introduction to Cryptography with Coding Theory (Second Edition)*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2006.
- [Usc08] Christian Uscatu. Electronic universal voting. *Informatica Economica*, 48(4):130–135, 2008.
- [Yao82] Andrew C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, pages 160–164, 1982.

Web References

- [1] Doodle AG. Doodle: Easy scheduling, Last accessed May 31, 2009.
<http://www.doodle.com>.
- [2] Twisted Documentation. Choosing a reactor and gui toolkit integration, Last accessed April 28, 2009.
<http://twistedmatrix.com/projects/core/documentation/howto/choosing-reactor.html#auto11>.
- [3] The Alexandra Institute Centre for IT security. Scet - secure computing economy and trust, Last accessed May 19, 2009.
<http://www.aicis.alexandra.dk/uk/projects/scet.htm>.
- [4] The Alexandra Institute Centre for IT security. Simap - secure information management and processing, Last accessed May 19, 2009.
<http://www.aicis.alexandra.dk/uk/projects/simap/index.htm>.
- [5] Michael Frei. Cs513: System security - secret sharing. Technical report, The Department of Computer Science at Cornell University, Last accessed May 15, 2009.
<http://www.cs.cornell.edu/Courses/cs513/2000SP/SecretSharing.html>.
- [6] Martin Geisler. Virtual ideal functionality framework - high-level design overview. VIFF Design Talk at a SIMAP meeting, September 2007.
<http://viff.dk/files/design-talk.pdf>.
- [7] Basic Research in Computer Science (BRICS). Secure multiparty computation language, Last accessed May 19, 2009.
<http://www.brics.dk/SMCL>.
- [8] RSA Laboratories. Rsa-200 is factored!, Last accessed June 9, 2009.
<http://www.rsa.com/rsalabs/node.asp?id=2879>.
- [9] H.W. Lang. Bitonic sort, Last accessed April 28, 2009.
<http://iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm>.

Web References

- [10] Jean-Yves Stervinou. Jy's weblog > jeu 3 juil 2003, Last accessed May 10, 2009.
<http://radio.weblogs.com/0001103/2003/07/03.html>.
- [11] Inc. Sun Microsystems. Securerandom (java 2 platform se v1.4.2), Last accessed May 10, 2009.
<http://java.sun.com/j2se/1.4.2/docs/api/java/security/SecureRandom.html>.
- [12] VIFF Development Team. Installation guide, Last accessed June 4, 2009.
<http://viff.dk/doc/install.html>.
- [13] VIFF Development Team. The history of viff, Last accessed May 1, 2009.
<http://viff.dk/doc/history.html>.
- [14] VIFF Development Team. Viff, the virtual ideal functionality framework, Last accessed May 15, 2009.
<http://viff.dk>.
- [15] VIFF Development Team. Overview, Last accessed May 4, 2009.
<http://viff.dk/doc/overview.html#security-assumptions>.
- [16] Twisted. Asynchronous programming with twisted, Last accessed May 2, 2009.
<http://twistedmatrix.com/projects/core/documentation/howto/async.html>.
- [17] Inc. UserLand Software. Xml-rpc home page, Last accessed May 10, 2009.
<http://www.xmlrpc.com>.

Appendix A

Multiplication Mathematics

A.1 Linear System Approach

Continuing from Section 3.4.4 the players can solve a linear system of equations. Each player can establish three equations using the formula as shown in Equation (A.1).

$$fg(i, j) = s_{i,j} = s_h + r_1j + r_2j^2 \quad (\text{A.1})$$

In Equation (A.1) i refers to the player holding the share and j refers to the player that created the share. Player 1 can do the following calculations:

$$\begin{aligned} fg(1, 1) &= s_{1,1} = 5 \\ fg(1, 2) &= s_{1,2} = -1 \\ fg(1, 3) &= s_{1,3} = -14 \end{aligned}$$

Organizing these values into a matrix yields:

$$\begin{bmatrix} s_h & r_1 & r_2 & 5 \\ s_h & 2r_1 & 4r_2 & -1 \\ s_h & 3r_1 & 9r_2 & -14 \end{bmatrix}$$

Player 1 wants to solve the equations with respect to s_h , which is player 1's share of the total polynomial. Solving the linear system can be done using Gaussian elimination [CHE87] as shown below:

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 1 & 5 \\ 1 & 2 & 4 & -1 \\ 1 & 3 & 9 & -14 \end{bmatrix} & \begin{array}{l} R_2 - 1 \cdot R_1 \\ \implies \\ R_3 - 1 \cdot R_1 \end{array} \begin{bmatrix} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -6 \\ 0 & 2 & 8 & -19 \end{bmatrix} \begin{array}{l} R_3 - 2 \cdot R_2 \\ \implies \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -6 \\ 0 & 0 & 2 & -7 \end{bmatrix} & \begin{array}{l} \frac{1}{2} \cdot R_3 \\ \implies \end{array} \begin{bmatrix} 1 & 1 & 1 & 5 \\ 0 & 1 & 3 & -6 \\ 0 & 0 & 1 & -3.5 \end{bmatrix} \end{aligned}$$

Appendix A. Multiplication Mathematics

Going backwards from row three, all the unknown variables can be calculated:

$$\begin{aligned} r_2 &= -3.5 \\ r_1 &= -6 - 3 \cdot r_2 = -6 - 3 \cdot (-3.5) = 4.5 \\ s_h &= 5 - r_1 - r_2 = 5 - 4.5 - (-3.5) = 4 \end{aligned}$$

The value $s_h = 4$ can be located in Table 3.3. Player 2 and player 3 have to use their values in order to calculate their value of s_h .

A.2 Vandermonde Matrix

Continuing from Section 3.4.4 the players do not need to solve the linear system. By using the inverse of a Vandermonde matrix each player can obtain its share of the total polynomial by means of a less complex calculation. The Vandermonde matrix is defined as [Tur66]:

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}$$

V is the Vandermonde matrix and I is the identity matrix, both of size 3×3 . For three players where $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$ the two matrices are defined as follows:

$$V = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Gauss-Jordan elimination is used to transform $[V|I]$ into $[I|V^{-1}]$.

$$\begin{aligned} \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 4 & 0 & 1 & 0 \\ 1 & 3 & 9 & 0 & 0 & 1 \end{array} \right] & \begin{array}{l} R_2 - 1 \cdot R_1 \\ \implies \\ R_3 - 1 \cdot R_1 \end{array} & \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 1 & 0 \\ 0 & 2 & 8 & -1 & 0 & 1 \end{array} \right] & \begin{array}{l} R_3 - 2 \cdot R_2 \\ \implies \end{array} \\ \\ \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 1 & 0 \\ 0 & 0 & 2 & 1 & -2 & 1 \end{array} \right] & \begin{array}{l} \frac{1}{2} \cdot R_3 \\ \implies \end{array} & \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 3 & -1 & 1 & 0 \\ 0 & 0 & 1 & \frac{1}{2} & -1 & \frac{1}{2} \end{array} \right] & \begin{array}{l} R_2 - 3 \cdot R_3 \\ \implies \end{array} \\ \\ \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -\frac{5}{2} & 4 & -\frac{3}{2} \\ 0 & 0 & 1 & \frac{1}{2} & -1 & \frac{1}{2} \end{array} \right] & \begin{array}{l} R_1 - 1 \cdot R_2 \\ \implies \\ R_1 - 1 \cdot R_3 \end{array} & \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 3 & -3 & 1 \\ 0 & 1 & 0 & -\frac{5}{2} & 4 & \frac{3}{2} \\ 0 & 0 & 1 & \frac{1}{2} & -1 & \frac{1}{2} \end{array} \right] \end{aligned}$$

A.2. Vandermonde Matrix

The tuple $(3, -3, 1)$ in the first row of the inverse Vandermonde matrix will always contain these values when three players are participating and they use the indexes 1, 2 and 3. This gives an advantage for solving the linear systems since no computation on solving the unknown variables needs to be done.

From Table 3.2 player 1 has received the tuple of share values $(5, -1, -14)$ from player 1, 2 and 3, respectively. In order for player 1 to find its share on the total polynomial, the matrix multiplication of the two tuples is calculated:

$$\begin{bmatrix} 3 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ -1 \\ -14 \end{bmatrix} = \begin{bmatrix} 15 + 3 - 14 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix}$$

The value 4 can be located in Table 3.3. Player 2 and player 3 will have to calculate the Vandermonde tuple $(3, -3, 1)$ with their own tuple from Table 3.2 in order to find their shares on the total polynomial.

Appendix A. Multiplication Mathematics

Appendix B

VIFF Installation Guide

The people behind VIFF are offering an installation guide at their web site which explains the necessary steps on Windows, Mac OS X and GNU/Linux [12]. The procedure for Windows is described below and my choice of versions is included in parenthesis.

B.1 Installation Steps

1. Download and install Python (2.5)¹.
2. Include the path to your Python installation (e.g. `C:\Python25\`) in the PATH system environment variable.
3. Download and install Twisted (8.2.0)².
4. Download and install GMPY (1.03)³.
5. Download and execute an OpenSSL Win32 Installer (v0.9.8j)⁴. If required, also install Visual C++ 2008 Redistributables from the same URL.
6. Download and install PyOpenSSL (0.8)⁵.
7. Download and install VIFF (0.7.1)⁶.

¹Python: <http://python.org>

²Twisted: <http://twistedmatrix.com>

³GMPY: <http://code.google.com/p/gmpy>

⁴OpenSSL: <http://www.slproweb.com/products/Win32OpenSSL.html>

⁵PyOpenSSL: <http://pyopenssl.sourceforge.net>

⁶VIFF: <http://viff.dk>

Appendix B. VIFF Installation Guide

B.2 Troubleshooting

If using Windows Vista, right-click on the installers and choose the option to run as administrator.

If a python program reports the error *No module named win32api*, install Python Extensions for Windows⁷.

B.3 Generation of Configuration Files

Execute the following commands from within the directory:

```
C:\Python25\Lib\site-packages\viff\apps
```

to generate configuration files and certificates for three players:

1. `python generate-config-files.py -n 3 -t 1 localhost:9001 localhost:9002 localhost:9003`
2. `python generate-certificates.py -n 3`

B.4 Additional Components

The *Rank the Authors* applications described in Chapter 7 is developed using the GUI toolkit PyGTK⁸. In order to run the application, the following steps are needed (in addition to the VIFF steps):

- Download and install GTK+ runtime (2.12.9)⁹, PyCairo (1.4.12)¹⁰, PyGObject (2.14.1)¹¹ and PyGTK (2.12.1)¹²
- Restart the computer for some changes to take effect.

The *Secure Web Voting* application (actually the computation servers) requires a MySQL package:

- Download and install mysql-python (1.2.2)¹³

⁷Python Extensions for Windows: http://sourceforge.net/project/showfiles.php?group_id=78018&package_id=79063

⁸PyGTK GUI Toolkit: <http://www.pygtk.org>

⁹GTK+ runtime: http://sourceforge.net/project/showfiles.php?group_id=98754

¹⁰PyCairo: <http://ftp.gnome.org/pub/GNOME/binaries/win32/pycairo/1.4/>

¹¹PyGObject: <http://ftp.gnome.org/pub/GNOME/binaries/win32/pygobject/2.14/>

¹²PyGTK: <http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/2.12/>

¹³MySQL support for Python: <http://sourceforge.net/projects/mysql-python>

Appendix C

Source Code Author Application

The source code of the author application is shown below. The rest of the source files are included in an attached ZIP file, see Appendix E.

C.1 author.py

Listing C.1: Source Code author.py

```
1 # The author program imports a list of names (author_config.txt)
2 # These participants can vote on who will be the 1th, 2th, etc.
3 # author one some paper. This is done via a graphical interface
4 # and it is not possible to vote for yourself.
5 #
6 # Note that if two (or more) players receive the same amount of
7 # points, the bitonic sort algorithm will will provide a default
8 # solution. The person nearest the bottom of the
9 # author_config.txt file will be ranked first.
10 #
11 # The implementation is based on VIFF, the Virtual Ideal
12 # Functionality Framework, while the GUI toolkit used is PyGTK.
13
14 from twisted.internet import gtk2reactor # For gtk-2.0
15 gtk2reactor.install()
16 from twisted.internet import reactor
17
18 from optparse import OptionParser
19 from viff.field import GF
20 from viff.runtime import Runtime, create_runtime, gather_shares
21 from viff.comparison import Toft07Runtime
22 from viff.config import load_config
23 from viff.util import rand, find_prime
24
25 import pygtk
26 pygtk.require('2.0')
```

Appendix C. Source Code Author Application

```
27 import gtk
28
29 import re
30 from math import log, floor
31 import time
32
33 class Protocol:
34     """Defining the protocol, which will be started at the
35     bottom of the file."""
36
37     def __init__(self, runtime):
38         """Imports the names of the participants and initiates
39         the creation of the GUI."""
40
41         # Save the Runtime for later use
42         self.runtime = runtime
43
44         # Read player names from config file
45         file = open("author_config.txt", "r")
46         lines = file.readlines()
47         file.close()
48         self.players = []
49         for i in lines:
50             # Transform unicode strings to UTF-8. Norwegian
51             # letters would otherwise have caused warnings.
52             self.players.append(re.sub("\n", "", str(i).decode("
53                 iso8859-15")))
54         self.number_of_players = len(self.players)
55         self.number_of_choices = len(self.players) - 1
56         self.create_gui()
57
58     def create_gui(self):
59         """The graphical interface is built using widgets from
60         the PyGTK toolkit."""
61
62         # Create main window
63         self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
64         self.window.set_title("Rank the Authors")
65         self.window.set_position(gtk.WIN_POS_CENTER)
66         self.window.set_default_size(220, 100)
67         self.window.set_border_width(10)
68
69         # Create the outhter box (widget container)
70         vbox = gtk.VBox(False, 0)
71         hbox = gtk.HBox(True, 0)
72         self.window.add(vbox)
73         vbox.pack_start(hbox, False, False, 0)
74
75         # Table for arranging labels and radio buttons
76         table = gtk.Table(self.number_of_players, self.
77             number_of_players, False)
78         table.set_row_spacings(2)
79         table.set_col_spacings(10)
80         j = 0
```

```

79     for i in range(0, self.number_of_players):
80         if not self.runtime.id == (i+1):
81             # Name of players
82             label = gtk.Label(self.players[i])
83             table.attach(label, j+1, j+2, 0, 1)
84             # 1th, 2th, 3th, etc. author
85             label = gtk.Label(str(j+1) + "th")
86             table.attach(label, 0, 1, j+1, j+2)
87             j += 1
88
89     # Set up grouped radio buttons. The exec statement is
90     # used to create dynamic variable names.
91     var_num = 0
92     active = 1
93     for i in range(0, self.number_of_choices):
94         k = 0
95         for j in range(0, self.number_of_players):
96             name = str(j+1)
97             if not self.runtime.id == (j+1):
98                 var_num += 1
99                 if (var_num % self.number_of_choices == 1):
100                     exec 'self.radio%s = self.setup_radio(
101                         None, "", name)' % var_num
102                     else:
103                         exec 'self.radio%s = self.setup_radio(
104                             self.radio%s, "", name)' % (var_num,
105                             var_num-1)
106                         exec 'table.attach(self.radio%s, k+1, k+2, i
107                             +1, i+2, 0)' % var_num
108                 k += 1
109             # Setting some radio buttons as checked by default
110             exec 'self.radio%s.set_active(True)' % active
111             active += self.number_of_choices + 1
112
113     vbox.pack_start(table, False, False, 0)
114
115     # Separator
116     separator = gtk.HSeparator()
117     vbox.pack_start(separator, False, True, 10)
118
119     # Submit button
120     submitbox = gtk.HBox(False, 0)
121     self.button = gtk.Button("Submit")
122     self.button.connect("clicked", self.check_votes, None)
123     submitbox.pack_start(self.button, True, False, 0)
124     vbox.pack_start(submitbox, False, False, 0)
125
126     # Separator 2
127     separator = gtk.HSeparator()
128     vbox.pack_start(separator, False, True, 10)
129
130     # Progress bar
131     self.pbar = gtk.ProgressBar()
132     vbox.pack_start(self.pbar, False, False, 0)

```

Appendix C. Source Code Author Application

```
129
130     # Separator 3
131     separator = gtk.HSeparator()
132     vbox.pack_start(separator, False, True, 10)
133
134     # Output
135     outputbox = gtk.HBox(False, 0)
136     self.label = gtk.Label("Make your choice, " + str(self.
137         players[self.runtime.id-1]) + "!")
138     outputbox.pack_start(self.label, True, False, 0)
139     vbox.pack_start(outputbox, False, False, 0)
140
141     self.window.show_all()
142
143     def setup_radio(self, group, label, value):
144         """Function to simplify the display of grouped
145         radio buttons."""
146
147         radio = gtk.RadioButton(group, label)
148         radio.set_name(value)
149         return radio
150
151     def on_error(self, widget):
152         """Function that displays an error message if
153         check_votes reports so."""
154
155         md = gtk.MessageDialog(self.window,
156             gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_ERROR,
157             gtk.BUTTONS_OK, "Error:")
158         md.format_secondary_text("You can only give one person
159             one vote. Try again!")
160         md.run()
161         md.destroy()
162
163     def check_votes(self, widget, data=None):
164         """Function for checking that the player have not given
165         more than one vote for each person."""
166
167         check = 1
168         self.list = []
169         button_number = 1
170         for i in range(0, self.number_of_choices):
171             exec 'group = self.radio%s.get_group()' %
172                 button_number in globals(), locals()
173             for j in group:
174                 if j.get_active():
175                     for k in range(0, len(self.list)):
176                         if int(j.get_name()) == self.list[k]:
177                             check = 0
178                             self.on_error(None)
179                             break
180                     self.list.append(int(j.get_name()))
181             button_number += self.number_of_choices
182         if check == 1:
```

```

179         self.submit(self, None)
180
181     def submit(self, widget, data=None):
182         """Function for actions when pressing the
183         submit button."""
184
185         # Create a list of the points given. Player1's received
186         # points is placed at points[0], player2's received
187         # points at points[1], etc. First author gives n-1
188         # points, second author gives n-2 points, etc.
189         self.points = range(self.number_of_players)
190         pts = self.number_of_choices
191         for i in range(0, self.number_of_choices):
192             x = self.list[i]
193             self.points[x-1] = pts
194             pts -= 1
195         # Give yourself 0 points
196         self.points[self.runtime.id-1] = 0
197
198         # Create a temporary output text
199         output = "For lth author you selected: " + str(self.
200             players[self.list[0]-1])
201         self.label.set_text(output)
202         self.button.set_sensitive(False)
203
204         # Setup for the comparison protocol to work
205         l = self.runtime.options.bit_length
206         k = self.runtime.options.security_parameter
207         modulus = 2**65
208         Zp = GF(find_prime(modulus, blum=True))
209
210         # The input is secret shared. We end up with
211         # n*n variables, where n = number of players
212         self.timel = time.clock()
213         var_num = 0
214         for i in range(0, self.number_of_players):
215             player = 1
216             for j in range(0, self.number_of_players):
217                 if self.runtime.id == player:
218                     exec 'a%s = self.runtime.shamir_share([
219                         player], Zp, self.points[i])' % var_num
220                     in globals(), locals()
221                 else:
222                     exec 'a%s = self.runtime.shamir_share([
223                         player], Zp)' % var_num in globals(),
224                     locals()
225                 var_num += 1
226                 player += 1
227
228         self.shares = 0
229         def step(value):
230             """Function for updating the progress bar."""
231
232             self.shares += 1

```

Appendix C. Source Code Author Application

```
228         percent = round((self.shares*100.0) / float(self.
229             number_of_players), 2)
230         text = str(percent) + "%"
231         self.pbar.set_fraction(self.shares / float(self.
232             number_of_players))
233         self.pbar.set_text(text)
234         return value
235
236     # Add callback for one variable per player. This way we
237     # know when a user hits the submit button and we can
238     # update the progress bar.
239     for i in range(0, self.number_of_players):
240         exec 'a%s.addCallback(step)' % i in globals(),
241             locals()
242
243     # Calculate the sum of points for each player
244     array = []
245     var_num = 0
246     for i in range(0, self.number_of_players):
247         exec 'sum%s = 0' % i in globals(), locals()
248         for j in range(0, self.number_of_players):
249             exec 'sum%s += a%s' % (i, var_num) in globals(),
250                 locals()
251             var_num += 1
252         exec 'array.append(sum%s)' % i in globals(), locals()
253             ()
254
255     # Initiate the sorting algorithm of the sums array
256     sorted = self.sort(array)
257
258     # Gather the results and call the self.results_ready
259     # method when they have all been received
260     sorted = gather_shares(map(self.runtime.open, sorted))
261     sorted.addCallback(self.results_ready)
262
263     # Open the sums (shares) and put them in a list
264     list_of_sums = []
265     for i in range(0, self.number_of_players):
266         exec 'open%s = self.runtime.open(sum%s)' % (i, i) in
267             globals(), locals()
268         exec 'list_of_sums.append(open%s)' % i in globals(),
269             locals()
270
271     #sorted.addCallback(lambda _: self.runtime.shutdown())
272
273     def sort(self, array):
274         """This sort function is provided as a part off the
275         VIFF framework. The algorithm used is bitonic sort which
276         was chosen in order to maximize the amount of work
277         (comparisons) done in parallel."""
278
279         # Make a shallow copy - the algorithm wont be
280         # in-place anyway since we create lots of new
281         # Shares as we go along.
```



```

275     array = array[:]
276
277     # Corresponding array for the order of authors according
278     # to their points
279     order = []
280     for i in range(0, self.number_of_players):
281         order.append(i+1)
282
283     def bitonic_sort(low, n, ascending):
284         if n > 1:
285             m = n // 2
286             bitonic_sort(low, m, ascending=not ascending)
287             bitonic_sort(low + m, n - m, ascending)
288             bitonic_merge(low, n, ascending)
289
290     def bitonic_merge(low, n, ascending):
291         if n > 1:
292             # Choose m as the greatest power of 2
293             # less than n.
294             m = 2**int(floor(log(n-1, 2)))
295             for i in range(low, low + n - m):
296                 compare(i, i+m, ascending)
297             bitonic_merge(low, m, ascending)
298             bitonic_merge(low + m, n - m, ascending)
299
300     def compare(i, j, ascending):
301
302         def xor(a, b):
303             # TODO: We use this simple xor until
304             # http://tracker.viff.dk/issue60 is fixed.
305             return a + b - 2*a*b
306
307         le = array[i] <= array[j] # a deferred
308
309         # We must swap array[i] and array[j] when they
310         # sort in the wrong direction, that is, when
311         # ascending is True and array[i] > array[j], or
312         # when ascending is False (meaning descending)
313         # and array[i] <= array[j].
314         #
315         # Using array[i] <= array[j] in both cases we see
316         # that this is the exclusive-or:
317         b = xor(ascending, le)
318
319         # We now wish to calculate
320         #
321         # ai = b * array[j] + (1-b) * array[i]
322         # aj = b * array[i] + (1-b) * array[j]
323         #
324         # which uses four secure multiplications. We can
325         # rewrite this to use only one secure multiplication
326         ai, aj = array[i], array[j]
327         b_ai_aj = b * (ai - aj)
328

```

Appendix C. Source Code Author Application

```
329         array[i] = ai - b_ai_aj
330         array[j] = aj + b_ai_aj
331
332         # Switch the order array correspondingly
333         oi, oj = order[i], order[j]
334         b_oi_oj = b * (oi - oj)
335
336         order[i] = oi - b_oi_oj
337         order[j] = oj + b_oi_oj
338
339         bitonic_sort(0, len(array), ascending=False)
340     return order
341
342     def results_ready(self, results):
343         """This function is called as a callback above and the
344         results variable will contain actual field elements, not
345         just shares."""
346
347         self.time2 = time.clock()
348         self.time = self.time2 - self.time1
349         print "time:", self.time
350
351         # Unpack the list
352         for i in range(0, self.number_of_players):
353             exec 'r%s = int(results[i].value)' % i
354
355         # Create the final output message
356         output = ""
357         for i in range(0, self.number_of_players):
358             exec 'output += "%sth: " + self.players[r%s-1] + " "'
359                 ' % (i+1, i)
360
361         self.label.set_text(output)
362         print output
363
364     # Parse command line arguments.
365     parser = OptionParser()
366     Runtime.add_options(parser)
367     options, args = parser.parse_args()
368
369     if len(args) == 0:
370         parser.error("You must specify a config file")
371     else:
372         id, players = load_config(args[0])
373
374     # Create a deferred Runtime and ask it to run our protocol
375     # when ready
376     pre_runtime = create_runtime(id, players, 1, options,
377                                 Toft07Runtime)
378     pre_runtime.addCallback(Protocol)
379
380     reactor.run()
```

Appendix D

Source Code Web Application

The source code of the Java applet and one of the computation servers are shown below. In addition the web application consists of several PHP files which are included in an attached ZIP file, see Appendix E.

D.1 Vote.java

Listing D.1: Source Code Vote.java

```
1 import java.awt.*;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import java.applet.*;
5 import java.io.*;
6 import java.math.BigInteger;
7 import java.net.*;
8 import java.security.NoSuchAlgorithmException;
9 import java.security.SecureRandom;
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.Iterator;
13 import javax.swing.*;
14
15 public class Vote extends Applet implements ActionListener {
16
17     String title;
18     String code;
19     String user_id;
20     String description;
21     int number_of_options;
22     HashMap<Integer, String> options = new HashMap<Integer,
23         String>();
24     ArrayList<Checkbox> radiobuttons;
```

Appendix D. Source Code Web Application

```
25 | JButton submitButton;
26 | JLabel titleLabel;
27 | JTextArea descriptionField;
28 | JScrollPane scrollArea;
29 | JTextArea outputField;
30 | CheckboxGroup radioGroup;
31 |
32 | String result = "";
33 |
34 | public void init() {
35 |     // Fetch the parameter values from the HTML code that
36 |     // is calling the applet
37 |     title = getParameter("title");
38 |     description = getParameter("description");
39 |     code = getParameter("code_var");
40 |     user_id = getParameter("userid");
41 |     number_of_options = Integer.parseInt(getParameter("
42 |         number_of_options"));
43 |
44 |     for (int i = 0; i < number_of_options; i++) {
45 |         options.put(i, getParameter("option" + i));
46 |     }
47 |
48 |     // Make the GUI
49 |     setLayout(null);
50 |     submitButton = new JButton("Submit");
51 |     titleLabel = new JLabel(title);
52 |     descriptionField = new JTextArea(description);
53 |     scrollArea = new JScrollPane(descriptionField);
54 |     outputField = new JTextArea();
55 |     outputField.setLineWrap(true);
56 |     radioGroup = new CheckboxGroup();
57 |     radiobuttons = new ArrayList<Checkbox>();
58 |
59 |     for (int i = 0; i < number_of_options; i++) {
60 |         radiobuttons.add(new Checkbox(options.get(i),
61 |             radioGroup, false));
62 |     }
63 |
64 |     // Set x and y, width and height
65 |     titleLabel.setBounds(20, 0, 350, 30);
66 |     descriptionField.setEditable(false);
67 |     descriptionField.setLineWrap(true);
68 |     descriptionField.setBorder(BorderFactory.
69 |         createLoweredBevelBorder());
70 |
71 |     scrollArea.setBounds(20, 30, 350, 40);
72 |     scrollArea.setAutoscrolls(true);
73 |
74 |     int y = 80;
75 |     for (int i = 0; i < number_of_options; i++) {
76 |         radiobuttons.get(i).setBounds(20, y, 350, 30);
77 |         y += 30;
78 |     }
```

```

76
77     submitButton.setBounds(20, y+10, 100, 30);
78     outputField.setBounds(20, y+50, 350, 30);
79     outputField.setEditable(false);
80
81     // Add the GUI components to the applet
82     add(titleLabel);
83     add(scrollArea);
84     for (int i = 0; i < number_of_options; i++) {
85         add(radiobuttons.get(i));
86     }
87     add(submitButton);
88     add(outputField);
89
90     // Add action to the button
91     submitButton.addActionListener(this);
92 }
93
94 // Actions when pressing the submit button
95 public void actionPerformed(ActionEvent event) {
96     if (event.getSource() == submitButton) {
97         String label = radioButton.getSelectedCheckbox().
98             getLabel();
99         int id = -1;
100        Iterator<Integer> it = options.keySet().iterator();
101        while(it.hasNext()) {
102            Integer key = it.next();
103            String val = options.get(key);
104            if (val == label) {
105                id = key;
106                break;
107            }
108        }
109        submitButton.setEnabled(false);
110        outputField.setText("You selected: " + label + "(" +
111            id + ")");
112        makeShares(id);
113    }
114 }
115
116 // Creation of the shares
117 public void makeShares(int id) {
118     BigInteger share1;
119     BigInteger share2;
120     BigInteger share3;
121
122     BigInteger random = BigInteger.ZERO;
123     BigInteger p = new BigInteger("
124         1461501637330902918203684832716283019655932542983");
125     // same modulus as in VIFF
126
127     SecureRandom secRandom;
128     try {
129         secRandom = SecureRandom.getInstance("SHA1PRNG");

```

Appendix D. Source Code Web Application

```
126         random = new BigInteger(160, secRandom); // 160 bits
127     }
128     catch (NoSuchAlgorithmException e) {
129         e.printStackTrace();
130     }
131
132     // share-x = id + random*x mod p
133     share1 = intToBig(id).add(random.multiply(intToBig(1))).
134         mod(p);
135     share2 = intToBig(id).add(random.multiply(intToBig(2))).
136         mod(p);
137     share3 = intToBig(id).add(random.multiply(intToBig(3))).
138         mod(p);
139
140     outputField.setText("Share 1: " + share1 + "Share 2: " +
141         share2 + "Share 3: " + share3);
142     encryptShares(share1, share2, share3);
143 }
144
145 // Function for converting an Integer to a BigInteger
146 public BigInteger intToBig(int number) {
147     return new BigInteger(Integer.toString(number));
148 }
149
150 // 1024 bits RSA encryption of the shares
151 public void encryptShares(BigInteger share1, BigInteger
152     share2, BigInteger share3) {
153     BigInteger n1 = new BigInteger("
154         968152160772263289847388078038937747255610409...93");
155     BigInteger n2 = new BigInteger("
156         918686459282295450909710816030529675197106090...29");
157     BigInteger n3 = new BigInteger("
158         115917249021135811596506973975170039300995937...09");
159     BigInteger e = new BigInteger("65537"); // 2^16+1
160
161     BigInteger c1 = share1.modPow(e, n1);
162     BigInteger c2 = share2.modPow(e, n2);
163     BigInteger c3 = share3.modPow(e, n3);
164
165     outputField.setText("Wait! Saving to db... Encr 1: " +
166         c1 + "Encr 2: " + c2 + "Encr 3: " + c3);
167     doPost(code, user_id, c1, c2, c3);
168 }
169
170 // Function which transmits the shares via a POST request to
171 // the PHP/Web server
172 public void doPost(String code, String user_id, BigInteger
173     share1, BigInteger share2, BigInteger share3) {
174     URL url;
175     URLConnection urlConn;
176     DataOutputStream output;
177     BufferedReader input;
178
179     try {
```

```
169         url = new URL(getCodeBase().toString() + "vote_post.  
170             php");  
171  
172         urlConn = url.openConnection();  
173         urlConn.setDoOutput(true);  
174         urlConn.setDoInput(true);  
175         urlConn.setUseCaches(false);  
176         urlConn.setRequestProperty("Content-Type", "  
177             application/x-www-form-urlencoded");  
178  
179         output = new DataOutputStream(urlConn.  
180             getOutputStream());  
181         String content = "code=" + code + "&userid=" +  
182             user_id + "&c1=" + share1 + "&c2=" + share2 + "&  
183             c3=" + share3;  
184         output.writeBytes(content);  
185         output.flush();  
186         output.close();  
187         DataInputStream in = new DataInputStream(urlConn.  
188             getInputStream());  
189         input = new BufferedReader(new InputStreamReader(in)  
190             );  
191         String str;  
192         while ((str = input.readLine()) != null) {  
193             result = result + str + "\n";  
194             outputField.setText(result);  
195         }  
196         input.close();  
197     }  
198 }
```

Appendix D. Source Code Web Application

D.2 server1.py

Listing D.2: Source Code server1.py

```
1 from optparse import OptionParser
2 from twisted.internet import reactor
3
4 from viff.field import GF
5 from viff.runtime import Runtime, create_runtime, gather_shares,
  Share
6 from viff.comparison import Toft07Runtime
7 from viff.config import load_config
8 from viff.util import rand, find_prime
9
10 from twisted.web import xmlrpc, server
11 import MySQLdb
12 from decimal import *
13 import smtplib
14 import string
15
16 class MyFunctions(xmlrpc.XMLRPC):
17
18     def xmlrpc_initiate_computation(self, code):
19         """This functions is called remotely from the
20         web server (PHP)."""
21
22         self.protocol.fetch_from_db(code)
23         return True
24
25     def save_instance(self, protocol):
26         """Saves the protocol instance such that is is
27         reachable for the MyFunctions class"""
28
29         self.protocol = protocol
30
31 class Protocol:
32     """Defining the protocol, which will be started at
33     the bottom of the file."""
34
35     def __init__(self, runtime):
36         # Save the Runtime for later use
37         self.runtime = runtime
38         self.start_xmlrpc_server()
39
40     def start_xmlrpc_server(self):
41         """Starts Twisted's XML-RPC server."""
42
43         host = "localhost"
44         port = 8001
45
46         r = MyFunctions(allowNone=True)
47         r.save_instance(self)
48         reactor.listenTCP(port, server.Site(r))
49
```



```

50 def fetch_from_db(self, code):
51     """Fetches the shares from DB and decrypts them."""
52
53     self.code = code
54     print "Poll ID:", self.code
55     print "Fetching encrypted shares from DB..."
56
57     # connect
58     db = MySQLdb.connect(host="mysql.stud.ntnu.no", user="
59         secret", passwd="secret",
60         db="havardv_smpc")
61     # create a cursor
62     cursor = db.cursor()
63     # execute SQL statement
64     if self.runtime.id == 1:
65         cursor.execute("SELECT share1 FROM result WHERE code
66             = '"+self.code+"'")
67     elif self.runtime.id == 2:
68         cursor.execute("SELECT share2 FROM result WHERE code
69             = '"+self.code+"'")
70     elif self.runtime.id == 3:
71         cursor.execute("SELECT share3 FROM result WHERE code
72             = '"+self.code+"'")
73     # get the resultset as a tuple
74     result = cursor.fetchall()
75     db.close()
76     # iterate through resultset
77     self.shares = []
78     i = 0
79     print "Decrypting shares..."
80     for record in result:
81         # from encr to share
82         # d is a unique private RSA key for server 1
83         n = 968152160772263289847388078038937747255610...93
84         d = 198588118120474470992240077102666282197188...45
85
86         exec 'c%s = int(record[0])' % i in globals(), locals
87         ()
88         getcontext().Emax = 10000000000000000000
89         exec 'm%s = pow(c%s, d, n)' % (i,i) in globals(),
90         locals()
91         exec 'self.shares.append(m%s)' % i in globals(),
92         locals()
93         i = i + 1
94     self.do_computation()
95
96 def do_computation(self):
97     """The encrypted values are converted to shares
98     and reconstructed."""
99
100    print "Starting computation of shares..."
101    self.modulus=2**160
102    self.prime = find_prime(self.modulus, blum=True)
103    #print "prime: ", self.prime

```

Appendix D. Source Code Web Application

```
97     Zp = GF(self.prime)
98
99     # Converting to shares
100     self.number_of_shares = len(self.shares)
101     for i in range(0, self.number_of_shares):
102         exec 'n%s = Share(self.runtime, Zp, Zp(self.shares[i
103             ]))' % i in globals(), locals()
104
105     # The results are secret shared, so we must open them
106     # before we can do anything useful with them.
107     list_of_values = []
108     for i in range(0, self.number_of_shares):
109         exec 'open%s = self.runtime.open(n%s)' % (i, i) in
110             globals(), locals()
111         exec 'list_of_values.append(open%s)' % i in globals
112             (), locals()
113
114     # We will now gather the results and call the
115     # self.results_ready method when they have all been
116     # received.
117     results = gather_shares(list_of_values)
118     results.addCallback(self.results_ready)
119
120     results.addCallback(lambda _: self.runtime.synchronize()
121         )
122     #results.addCallback(lambda _: self.runtime.shutdown())
123
124 def results_ready(self, results):
125     """This function is called as a callback above and the
126     results variable will contain actual field elements, not
127     just shares."""
128
129     self.fetch_poll_info(self.code)
130     self.number_of_options = len(self.options)
131
132     for i in range(0, self.number_of_options):
133         exec 'sum%s = 0' % i
134
135     for i in range(0, self.number_of_shares):
136         temp = results[i].value
137         exec 'sum%s += 1' % temp
138
139     print "Public output:"
140     output = ""
141     for i in range(0, self.number_of_options):
142         newline = "\n"
143         exec 'line = self.options[i] + " received " + str(
144             sum%s) + " vote(s)."' % i
145         output = output + newline + line
146
147     print output
148
149     self.send_mail(self.participants_email, output)
```

```

146 def send_mail(self, list, output):
147     """Sending of e-mails to the participants of
148     the vote."""
149
150     emails = []
151     emails = list.split(";")
152     # Remove the extra semicolon
153     emails.pop()
154
155     fromaddr = self.author_email
156     toaddrs = ""
157     cc = "" #poll_author
158     bcc = emails
159     subject = "Result of poll: " + self.title
160     text = "The output of the vote is:\n" + output
161
162     # Prepare actual message
163     message = string.join((
164         "From: %s" % fromaddr,
165         "Subject: %s" % subject,
166         "",
167         text
168     ), "\r\n")
169
170     # Send the mail
171     server = smtplib.SMTP('smtp.stud.ntnu.no')
172     server.set_debuglevel(1)
173     server.sendmail(fromaddr, bcc, message)
174     #print server.verify(toaddrs)
175     print "E-mails are sent..."
176
177     server.quit()
178
179 def fetch_poll_info(self, code):
180     """Fetching of e-mail addresses, poll title, etc.
181     from the database."""
182
183     print "Fetching the recipient's e-mail addresses from DB
184     ..."
185     # connect
186     db = MySQLdb.connect(host="mysql.stud.ntnu.no", user="
187         secret", passwd="secret",
188         db="havardv_smpc")
189     # create a cursor
190     cursor = db.cursor()
191     # execute SQL statement
192     if self.runtime.id == 1:
193         cursor.execute("SELECT title, author_email,
194             participants_email, options FROM poll WHERE code
195             = '%'+self.code+''"")
196
197     # get the result
198     result = cursor.fetchone()
199     db.close()
200     self.title = result[0]

```

Appendix D. Source Code Web Application

```
196         self.author_email = result [1]
197         self.participants_email = result [2]
198         self.options = []
199         self.options = result [3].split(";")
200         # Remove the extra semicolon
201         self.options.pop()
202
203 # Parse command line arguments.
204 parser = OptionParser()
205 Runtime.add_options(parser)
206 options, args = parser.parse_args()
207
208 if len(args) == 0:
209     parser.error("You must specify a config file")
210 else:
211     id, players = load_config(args [0])
212
213 # Create a deferred Runtime and ask it to run our protocol when
ready.
214 pre_runtime = create_runtime(id, players, 1, options,
215                             Toft07Runtime)
216 pre_runtime.addCallback(Protocol)
217
218 # Start the Twisted event loop.
219 reactor.run()
```

Appendix E

Attachment/ZIP file

Included with this master's thesis is an attachment in the form of a ZIP file. Three directories exist in this file, and the content is listed next.

E.1 Rank the Authors

Refer to Section 7.1.2 for a description of the files.

- author.py
- author_config.txt

E.2 Secure Web Voting

Refer to Section 8.2 for a description of the files.

- Computation Servers (directory)
 - server1.py
 - server2.py
 - server3.py
- Web Site (directory)
 - config.php
 - index.php
 - process.php
 - style.css
 - Vote.class
 - vote.php
 - vote_functions.php

Appendix E. Attachment/ZIP file

- vote_post.php
- xmlrpc-1.php
- xmlrpc-2.php
- xmlrpc-3.php
- xmlrpc.inc
- xmlrpc_wrappers.inc
- xmlrpcs.inc
- images (directory)
- Java applet and MySQL (directory)
 - Vote.java (The source code of the Java applet.)
 - phpMyAdmin SQL Dump.sql (SQL dump of the three database tables.)

E.3 Secure Position Determination

This program was not included in the thesis, but was a test program implementing the protocol by Tord Ingolf Reistad from his paper *Multi-party Secure Position Determination* [Rei06].

- position.py