Tord Ingolf Reistad

A General Framework for Multiparty Computations

Thesis for the degree of Philosophiae Doctor

Trondheim, May 2012

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

ISBN 978-82-471-3572-3 (printed ver.)
ISBN 978-82-471-3573-0 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2012:143

Printed by NTNU-trykk

Abstract

Multiparty computation is a computation between multiple players which want to compute a common function based on private input. It was first proposed over 20 years ago and has since matured into a well established science. The goal of this thesis has been to develop efficient protocols for different operations used in multiparty computation and to propose uses for multiparty computation in real world systems. This thesis therefore gives the reader an overview of multiparty computation from the simplest primitives to the current state of software frameworks for multiparty computation, and provides ideas for future applications.

Included in this thesis is a proposed model of multiparty computation based on a model of communication complexity. This model provides a good foundation for the included papers and for measuring the efficiency of multiparty computation protocols. In addition to this model, a more practical approach is also included, which examines different secret sharing schemes and how they are used as building blocks for basic multiparty computation operations. This thesis identifies five basic multiparty computation operations: sharing, recombining, addition, multiplication and negation, and shows how these five operations can be used to create more complex operations. In particular two operations "less-than" and "bitwise decomposition" are examined in detail in the included papers. "less-than" performs the "$<$" operator on two secret shared values with a secret shared result and "bitwise decomposition" takes a secret shared value and transforms it into a vector of secret shared bitwise values.

The overall goal of this thesis has been to create efficient methods for multiparty computation so that it might be used for practical applications in the future.

Preface

This thesis is submitted in partial fulfillment of the requirements of the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The PhD study was conducted from 2004 to 2012. During this study period, I have been hosted by the the Department of Telematics, NTNU, and I spent one year at the University of Aarhus, Department of Computer Science. Academically, the PhD study was conducted through the Department of Telematics, NTNU, and has been supervised by Professor Stig Frode Mjølsnes.

The document and included papers have been formatted in LaTeX using a modified version of the document class *kapproc.cls* provided by Kluwer Academic Publishers.

Acknowledgements

First of all I would like to dedicate this thesis to Karl Steffen Reistad, my brother who did not live to see this thesis finished. His life was cut short by a falling rock.

Writing a thesis is not an easy task. First you have to come up with novel ideas. Then those ideas have to be written down and published. While getting the ideas might be hard for some, it has not been my problem since I got interested and gained an understanding of multiparty computation. My problem has been to write those ideas down and get them into a form such that they can be published. Writing has never been my strong suit and Writer's block, procrastination and Internet surfing have stood by me trying to delay me every time I tried to write something. Thus it has taken me over 8 years from starting my Ph.D. work to submitting a finished thesis.

The Ph.D. was started on 1st of January 2004 and the papers included in the second chapter where written in the years 2005 to 2009. The introduction chapter in this thesis was started some time in 2008, but writing it was going slowly as I was trying to develop a working multiparty computation program at the same time. This project has gone through multiple iterations, but is still not finished. I stopped working on the thesis from the start of 2010 until the summer of 2011 because I got a job. This therefore proves that for me it is impossible to finish a Ph.D. and work full time at the same time. The only reason I could finish is that my employer Difi (Agency for Public Management and eGovernment) gave me the opportunity to take some time off in 2011 to finish writing the introduction to this this thesis. For this I would like to thank Difi and Tor Alvik. The introduction chapter was written in collaboration with my advisor Stig F. Mjølsnes, who helped me structure the introduction and gave a very thorough examination of the finished product. I would also like to thank Karin Holt and Harald for proofreading the introduction. The papers have not been proofread, therefore they are presented as they where published.

I am grateful for the support of many people and for their tireless encouragement in trying to get me to finish this Ph.D. thesis. Without their support this thesis would not have been written, in particular this includes my parents Alis and Ole Reistad, my advisor Stig Frode Mjølsnes and my fiancé Anlaug Landfald. I would also like to thank the great group of people I worked with in Aarhus, Denmark who gave me the inspiration for much of this thesis, in

particular Tomas Toft who I worked closely with. I would also like to thank my co-workers at the department of Telematics.

I would like to thank my friends in the gaming group a.k.a. rollespillgjengen Tord(Tård), Stein, Harald, Finn Olav, Pernille, Morten, Rune, Jardar and Anders. You have been the greatest friends I know, and we have been through some great adventures together. I would also like to thank Kjetil for his support.

To all those not mentioned by name, and all other friends and family, you are not forgotten. It is just that I am not good with names and it is difficult to start naming all the people you want to thank and not leave anyone out.

Contents

Publications Included in the Thesis

- PAPER A:
  Tord Ingolf Reistad *Multi-party Secure Position Determination* At Norsk informatikkonferanse 2006 (NIK 06) Molde, Norway, November 20-22, 2006.

- PAPER B:
  Tord Ingolf Reistad and Tomas Toft *Secret Sharing Comparison by Transformation and Rotation* In ICITS (International Conference on Information Theoretic Security) 2007, LNCS (Lecture Notes in Computer Science) 4883. Madrid, Spain, May 25-28, 2007.

- PAPER C:
  Tord Ingolf Reistad *Multiparty comparison, An improved multiparty protocol for comparison of secret-shared values* In Proceedings of SECRYPT 2009, International conference on security and cryptography. Milano, Italy. July 6-10, 2009.

- PAPER D:
  Atle Mauland, Tord Ingolf Reistad, Stig Frode Mjølsnes *Realizing Distributed RSA Key Generation using VIFF* At NISK (Norsk Informasjonssikkerhetskonferanse) 2009. Trondheim, Norway, November 24-25, 2009.

- PAPER E:
  Md. Abdul Based, Tord Ingolf Reistad, Stig Frode Mjølsnes *Internet Voting using Multiparty Computations* At NISK (Norsk Informasjonssikkerhetskonferanse) 2009. Trondheim, Norway, November 24-25, 2009.

- PAPER F:
  Tord Ingolf Reistad and Tomas Toft *Linear, constant rounds Bit-decomposition* At ICISC (International Conference on Information Theoretic Security) 2009. Shizuoka, Japan, December 3-6, 2009.

I

THESIS INTRODUCTION

1. Introduction

1.1 Motivation

This thesis covers the topic of secure multiparty computation which is hereafter referred to as multiparty computation. One example of multiparty computation is to imagine a group of millionaires who want to find out which person among them is the richest, but being millionaires they do not want to reveal exactly how rich they are. This problem is called the "millionaire problem" and was suggested by Yao in 1982 [Yao82]. While this problem might seem unsolvable as it is impossible to compute a function based on input which is kept secret, this problem can be solved using multiparty computation. Which ensures that the millionaires do not reveal how rich they are to each other while at the same time allowing them to determine which person is the richest.

Another example; imagine a group of investors that have ten projects (investment opportunities) that they can fund. The investors also know that they only have resources to fund and oversee three projects. Each investor has his own ideas about how good the project is and how much he wants to invest in that project, but he does not want to reveal this information to his fellow investors for various reasons. First the group of investors determine the objective parameters and the formulas for which the projects are worth funding. Then each investor can rate each project according to those parameters which have already been set up. The group can hire a lawyer or other trusted third-party to do the calculations. The lawyer will then gather confidential information from all investors about how they have rated each project. Apply a formula on the information to compute a set of results, then the lawyer will convey the results to the group of investors. Afterwards the lawyer destroys all the confidential information so the information cannot be misused. Alternatively, the investors can use multiparty computations achieve the same result. Thus avoiding the need to hire a lawyer and trusting that lawyer with the private information. Multiparty computation can in this context be thought of as a virtual trusted third-party.

Any calculations computed on private input cannot be totally secure, for example a lawyer can be bribed or coerced into revealing private information or can modify the calculations. If the lawyer modified the calculations in the previous example this could change which projects are funded or how the projects are funded. Also since the calculations were done based on private information it would be very difficult to detect such modifications. The same is the case with multiparty computation where some players (investors or millionaires) in the group could work together to reveal or modify information. How difficult it is to reveal or modify information depends upon the particulars surrounding each multiparty computation, for example, which secret sharing scheme is used, if input is verified, if the channels between players is encrypted, etc. It can therefore take as little as one player to modify the information, while

with other protocols, it might require all players to perform the calculations flawlessly in order for any of them to get any results. Typically one third or half of the players would have to collude for them to gain any information about the other players private information.

In general, the method of multiparty computations can be expressed as the method whereby a group of players can compute some common agreed upon function with private information as input. Information is here defined as a set consisting of data (or values) and something that gives the data meaning. For example, the number "8" is only data, but when the number "8" refers to the number of years it took me to write this thesis then that number has a meaning and is relevant information. Private information is pieces of information that is usually only known by one player (or to a set of players) but not to all players. Multiparty computation is accomplished by the players sharing data that represent the different pieces of information without sharing the actual data.

For example the number "8" could be shared to two players by giving one player the number "1" and the other player the number "7". Then neither player has the actual data (8), but each player has a representation of the data (1 and 7), and together the two representations can be added together to find the actual data ($1 + 7 = 8$).

1.2 Technology and Applications

Currently, the notion of multiparty computations is moving from a purely theoretical construct to technology that can be applied to practical problems. The first large scale practical application of multiparty computation was a double blind sugar bead auction in Denmark [BCD+08], but the possibilities for where multiparty computation could be used is almost limitless.

Here are some examples where multiparty computation could be used:

- Financial markets are prone to catastrophic collapse. Presently this is due to a high degree of interconnection in the financial markets, lack of transparency and agreements that are often structured to fail under the same conditions. These catastrophic collapses could possibly be avoided and it would be beneficial for the financial system as a whole if the overall stability of the system could be calculated and common vulnerabilities could be identified.

- Due to privacy laws, financial institutions in Norway cannot share data about individuals. Therefore each financial institution must evaluate individuals based on historical information about income and net worth for each person. While this information gives some indication about how creditworthy the individual is, the individual can present the same set of facts to many financial institutions and get lines of credit that could put that individual in an unsustainable financial situation. Multiparty computations could then be used to identify individuals which are in

financial distress or at risk of a default, thereby solving the financial situation before it becomes unsustainable.

- The original setting for multiparty computations was the "millionaire problem" where two or more millionaires wanted to see who was the richest without revealing how rich they are [Yao82]. In the same genre of problems is a set of companies that want to optimize some production parameters, based on internal prices that they do not want to reveal. [Tof09]
- Electronic voting could also benefit from using multiparty computation for tallying the number of votes that are cast. By distributing the tallying process the voters will not have to trust a single centralized computer system, but can instead trust a group of computers controlled by different political parties which together compute the tally. This is somewhat analogous to paper voting where the tallying process is distributed, as each district or polling station tallies the votes locally before sending the sum of votes to a central system. Therefore polling fraud both at the local level and at the central system would be easier to detect. The current state of electronic voting can be compared to a system for paper voting where all polling stations would pack all the votes into a black box and ship that box off to a central tallying system. Such a system would in addition to being slow also be viewed as less secure.
- Ad-hoc networks and sensor networks are well suited for multiparty computations. These networks operate without any central controls. Each node has to cooperate with other nodes to route packets through the network and ensure that the network works efficiently. There are several potential routing algorithms in ad-hoc networks, each with its own strengths and weaknesses. But most of these routing algorithms work on the premise that all nodes share information about themselves and the rest of the network freely and correctly. If such free flow of information is not possible, then multiparty computation could be used facilitate computations based on private information without releasing that information. Multiparty computation could also add features to an ad-hoc network that are not present in current protocols, such as sending and receiving packages without knowing which node it originates from and where it is going to be transmitted.

1.3 The Problem

1.3.1 Practice

Multiparty computations have many properties which could be beneficial to real world applications, but thus far there has been little adoption of the technology in practice. The challenges faced by this technology are technical, organizational and conceptual in nature.

From a technical standpoint, there are two main roadblocks. The first roadblock is one of efficiency. For example the VIFF framework [Tea09] can compute about 1800 multiplications per second or about 8 comparisons per second. (Measurements done by Martin Geisler in 2009.) The Sharemind framework [oTCa] is more efficient with 100,000-800,000 operations per second [Cyb], this is comparable to typical computer performance of the 1980s. The second technical roadblock is to get useful standard software platforms so that the basic multiparty protocols do not have to be rewritten for each new implementation. This is discussed further in Section 6.

The conceptual challenge is that it is difficult for people that have not worked with multiparty computation to understand the concept and trust that this technology works. Multiparty computation is somewhat counterintuitive, as it is a system for computing with seemingly random information that gets useful results. The players which are willing to share private information using multiparty computation are asked to trust the group as a whole, or a sufficiently large subgroup, but at the same time, distrust any other player. It is also very difficult to hold someone accountable if something goes wrong unless it is built into the protocols from the very beginning.

The third challenge is organizational and is linked with the conceptual challenge. There has to be a minimum number of players for multiparty computations to work. Convincing one organization to use multiparty computation is therefore not enough, there has to be at least two or three organizations that are willing to use this technology before the multiparty setting can be reached.

1.3.2 Technical

My motivation for working on this topic is that the basic operations in multiparty computation and the basic models are now well understood, but the question of finding more efficient protocols for more complicated operations and building blocks is still an open research question. We are beginning to find good, efficient protocols for more complicated operations, but there is still much room for improvement. Therefore it is an interesting field to work on. The challenges of finding improvements or new applications is part mathematical, part computer science, and part security awareness.

My research presented in this thesis has focussed on the question of efficient implementation of multiparty computation. In particular, the research results are centered around the "greater than" and "bit-decomposition" operation. The reason for choosing these operations in particular was that efficient algorithms for some of the basic operations already existed when the papers where written, making further improvements on these operations difficult to achieve. Other more complex operations, such as "division" were too difficult to make progress on. Making the two operations "'less-than" and "bit-decomposition" was ripe for further development. Progress on these operations can hopefully in turn be used to achieve results with more complex operations.

1.3.3 Context and Scope

The work has been mostly theoretical as the largest improvements will come from faster algorithms. Although many of the ideas have been formulated by using Shamir's secret sharing scheme, it has also been a goal to make the protocols independent of any particular secret sharing scheme so that they can be useful for most secret sharing schemes. My research has not delved into specific secret sharing schemes, but rather has focused on making improvements that can be applied to many secret sharing schemes.

1.4 The Structure of the Thesis

This section provides an introduction to multiparty computation and the motivation for working with multiparty computation. Section 2 examines a model for multiparty computation based on communication complexity. Section 3 examines three of the most popular secret sharing schemes. Section 4 shows how multiparty computations can be achieved with secret sharing schemes. The section also gives an overview of a fully homomorphic encryption scheme. Section 6 examines currently available software frameworks for computing multiparty computations. Finally, Section 7 provides an overview of the papers included in part two of this thesis.

2. The Communication Model

2.1 Introduction

This section presents a communication model for multiparty computations. The model will be based on the definitions of communication complexity by Yao [Yao79]. The multiparty communication model will be generalized from Yao's two party model to a model for three parties or more, and include secret sharing computation.

The security properties of the protocols based on this model are derived from the security properties of the underlying secret sharing schemes and multiparty operations, without limiting the model to any specific secret sharing scheme. The weakness of such a model is that although it gives a good understanding of the communication involved, it provides very little information about the security properties of secret sharing and multiparty computation. Other models have been proposed, such as Toft and Thorbek [Tof07, Tho09], which are based on the universal composability framework [Can01].

The model presented here can be used as a basis to understand the round and multiplication costs which are used in paper **B**, **C** and **F**. The emphasis has been put on proving the efficiency of the proposed protocols, rather than to give rigorous proofs of the security of the protocols. This has been done to view multiparty computation from a more practical approach as opposed to a pure mathematical approach.

2.2 Notation

The following notation will be used:

- There are n players, they are labeled $P_1, P_2, \ldots, P_n$.
- The function f is a function from the domain of inputs X to the range of outputs Y, $f : X \rightarrow Y$ also expressed as $y = f(x)$
- Let x be an element in the domain X, and y be an element in the range Y. The domain X can be considered a Cartesian product of multiple sets, hence an element x becomes an ordered tuplet of values. The function f can therefore also be written as $y = f(x_1, x_2, \ldots, x_n)$. Likewise, the range of the function can be considered a Cartesian product from multiple sets, hence in general we write $(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n)$, where n corresponds to the number of players.
- Player P_i, where $i \in \{1, \ldots, n\}$, has inputs from the domain X_i, such that $X_1 \times X_2 \times \ldots \times X_n = X$.
- All players $P_1, P_2, \ldots P_n$ may receive the same output Y. However, the model also allows each player to receive different outputs, so that the player P_i, where $i \in \{1, \ldots, n\}$, will receive output from the range Y_i, where the constraint is that $Y_1 \times Y_2 \times \ldots \times Y_n = Y$.
- The domains and ranges can be represented using bits, thus $X_i \in \{0,1\}^{m_x i}$ and $Y_i \in \{0,1\}^{m_y i}$, where $m_x i, \in \mathbb{N}$, $m_y i, \in \mathbb{N}$ and $i \in \{1, \ldots, n\}$.
- $\mathcal{P}$ denotes the protocol between the players.
- The communication complexity for the function f implemented by the most efficient protocol $\mathcal{P}$ is denoted by $D(f)$.

2.3 Communication Complexity

Ref. [KN06] describes the general communication problem as:

> A system must perform some task that depends on information distributed among the different parts of the system (called *processors, parties, or players*). The players thus need to communicate with each other in order to perform the task.

The notion of communication complexity was first introduced by Yao in 1979 [Yao79] as a model with two players with the following assumptions:

- Each player in the system gets a fixed part of the input information.
- The only resource of interest is communication.
- The task performed is the computation of some pre-specified function of the input.

Since the model is only interested in the amount of communication between the players, it is assumed that unless otherwise stated the players have unlimited computational power. The communication between the players is performed according to some fixed pre-defined protocol $\mathcal{P}$ which computes the function f by sending messages between the players until the value $y \in Y$ can be determined uniquely.

The cost of a protocol $\mathcal{P}$ is the worst case number of bits communicated between the players. There are many ways to formalize this notion of cost. One approach is to formalize it from the players' point of view. This thesis will instead formalize it from the protocol designers' point of view. This leads us to the following definition from [KN06]:

Definition 1. *A protocol $\mathcal{P}$ over domain $X_1 \times X_2$ with range Y is a binary tree where each internal node v is labeled either by a function $a_v : X_1 \to \{0, 1\}$ or by a function $b_v : X_2 \to \{0, 1\}$, and each leaf is labeled with an element $y \in Y$.*

The value of the protocol $\mathcal{P}$ on input (x_1, x_2) is the label of the leaf reached by starting from the root, and walking on the tree. At each internal node v labeled by a_v walking left if $a_v(x_1) = 0$ and right if $a_v(x_1) = 1$, and at each internal node labeled by b_v walking left if $b_v(x_2) = 0$ and right if $b_v(x_2) = 1$. The cost of the protocol $\mathcal{P}$ on input (x_1, x_2) is the length of the path taken on input (x_1, x_2). The cost of the protocol $\mathcal{P}$ is the height of the tree.

The communication complexity of f is then defined as:

Definition 2. *For a function $f : X_1 \times X_2 \to Y$ the deterministic communication complexity of f is the minimum cost of $\mathcal{P}$, over all protocols $\mathcal{P}$ that compute f. This is denoted by $D(f)$.*

From these definitions we can say that for two players P_1 and P_2, P_1 "knows" the value $x_1 \in X_1$ and P_2 "knows" $x_2 \in X_2$. To be more specific, P_1 "knows" the value $x_1 \in X_1$ means that P_1 has absolute knowledge of the bits used to represent x_1 and when information from the domain X_1 is communicated to the other player then only P_1 is allowed to supply this information. Player P_2 does not "know" the value x_1 that means that the player P_2 does not have absolute knowledge of the bits representing x_1 and cannot use any probability functions to estimate the value x_1, but most wait for player P_1 to communicate information about the value x_1 as the function is deterministic and both players must arrive at the same result.

For two players, the set of bits representing the domains X_1 and X_2 must be disjoint sets, because if there was some overlap between the domains X_1 and X_2 then that information would be common knowledge for both players and need not be communicated, therefore such overlapping information can be removed from the input domains.

Using this model we can see that a simple protocol is for player P_1 to send x_1 to P_2 and for player P_2 to send x_2 to P_1. Both players can then compute

$y = f(x_1, x_2)$. This leads to the following equation:

$$D(f) \leq log_2|X_1| + log_2|X_2| \tag{1}$$

The definition for communication complexity is limited to the number of bits being sent. In practice, there are multiple reasons to limit the number of interactions between the players. For example computer protocols incur a substantial overhead cost for each message sent. The number of interactions can be modeled by defining rounds of communication.

Definition 3. *A k round protocol is a protocol where the players send k messages to each other before determining the output. This is denoted $D^k(f)$, which is the best k round protocol for f.*

With this definition, the simple protocol where P_1 sends x_1 to P_2 and P_2 sends x_2 to P_1 is a two round protocol. This is also a lower bound for the number of rounds. (Note that this definition differs from the definition in [KN06].)

2.4 Multiple Players

The two player model examined so far can be expanded to multiple players. When the model is expanded to n players $P_1, P_2, \ldots, P_n$, where $n > 2$, both how the input domain X is distributed and the communication model between players becomes more complex.

For two players the input domain X is split into two domains X_1 and X_2, therefore for n players the domain X is split in into n domains $X_1, X_2, \ldots, X_n$. But while the sets of bits representing X_1 and X_2 where disjoint for two players, this is not generally the case for three or more players. In fact the only requirement for how the domain X is split for multiple players is that no single bit of the input domain is "known" to all players and all bits representing the input domain is "known" to at least one player. Therefore there might be bits of information "known" to only one player, bits of information "known" to a subset of players or bits of information "known" to all players except one.

Communication between two players is simple when each player can only communicate with one other player. For three or more players the communication becomes more difficult as each player has a range of options when sending a message. In general there are three modes of communication:

- Point-to-point - This corresponds to sending a message to a single player.
- Multicast - This corresponds to sending a message to set of players.
- Broadcast - This corresponds to sending a message to all players.

Different networks support different modes of communication. For example, in a wireless network it is only possible to use broadcasting of messages to all players within range. In a TCP/IP network the mode of communication

is point-to-point communication, with some support for multi-cast communication. In [KN06] they use a broadcast model for communication between players and a "number on the forehead" model is used for distributing the input domain. That means that the player P_i will "know" all bits of the input domain X except the bits representing input x_i. This definition is not very useful for multiparty computation, because in multiparty communication each player will try to keep as much information as possible private and will only share information from the input domain with other players in such a way that no information about the input is revealed. Therefore it is likely that the bits representing each player's input domain is unique.

If two (or more) players "know" some information, then both players should have to contribute the same information. This enables all players to compare these two inputs and to verify that both players have communicated the same value. This comparison keeps both players honest, because if the two inputs where not equal then one of the players had given an incorrect input value.

In many cases a secret sharing scheme is used to share input in such a way that no information about the input is revealed. It is then important that communication among players is private. In practice, multiparty communication will be performed using TCP/IP networks. Therefore communication is best modeled using point-to-point communication, as this will correspond closest to how communication is done in the real world. On the other hand, it is not important to model other aspects of TCP/IP networks such as delay, packet size, packet loss, packet ordering, etc. The model assumes that there are error free private channels between each set of players.

Therefore communication between multiple players in this model can be summarized using the following definition:

Definition 4. *For n players the input domain X will be split into n domains labeled $X_1, X_2, \ldots, X_n$. This split is done in such a way that player P_i only "knows" $x_i \in X_i$, where $i \in \{1, \ldots, n\}$. Each player can only communicate with other players using point-to-point communication over an error free private channel.*

Using the definition above we see that for example using four players P_1, P_2, P_3 and P_4. If player P_1 communicated one bit of information to player P_2 and player P_3 communicated one bit of information to player P_4, then two bits of information was communicated. In the real world this communication could be done in parallel, to represent this in the model, the concept of time and parallel communication is introduced into the model.

Definition 5. *One bit of information can be communicated between one pair of players using one unit of time. Communication can be done in parallel between all pairs of players at the same time.*

For example if all n players sends r-bits of information to all $n-1$ other players then $n(n-1)r$-bits of information is communicated, but only r units of

time is used. Each player is therefore assumed to have $n-1$ private channels to all other players (and one private channel to and from itself).

To make the definition for multiple players complete, the definition for the protocol will have to be expanded to n players. Because the model uses point-to-point communication and not broadcast communication, each player will have an independent binary tree. Also the cost of the protocol is the height of the highest tree.

Definition 6. *A protocol $\mathcal{P}$ over domain $X_1 \times X_2 \times \cdots \times X_n$ with range Y is a binary tree where each internal node v is labeled by of the functions $a_{1v} : X_1 \to \{0,1\}$, $a_{2v} : X_1 \to \{0,1\}$, ... or $a_{nv} : X_1 \to \{0,1\}$, and each leaf is labeled with an element $y \in Y$.*

The value of the protocol $\mathcal{P}$ on input $(x_1, x_2, \ldots, x_n)$ is the label of the leaf reached by starting from the root, and walking on the tree. At each internal node v labeled by a_{1v} walking left if $a_{1v}(x_1) = 0$ and right if $a_{1v}(x_1) = 1$, and at each internal node labeled by a_{2v} walking left if $a_{2v}(x_2) = 0$ and right if $a_{2v}(x_2) = 1$. The cost of the protocol $\mathcal{P}$ on input $(x_1, x_2, \ldots, x_n)$ is the length of the path taken on input $(x_1, x_2, \ldots, x_n)$. The cost of the protocol $\mathcal{P}$ is the height of the tree.

The definition for communication complexity is unchanged for n players when $n > 2$, except that the input changes from $X_1 \times X_2$ to $X_1 \times X_2 \times \cdots \times X_n$.

The definition of rounds of communication will also changed for n players when $n > 2$, because of the introduction of point-to-point communication and to include the concept of time.

Definition 7. *A k round protocol is a protocol where players send k messages to each other before determining the output. In each round all communication between pairs of players can be done in parallel, but all internal computations can only be done between rounds. The k-round protocol is denoted $D^k(f)$, which is the best k round protocol for f.*

The simplest protocol is now for player P_1 to send x_1 to P_n, P_2 to send x_n to P_n and so on. P_n can then compute $y = f(x_1, x_2, \ldots, x_n)$ and send y back to all players. We therefore have that

$$D(f) \leq log_2|X_1| + log_2|X_2| + \cdots + log_2|X_{n-1}| + (n-1)log_2|Y| \qquad (2)$$

The time used to for this communication is

$$T \leq max(log_2|X_1|, log_2|X_2|, \ldots, log_2|X_{n-1}|) + log_2|Y| \qquad (3)$$

This is a two round protocol, as the first round is used for all players to send information to P_n and the next round is used for player P_n to send information to all other players. The current model for multiple players also allows for the possibility of a one round protocol. This consists of all players sending their input to all other players in parallel. After all players have received all input

then each player individually can compute $y = f(x_1, x_2, \ldots, x_n)$. This differs from the two party model because communication between two players can now go simultaneous in both directions.

2.5 Modeling Information Sharing

One of the key ideas in secret sharing and homomorphic cryptography is that each player's input is kept private from all other players. This idea stands in stark contrast to communication complexity where all input is shared freely between the players. In order to model secret sharing, the model will have to reflect the fact that the input is kept secret. This is done by examining how secret sharing schemes and homomorphic cryptography keeps the input private, without going into the details of each individual scheme. Readers who are not familiar with secret sharing and homomorphic encryption can read Section 3 for more details.

Informally one could say that the input is kept private by "mixing" it with random values, until the information that is sent to the other players looks completely random. A more formal explanation is that each player uses a function. This function takes the information that is going to be sent to the other players and some random information as input. The output of the function is a set of *shares* where each player gets one share. The output should be such that each share is either completely random or a value statistically indistinguishable from a random value, so that each player might not learn anything about the original input. In secret sharing, each share is a different value and only certain sets of players can reconstruct the original information, while in homomorphic cryptography all shares are the same value and only the player(s) who know the private key can reconstruct the original information. When information is shared using functions that are not information theoretically secure the players do not have unlimited computational power. For example when information is shared using homomorphic cryptography, the assumption is that the players cannot decrypt the encrypted information.

The input domains X_i and output range Y have be represented using bits. The model will now shift from focusing on individual bits and will instead focus on variables. This leads to the following definition:

Definition 8. *Player P_i only "knows" $x_i \in X_i$, X_i can be represented as a set of variables V_{Xi} and x_i can be represented as $v_{Xi} \in V_{Xi}$ for all $i \in \{1, \ldots, n\}$.*

Instead of all players computing a common output Y, each player now only gets its predefined set of output variables and this leads to the following definition:

Definition 9. *The range of output Y is spilt into n ranges such that $Y = Y_1 \times Y_2, \times \ldots \times Y_n$. When the protocol is finished player Y_i will only "know" $y_i \in Y_i$. Y_i can be represented as a set of variables V_{Yi} and y_i can be represented as $v_{yi} \in V_{Yi}$ for all $i \in \{1, \ldots, n\}$.*

The input is kept secret, therefore each variable is converted to a set of shares and only the shares are sent to other players. The following four assumptions are being made about shares:

- All shares are assumed to be of the same length (l-bits long).
- One variable is converted to n shares, and each player receives one share for each variable.
- A player that wants to convert a set of shares into an output variable will need one share from each player (for a total of n shares).
- When a player sends a share to himself/herself, this costs l-bits of communication.

Without going into details, the first and second assumptions are true for many secret sharing schemes. The third assumption that all shares are needed to convert shares to output variables is true for some secret sharing scheme, but in general, an output variable can be computed once a threshold number of shares are received. On the other hand, a player who is going to reveal an output variable will want to receive shares from all players, so that it can discover if one of the players is cheating or sending false information. The fourth assumption challenges breaks one of the original assumptions: The assumption that only communication between players is counted. The reason for including this fourth assumption is that it keeps the model more inline with the included papers. It also eases the calculations when it comes to computing communication cost.

A function f converts the input domain into the output range. When information is communicated between the players, the computations involved in computing the function f are called *multiparty computation*. Although the function f can be expressed using all kinds of operations, only a limited set of basic operations are available in multiparty computations. Therefore the function f must be expressed using a more limited set of basic operations. The following assumptions are made about which basic operations exist and their associated cost:

- The basic operations are sharing a variable, revealing a shared variable, addition, multiplication and negation.
- The cost of sharing a variable and revealing a shared variable is nl-bits and cost one round of communication.
- There is no communication cost associated with addition, negation or scalar multiplication (multiplication with a fixed value). These operations do not require communication between the players.
- The communication cost of multiplying two shared variables is equal to each player sharing one input variable.

Given these definitions and assumptions, we see that the communication costs of sharing an input variable or revealing an output variable is nl-bits and cost one round of communication. The communication cost of multiplying two shared variables is equal to $n(nl) = n^2l$-bits of communication and one round of communication.

The operations of sharing, revealing or multiplying shares variables can be done in parallel, within the same round of communication as long as there is no need to do internal computations on the shares.

2.6 Further Refinements

Thus far the model provides a basic framework for modeling the amount of communication used for multiparty computations. As multiparty computation is communication intensive, it is important to find optimizations that can improve the efficiency of multiparty computations. One refinement that can be done with the current model is to split the players into three groups corresponding to the three phases in multiparty computation.

One group of players contribute with input, one group compute the function f and one group get some part of the output. A player must be part of at least one group, but might not be part of all groups. As the amount of communication depends upon the number of players computing the function f, it is important to make this group of players as small as possible. On the other hand fewer players in the computation phase might affect the security negatively. The security of each secret sharing scheme will be examined in the next section.

3. Secret Sharing Schemes

3.1 Introduction

This section will examine secret sharing schemes in more detail, by first giving an overview over what secret sharing is and some of the mathematical definitions proposed. Then three secret sharing schemes will be examined in more detail. These three secret sharing schemes will then be used again in the next section to show how they are useful for multiparty computation. This section will also describe homomorphic encryption.

Imagine a vault containing some secret value. The vault is locked, not with one key, but with a set of keys in such a way that a certain subset of the keys are needed to unlock the vault. What complicates this illustration is that the keys are also part of the original vault, since the keys are made based on the secret value. Another way to view secret sharing schemes is to think of a function that takes some input value x and splits it into a set of shares (values) $\{x_1, x_2, \ldots, x_n\}$, such that each individual share is (almost) a random value. In addition to the function that splits the value into shares, there is also the inverse function, i.e. given some subset of shares the inverse function can reconstruct the original value.

When discussing secret sharing schemes, one often comes across the role of a dealer. A dealer is someone who has some secret input, but will not take part in the multiparty computations. When using a dealer, the dealer takes some secret value and splits that value into a set of shares. The dealer then gives each player one share. The dealer has a role only when sharing values and not when recombining shares. The dealer will always be one of the players in this thesis.

Using secret sharing for multiparty computation works out nicely when there are three or more players, but when there are only two players there is the problem that revealing one share of information can lead to leaking all of the secret information. This comes from the fact that if there are only two players then the function $y = f(x_1, x_2)$ consists of only one unknown value for each player, so the equation can be solved. Therefore, in the case where there are only two players, we use homomorphic encryption instead of a secret sharing scheme. The roles of the two players can then be defined so that one player arranges the calculations and the other player performs as a multiplication and decryption oracle.

Many secret sharing and homomorphic encryption schemes have been proposed: additive secret sharing, Shamir's secret sharing scheme [Sha79], hierarchal secret sharing schemes [FP09], LISS [DT06], the Paillier cryptosystem [DJ01], replicated secret sharing, DNF-based secret sharing, among others. This thesis will examine the additive secret sharing, Shamir's secret sharing scheme, and the Paillier cryptosystem in more detail. These three are chosen because they are easy to understand and to implement. This gives the reader an easy transition from the theoretical ideas behind secret sharing to practical applications that computers can run.

3.2 Definitions

There are many kinds of security properties of a secret sharing scheme. The overall goal of a secret sharing scheme is to make it difficult to reconstruct the original value without authorized involvement.

One way of categorize the security of a secret sharing scheme is to relate it to the computational power of the adversary. For example, a scheme can be categorized according to how difficult it would be for an adversary to reconstruct the original value using only one share. The following three categories are based on the categories in [KLR06].

Perfect security This category requires that a share cannot be distinguished from a random value. Therefore even a computationally unbounded adversary cannot compute the original value.

Statistical security This category requires that a share is statistically indistinguishable from a random value. It should be computationally too costly for the adversary to collect enough different shares of the same value to compute the original value with some non-negligible probability.

Cryptographic security The share is an encryption of the original value. The security in this category relies on the assumed hardness of the underlying computational problem used for encryption. The adversary is therefore restricted to running in some probabilistic polynomial-time algorithm.

Another way to categorize the security of secret sharing schemes is to see which subsets of players can reveal the information and which players cannot reveal the information contained in the secret sharing scheme. All subsets of the set of all players are labeled either as a *qualified* or a *forbidden* set, with the following definitions:

Definition 10. *An input value s is split into shares using some secret sharing scheme. A qualified set is a set of players that are allowed to reconstruct the value s based on their shares. A forbidden set is a set of players that are not allowed to reconstruct the value s based on their shares.*

Assume that a secret sharing scheme exists that protects against any adversarial structure Γ under this definition. This means that the secret sharing scheme will protect the input from any set of adversaries that is included in the forbidden set and that the input is not protected from any set of adversaries that are in the qualified set. For more on adversarial structures see [Tho09].

This thesis will only examine simple secret sharing schemes, where the qualified and forbidden subsets only depend on the number of players in each set. These are known as (t, n) secret sharing schemes and can be defined as follows:

Definition 11. *An input value s is split into n shares using some (t, n) secret sharing scheme. This implies that any set containing t or more shares can reconstruct the value s, any set with fewer than t shares cannot reconstruct the value s.*

Finally, the secret sharing can be made verifiable to protect against a malicious dealer. A malicious dealer is a player that is going to share a value, but does so incorrectly. A malicious dealer might perform the secret sharing correctly internally, but then modify some of the shares before sending them out. A verifiable secret sharing scheme is in addition to an ordinary secret sharing scheme, that includes something which the dealer commits to. All the players can then verify that they have received the same commitment and that this commitment proves that the sharing was done correctly. For more on verifiable secret sharing schemes, see for example Feldman's scheme [Fel87].

Notation The following notation will be used for secret sharing

- A value s is an element (integer) in a finite field $\mathbb{Z}_p$, where p is a large prime number.
- There are n players $P_1, P_2, \ldots, P_n$.

- s_i is the share destined for player i, for $i \in \{1, \ldots, n\}$.
- $[s]$ is the set of shares that combined reveals the value s.
- A value s can be shared in many different sets of shares $[s]$, thereby making it impossible to go from one share s_i to the value s.

3.3 Additive Secret Sharing Scheme

3.3.1 Introduction

An additive secret sharing scheme is a very simple secret sharing scheme, because the secret is shared as the sum of shares. The simplicity of the scheme makes it easy to explain the concept of secret sharing to people unfamiliar with secret sharing. It is also good if all players are honest, but the scheme is very vulnerable to modifications by malicious parties, therefore this scheme is not very useful in practice.

The additive secret sharing scheme will be shown over $\mathbb{Z}_p$, where p is a large prime, the scheme can also be extended to computations over $\mathbb{Z}$, but this will not be shown in this thesis.

3.3.2 Sharing

A value s is shared by by giving all the players P_i, except the last player a random value s_i over $\mathbb{Z}_p$ as their share, for $i \in \{1, 2, \ldots, (n-1)\}$. The share s_n for the last player P_n is computed as follows $s_n = s - \sum_{i=1}^{n-1} r_i \mod p$.

3.3.3 Recombining

The shares are recombined by each player revealing their share and computing the sum. This also serves as the definition for additive secret sharing scheme.

Definition 12. *An additive secret sharing scheme is a secret sharing scheme where each player P_i is given a share s_i of the secret s such that the following equation holds:*

$$s = \sum_{i=1}^{n} s_i \mod p \tag{4}$$

3.3.4 Security

The scheme has perfect security when computed over $\mathbb{Z}_p$, and statistical security when computed over $\mathbb{Z}$. The security comes from the fact that all players are given random values except for one player, which is computed based on the secret and the other shares. This can be seen as a form of one-time pad (OTP).

The scheme is very secure in one sense, because all n shares are needed to reveal the secret shared value s. On the other hand, all players can modify

their share and thereby modify the secret value in a predictable manner. In other words this scheme is very secure against passive adversaries (adversaries that are curious but follow the protocol), but the scheme is not secure against active adversaries (adversaries that might modify their output and not follow the protocol).

3.4 Shamir's Secret Sharing Scheme

3.4.1 Introduction

Shamir's secret sharing scheme was proposed in 1979 [Sha79] and it is still a very useful scheme. It is simple to understand, because it is a (t, n) secret sharing scheme, and the bit-length of the shares are equal to the bit-length of the largest secret shared value making it a very efficient scheme.

3.4.2 Sharing

A value s is shared as $[s]$, each player $P_i \in \{P_1, \ldots, P_n\}$ is given a share s_i over $\mathbb{Z}_p$. These shares are computed as points over a polynomial. Each player P_i is associated with a unique identifier $Q_i \in \mathbb{Z}_p$. It is normal to set $Q_i = i$, but in general the only restriction on Q_i is that $Q_i \neq Q_k$, for all $i \neq k$, where $i, k \in \{1, \ldots, n\}$.

Definition 13. *Shamir's secret sharing scheme is a (t, n) secret sharing scheme computed over a field $\mathbb{Z}_p$. Each share s_i computed as the point $f(Q_i)$, where $f(x)$ is a polynomial of degree t.*

$$f(x) = s + r_1 x + r_2 x^2 + \cdots + r_{t-1} x^{t-1} \mod p$$

Where the coefficients $r_1, r_2, \ldots, r_{t-1}$ are random values.

3.4.3 Recombining

When recombining shares, the player that is going to reveal the value receives all the shares connected to that value. The player does not know the random values in the original polynomial, therefore the player has to compute an interpolation polynomial. There are many ways of computing an interpolation polynomial, but an efficient way of computing an interpolation polynomial is to compute the first row of the inverse Vandermonde matrix based on the shares s_i. The following equations for determining the inverse Vandermonde assumes that all the shares are known. As only t shares are needed to reconstruct the secret shared, the Vandermonde matrix can be reduced accordingly. A player should compute the inverse Vandermonde matrix based on many

different sets of t players to verify that the other players are honest.

$$\begin{bmatrix} 1 & Q_1 & Q_1^2 & \dots & s_1^{n-1} \\ 1 & Q_2 & Q_2^2 & \dots & P_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & Q_n & Q_n^2 & \dots & P_n^{n-1} \end{bmatrix}^{-1} = \begin{bmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_n \\ \mu_{1,1} & \mu_{2,1} & \dots & \mu_{n,1} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{1,n-1} & \mu_{2,n-1} & \dots & \mu_{n,n-1} \end{bmatrix} \tag{5}$$

The value s can then be recombined using the following equation

$$s = \sum_{i=1}^{n} \lambda_i s_i \mod p \tag{6}$$

3.4.4 Security

The security of Shamir's secret sharing scheme is based on the fact that a function of degree $t-1$ over $\mathbb{Z}_p$ cannot be reconstructed with less than t points on the function. The security is information-theoretic as all secret values s are equally likely given only $t-1$ points on the function.

3.5 Paillier Cryptosystem

3.5.1 Introduction

The Paillier cryptosystem is a method for key generation, encryption and decryption of information (transforming clear text into encrypted information and vice versa). The Paillier cryptosystem was first proposed by Paillier in 1999 [Pai99]. The version described below describes the original cryptosystem, but it also includes a generalized version by Damgård and Jurik [DJ01], called the Damgård-Jurik cryptosystem. This cryptosystem is an example of a homomorphic cryptosystem and these cryptosystems can be used for secret sharing, especially in the two players setting. This will be examined in the next section as this section will only focus on the cryptosystem.

The original Paillier cryptosystem is computed over $\mathbb{Z}^*_{n^2}$ and the generalized version is expanded to work over $\mathbb{Z}^*_{n^{s+1}}$, where $s \geq 1$. The generalized version includes Paillier's cryptosystem as a special case for $s = 1$. The multiplicative group $\mathbb{Z}^*_{n^{s+1}}$ is homomorphic to two groups $G \times H$, where G is a cyclic group of order n^s and H is isomorphic to $\mathbb{Z}^*_n$. Very loosely, we can say that the secret message is hidden in the group G and the group H contains a random value, so that two encryptions of the same secret message do not result in the same encrypted value.

3.5.2 Key Generation

The key generation protocol proceeds as follows:

1 Two large prime numbers p and q are created. The value n, Euler's totient function $\phi(n)$ and Carmichael's function are computed as (lcm

is the least common multiplier):

$$n = pq \qquad \phi(n) = (p-1)(q-1) \qquad \lambda(n) = lcm(p-1, q-1)$$

Before continuing we also need to define a function L as:

$$L(x \mod n^{s+1}) = \frac{x-1}{n} \mod n^s \tag{7}$$

This equation is used to easily compute i from $(1+n)^i \mod n^{s+1}$. For $s = 1$ this is simple because $(1+n)^i \mod n^2 = 1 + in \mod n^2$. Therefore:

$$L((1+n)^i \mod n^2 = \frac{1+in-1}{n} \mod n = i \tag{8}$$

For $s > 1$ the function becomes more complicated, for a full description see [DJ01].

2 The next step is to select an element $g \in \mathbb{Z}^*_{n^{s+1}}$. The element g must be an element of order $n^s\alpha$, where $\alpha \in 1, \ldots, \lambda$. The choice of g does not affect the security, so choosing $g = n + 1$ will always result in a valid g. For ease of computation $g = 2$ is another good choice (if it is valid).

3 The final step is setting the decryption value of d, the easiest choice is setting $d = \lambda$, but any $d = 0 \mod \lambda, d = 1 \mod n^s$ will work.

3.5.3 Encryption

Given a plaintext $m < n^s$ and a random value $r \in \mathbb{Z}^*_{n^{s+1}}$ the encryption is given as:

$$c = g^m r^{n^s} \mod n^{s+1} \tag{9}$$

Depending on the implementation it might be more effective if the value g is set to a small integer. The computation of c can be sped up by pre-computing the values g^{2^i} for all $i < \log(n^{s+1})$.

3.5.4 Decryption

Given ciphertext c, the decryption key $d = 0 \mod \lambda$ and g such that $g = (1+n)^j x \mod n^{s+1}$, where j is relative prime to n and $x \in H$, first compute $c^d \mod n^{s+1}$.

$$c^d = (g^m r^{n^s})^d = ((1+n)^j m x^m r^{n^s})^d = (1+n)^{jmd}(x^m r^{n^s})^d = (1+n)^{jmd} \tag{10}$$

It is important to remember that $(1+n)$ is a value of order n^s, we also have that $x^d = 1$ and $r^{dn^s} = 1$. Apply the function L, compute $jmd \mod n^s$ from $(1+n)^{jmd}$. Replacing c with g and applying the same method produces the value $jd \mod n^s$. The cleartext m is then extracted by computing $m \mod n^s = (jmd)(jd)^{-1} \mod n^s$.

For $s = 1$ this is simplified to the following equation:

$$m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \tag{11}$$

The decryption key might be shared among multiple parties. The protocols for decryption and verifying the decryption is given in [DJ01]. Essentially the decryption key is shared using Shamir's secret sharing scheme, and the parties have to compute a distributed exponentiation.

3.5.5 Security

The security of the Paillier cryptosystem is based on the *Decisional Composite Residuosity Assumption* (DCRA). The DCRA problem is the problem of finding n-th residues modulo n^2.

Definition 14. *A number z is said to be a n-th residue modulo n^2 if there exists a number $y \in \mathbb{Z}^*_{n^2}$ such that*

$$z = y^n \mod n^2 \tag{12}$$

This definition is taken from [Pai99]. The same paper shows that the decisional composite residuosity problem is as hard as the problem of factoring n.

3.6 Changing Secret Sharing Scheme

3.6.1 Introduction

Most often secret shared values are shared over the same finite field and using the same secret sharing scheme. This allows for efficient multiparty computations and is practical. But in some cases there might be a need to change the secret sharing scheme or the finite field that the computations are computed over. For these cases there are methods whereby secret shared values can be transformed from one secret sharing scheme to another secret sharing scheme without revealing the value.

This section will only cover transformations between the three secret sharing already mentioned. In addition we assume that the players are honest and that there are protocols whereby a $[s] > k$ can be computed without revealing the result, where $[s]$ is a secret shared value and k is a constant. For a more general approach see [CDI05].

3.6.2 Transforming SSSS to ASSS and ASSS to SSSS

Transforming from additive secret sharing scheme (ASSS) to Shamir's secret sharing scheme (SSSS) and vice versa is simple and requires no interaction between the players, if the finite field is kept the same. This transformation can be seen from the following two equations:

When recombining a value based on Shamir's secret sharing scheme, a player gathers all shares from the other players and computes the equation:

$$s = \sum_{i=1}^{n} \lambda_i s_i \mod p \tag{13}$$

The equation for recombining a value based on an additive secret sharing scheme is not very different

$$s = \sum_{i=1}^{n} s_i \mod p \tag{14}$$

To transform a secret shared value from Shamir's secret sharing scheme to an additive secret sharing scheme each player individually computes their additive share $s_i^* = \lambda_i s_i \mod p$ based on their respective share s_i and lambda value. Going the other way, the players divide by λ_i instead of multiplying by λ_i.

3.6.3 Transforming ASSS and SSSS to Paillier Cryptosystem

If the transformation begins with a value secret shared using Shamir's secret sharing scheme, then the shares are first transformed into shares using an additive secret sharing scheme.

Transforming between an additive secret sharing scheme and the Paillier cryptosystem can be accomplished by each player encrypting their share s_i using the Paillier cryptosystem. The encrypted values can then be shared among the players so that all players can compute $E(s) = \sum_{i=1}^{n} E(s_i) \mod p$ where $E(s)$ is an encryption of s.

This works fine except that the Paillier cryptosystem is computed over a ring $m = p^* g^*$, where p^* and q^* are primes, while the additive secret sharing scheme is computed over a prime p. Therefore the value $E(s)$ can in fact be an encryption of $E(s + tp)$ where $0 \leq t < n$, where n is the number of players. We know that the initial secret shared value $s < p$. Therefore the players can subtract p from the resulting encryption (up to n times), until the result is less than p.

3.6.4 Transforming Paillier Cryptosystem to ASSS and SSSS

For this transformation the players pick a random value r that is both secret shared over Shamir's (or additive) secret sharing scheme and encrypted using the Paillier cryptosystem. There are many ways of picking such a random value, ensuring that it is random. Picking such a random value is also further complicated because the two schemes are computed over different rings. To not complicate things, we assume that the players have computed such a random value, for example this can be done by each player choosing a random value r_i in the smallest of the fields and sharing that value using both additive secret

sharing and the Paillier cryptosystem. Then a random value can be created by adding the shared values together $r = \sum r_i$.

Once such a random value is found the players can add r to the Paillier encrypted value s that we are trying to transform, and reveal the value $r + s$. Revealing $r + s$ does not reveal any information about s if r is either a totally random value over the whole field or if r is much greater than s ($r >> s$). Once $r + s$ is revealed each player can compute $(r + s) - r$ where r is shared using additive secret sharing.

4. Multiparty Computation

4.1 Introduction

The previous section examined three secret sharing schemes and showed how to share values, reveal the secret shared values, and the security of each scheme.

Secret sharing schemes without multiparty computation are just operations on static values. The value that is secret shared is the same that is revealed later. One way to visualize a secret sharing scheme is to imagine putting the information that is to be secret shared into a multilock safe and then handing over one lock key to each player. The players cannot extract any information from the key itself. The players have to cooperate, collect the keys, and open the safe to reveal the secret value inside. Multiparty computations go further than this, because these methods allow the players to modify the values while locked in the safe, without opening the safe. The players can also combine values of many safes to construct a new safe containing a computational result of the values in the original safes. This can all be done without opening up any of the safes.

Multiparty computation consists of only two basic operations. The secret shared values can be added or multiplied with other secret shared values or values known to all players. Although addition and multiplication do not seem like much, these two basic operations can be used together with sharing and revealing to form more complicated operations, as we will see in later sections.

First we examine the concept of adversaries. Then we examine the additive secret sharing scheme, Shamir's secret sharing scheme and the Paillier cryptosystem, and show how addition and multiplication can be accomplished in these secret sharing schemes. This section also includes an introduction into a fully homomorphic encryption scheme. Fully homomorphic encryption schemes are schemes where both addition and multiplication can be done without interaction. It was long thought that this was not possible, but results by Craig Gentry [Gen09] have shown this is possible. This section concludes with some optimizations for Shamir's secret sharing scheme, these optimizations might be applicable to other secret sharing schemes as well.

4.2 Adversaries

To examine the security in multiparty computations the players can be divided into three types: honest players, passive adversaries and active adversaries.

- Honest players follow the agreed upon protocol as specified and complete the protocol with no errors. An honest player does not try to cheat or find out anything more about the secret shared values than what he or she is supposed to know.

- Passive adversaries will follow the agreed upon protocol as specified, but will be interested in learning more about the secret shared values than what he or she is supposed to know. The passive adversaries may collude with other passive or active adversaries by sharing information with them.

- An active adversary is a passive adversary, but might also deviate from the agreed upon protocol and send arbitrary information to other players. An active adversary might also try to falsify or ruin the calculations for other players.

The model in Section 2 assumes that the communication channels between the players are error free. Errors arising from problems in the communication channels might therefore be modeled as adversaries. When there is a loss of a communication channel between players, can be modeled as an attack by a passive adversary. Communication errors that are not caught by the communication layer, can be modeled as attacks by active adversaries. In fact, there is in theory no way to distinguish between an honest party using a communication channel that introduces transmission errors in the messages, and a perfect channel with an active adversary that sends modified messages. A message sent on an error-prone communication channel will often be accompanied with error detection checksum values, which will enable the receiver to discard messages with errors. False messages from active adversaries, on the other hand, can have correct checksum values, but not necessarily so.

4.3 Additive and Shamir's Secret Sharing Schemes

4.3.1 Introduction

This section will show how addition can be done using the additive scheme and Shamir's secret sharing scheme. No multiplication protocol is given for the additive secret sharing scheme, as this can be accomplished by transforming the additive shares into Shamir's shares, and doing the multiplication using Shamir's secret sharing scheme, and finally transforming the shares back into the additive secret sharing scheme.

4.3.2 Addition for Additive Secret Sharing

Given two secret shared values $[a]$ and $[b]$, the secret shared sum $[c] = [a]+[b]$ is computed by each player adding their respective shares a_i and b_i. This can be seen from the following equation:

$$[c] = [a]+[b] = \sum_{i=1}^{n} a_i \mod p + \sum_{i=1}^{n} b_i \mod p = \sum_{i=1}^{n} (a_i + b_i) \mod p = [a+b] \tag{15}$$

4.3.3 Addition for Shamir's Secret Sharing Scheme

Theorem 1. *Given two secret shared values $[a]$ and $[b]$, the secret shared sum $[a+b]$ is computed by each party adding the shares together $[a]+[b]$.*

Proof. First we have that the secret shared values a and b are shared as polynomials over a finite field.

$$f_a(x) = a + r_{1,a}x + \cdots + r_{t-1,a}x^{t-1} \mod p$$
$$f_b(x) = b + r_{1,b}x + \cdots + r_{t-1,b}x^{t-1} \mod p$$

Each player gets a share corresponding to one point on this polynomial. If the shares are added together, the players get shares which are points on the polynomial:

$$f_{a+b}(x) = a + b + (r_{1,a} + r_{1,b})x + \cdots + (r_{t-1,a} + r_{t-1,b})x^{t-1} \mod p$$

This can be simplified to the following equation:

$$f_{a+b}(x) = a + b + r_{1,v}x + \cdots + r_{t-1,v}x^{t-1} \mod p, \tag{16}$$

where $r_{i,v} = r_{i,s} + r_{i,t}$ for all i. From this we can see that the polynomial $f_{a+b}(x)$ has constants $a+b$ and is of the correct degree. □

4.3.4 Multiplication for Shamir's Secret Sharing Scheme

In Shamir's secret sharing scheme the secret values a and b are shared using two polynomials of degree t, this means that each player P_i has the values $f_a(i)$ and $f_b(i)$. The goal of this protocol is for each player to get a share $f_{ab}(i)$ which is also a polynomial of degree t.

$$f_a(x) = a + r_1x + r_2x^2 + \cdots + r_tx^t$$
$$f_b(x) = b + s_1x + s_2x^2 + \cdots + s_tx^t$$
$$f_{ab}(x) = ab + u_1x + u_2x^2 + \cdots + u_tx^t$$

where the coefficients $r_1, r_2, \ldots, r_t$, $s_1, s_2, \ldots, s_t$ and $u_1, u_2, \ldots, u_t$ are random values.

The protocol multiplication consists of each player multiplying the two shares $f_a(i)$ and $f_b(i)$ together. The resulting polynomial is called $h(x)$ and is of degree $2t$. Each player P_i computes a share $h_i = h(i)$ of this polynomial.

$$h(x) = ab + v_1 x + v_2 x^2 + \cdots + v_{2t} x^{2t} \tag{17}$$

where the coefficients $v_1, v_2, \ldots v_{2t}$ depends on the coefficients $r_1, r_2, \ldots, r_t$ and $s_1, s_2, \ldots, s_t$.

The next step consists of all players P_i share their h_i values with the other players using the normal method of sharing. Thus a value h_i becomes the constant term in a new polynomial $h_i(x)$.

$$h_i(x) = h_i + w_1 x + w_2 x^2 + \cdots + w_t x^t$$

So the shares of all $h_i(x)$ are distributed. This results in that the player P_j receives the share $h_i(j)$ from player P_i. After receiving at least $2t + 1$ shares the players can recombine the shares. The resulting value is a share of the polynomial f_{ab} of degree t. This protocol only works for honest players. For a proof of this protocol and for verifiable protocols, that work against active adversaries see [GRR98].

A summary of the protocol

1 Each player P_i has the shares $f_a(i)$ and $f_b(i)$ and computes the product of the two shares $f_a(i)f_b(i) = h(i) = h_i$.

2 The value h_i is then secret shared using a random polynomial of degree t, and the shares are distributed to the other players. The player P_j receives the value $h_i(j)$.

3 Each player P_j can then locally compute $f_{ab}(j)$ which is his share of the value ab, this is done by computing the polynomial:

$$f_{ab}(j) = \sum_{i=1}^{2t+1} \lambda_i h_i(j) \tag{18}$$

where λ_i is the first row of the inverse Vandermonde matrix.

4.4 Paillier Cryptosystem

4.4.1 Introduction

The Paillier cryptosystem was introduced in Section 3.5, but not described how it can be used for multiparty computation. There is a fundamental difference between how multiparty computation performed by using a public key cryptosystem versus those that utilize a secret sharing scheme. In secret shar-

ing schemes each player gets a share value that is unique to that player. Using a cryptosystem, the shares become cipher texts of the secret value. Obviously, players that have access to the private decryption key can reconstruct the clear text, that is, the original value. Therefore it is important that only the final result of the multiparty computation is communicated to a player holding the private decryption key.

Using a cryptosystem as a basis for multiparty computation makes it possible to do multiparty computations between only two players. These two players will have different roles. One role will be called the *evaluator*, and the other role will be called the *multiplication oracle*. The evaluator will receive the encrypted input values and will perform the addition operation. This role must not have access to the decryption key. The multiplication oracle will perform a multiplication protocol with the evaluator. The multiplication oracle must also be able to decrypt the output value(s), therefore this role must have access to the decryption key. The roles are split to ensure that the players handling the encrypted information has no way of decrypting it, and the player that has the decryption key does not have access to any of the information.

When this cryptosystem is expanded to more than two players, the roles will have to be distributed among the players. The only restriction when distributing the roles is that any player handling encrypted information should not have full knowledge of the decryption key. The decryption key in a cryptosystem might be shared among multiple parties, as was first proposed by Yvo Desmedt in 1988 [Des88]. When the decryption key is shared between a set of players, then they function as a multiplication oracle. Since none of them have full knowledge of the decryption key, it also implies that the same players might also function in the evaluator role. The communication between the players will then have to be structured so that the two roles do not overlap. Protocols for decryption and verifying the decryption in the Paillier cryptosystem with a distributed key can be found in [DJ01]. Essentially the decryption key is shared using Shamir's secret sharing scheme and the parties compute a distributed exponentiation.

The protocols are given only for Paillier cryptosystem, but in fact any cryptosystem can used for multiparty computation. Homomorphic cryptosystems are especially useful for multiparty computation. This is because in a homomorphic cryptosystem addition (or multiplication) of two encrypted values can be computed without decrypting them. In addition the cryptosystem should have ciphertexts that are indistinguishable under chosen-plaintext attack. That means that given two values a and b and and two encryptions of the same two values $E(a)$ and $E(b)$, it should not be possible for an attacker to distinguish which ciphertext corresponds to a given plaintext. (The RSA cryptosystem is not useful for multiparty computation as the same plaintext value always results in the same encrypted value.)

The protocols for addition and multiplication for the Paillier cryptosystem will be shown in the next subsections. The notation used is that $E(a, r)$ is an encryption of the value a and r is a random value.

4.4.2 Addition in the Paillier Cryptosystem

The Paillier cryptosystem is additive homomorphic. This means that if two ciphertexts are multiplied together, the corresponding clear texts are added. Given $E(a, r_1) = g^a r_1^{n^s}$, $E(b, r_2) = g^b r_2^{n^s}$ and $r = r_1 r_2$. This can then be easily seen from the following equation:

$$E(a, r_1)E(b, r_2) = g^a r_1^{n^s} g^b r_2^{n^s} = g^{a+b}(r_1 r_2)^{n^s} = g^{a+b} r^{n^s} = E(a + b, r) \quad (19)$$

4.4.3 Multiplication in the Paillier Cryptosystem

To perform multiplication of two encrypted values, the evaluator must transmit some information to the multiplication oracle which can then decrypt the information, compute a multiplication and encrypt the result before returning the results to the evaluator. $E(a, r_a)$ and $E(b, r_b)$ are the original encrypted values that the evaluator wants the multiplication oracle to multiply. These cannot be sent directly to the multiplication oracle, therefore the evaluator picks random values u, v, r_u, r_v and r_2. The evaluator then computes $E(a+u, r_a r_u)$ and $E(b+v, r_b r_v)$ locally and sends these values to the multiplication oracle. The multiplication oracle decrypts $E(a+u, r_a r_u)$ and $E(b+v, r_b r_v)$. Computes the multiplication $(a + u)(b + v)$ and encrypts the value. The multiplication oracle then returns the encrypted value $E((a+u)(b+v), r_1)$, where r_1 is a random value picked by the multiplication oracle.

The evaluator then uses the following equation can to compute the multiplication.

$$E(ab, r_{ab}) = E((a + u)(b + v), r_1) - E(a, r_a)v - E(b, r_b)u - E(uv, r_2) \quad (20)$$

Where $E(uv, r_2)$ can be computed by the evaluator.

For a more advanced protocol where the decryption key is split between multiple players see [DJ01].

4.5 Fully Homomorphic Cryptosystem

4.5.1 General overview

The first result for a fully homomorphic cryptosystem was shown by Craig Gentry [Gen09] in 2009, his idea was to work over ideal lattices. This thesis will present a simplification based on by Dijk et al. [vDGHV09], where the computations are done over the integers, making the cryptosystem easier to understand. Any such cryptosystem starts with a *somewhat homomorphic* cryptosystem leading up to the fully homomorphic cryptosystem. A *somewhat*

homomorphic cryptosystem is homomorphic both for addition and multiplication up to some limit. For example the Paillier cryptosystem is homomorphic for addition of encrypted values and it is also possible to perform multiplication with a clear text value. The limit for the Paillier cryptosystem is that the resulting value should not be larger than the modulus n, otherwise it is impossible to decrypt the value correctly. The same is the case for a somewhat homomorphic cryptosystem because a certain number of additions and multiplications can be performed. After that limit is reached then either the noice used to hide the encrypted value becomes too great, making it impossible to decrypt correctly, or the encrypted values become too large, making the scheme impossible work with.

The idea for a fully homomorphic cryptosystem is then to start with a somewhat homomorphic cryptosystem and then reveal some more information about the public key, making it possible to do computations for any number of additions and multiplications. The presentation in this thesis will be limited to the somewhat homomorphic case that works on bits $m \in \{0, 1\}$. For the fully homomorphic case see Dijk et al. [vDGHV09].

4.5.2 Somewhat Homomorphic Cryptosystem

A somewhat homomorphic scheme that works with individual bits $m \in \{0, 1\}$, can be written as follows:

- A secret key is a random odd η-bit integer p.

- To generate the public key, a set of values are chosen randomly from the set $[0, \frac{2^\gamma}{p}]$ and these values are called q_i for $i = 0, 1, 2, \ldots, \tau$. The values q_i are sorted so that q_0 is the largest value. Another set of random values are chosen randomly from the set $[-2^\rho, 2^\rho]$ and these values are called r_i for $i = 0, 1, 2, \ldots, \tau$. A set of values x_i for $i = 1, 2, \ldots, \tau$ are then computed as $x_i = q_i p + 2r_i \mod q_0$ and finally x_0 is computed as $x_0 = q_0 p + 2r_0$.

To encrypt a message $m \in \{0, 1\}$ in this cryptosystem, choose a random subset $S \subset \{1, 2, \ldots, \tau\}$ and a random integer $r \in [-2^\rho, 2^\rho]$ and set $E(m) = c = m + 2r + \sum_{i \in S} x_i \mod x_0$. Messages can then be also be decrypted as $D(c) = m = (c \mod p) \mod 2$.

4.5.3 Addition and Multiplication

Addition and multiplication of encrypted bits is then the same as addition and multiplication over the integers.

For addition, we can see the following equation.

$$E(m_1) + E(m_2) = (m_1 + 2r_1 + \sum_{i \in S_1} x_i \mod x_0) + (m_2 + 2r_2 + \sum_{i \in S_2} x_i \mod x_0)$$
$$= ((m_1 + m_2 \mod 2) + 2(r_1 + r_2 + m_1 m_2) + \sum_{i \in S_1 \cup S_2} x_i + \sum_{i \in S_1 \cap S_2} x_i) \mod x_0$$
$$= E(m_1 + m_2 \mod 2)$$

This final equality is not totally correct, because the random value r is now taken from a larger set of values $r \in [-2^{\rho+1}, 2^{\rho+1} + 1]$ and is not chosen uniformly over that set. Also the set of elements which are both in S_1 and S_2 are counted twice in the sum.

For multiplication, we can see that $D(E(m)) = (E(m) \mod p) \mod 2 = m$, this leads to the following equation:

$$D(E(m_1)E(m_2)) = ((E(m_1)E(m_2)) \mod p) \mod 2$$
$$= ((((E(m_1) \mod p) \mod 2)((E(m_2) \mod p) \mod 2)) \mod p) \mod 2$$
$$= ((m_1 m_2) \mod p) \mod 2 = m_1 m_2 \mod 2$$

This means that a decryption of the product of two messages is equal to the product modulo 2 of the two messages. This does not hold for multiple products as the noise r grows quickly and the size of the encryption roughly doubles with each multiplication. If the noise r becomes greater than p the decryption will be wrong.

4.5.4 Practicality

This somewhat homomorphic cryptosystem will be impractical to use because if the security parameter is λ then $\rho = \lambda$, $\eta \approx \lambda^2$, $\gamma \approx \lambda^5$ and $\tau = \gamma + \lambda$. Therefore the size of the public key must be approximately λ^{10}. For information on how to squash the decryption circuitry to make this somewhat homomorphic cryptosystem into a fully homomorphic cryptosystem, see Dijk et al. [vDGHV09].

4.6 Optimizing Multiparty Computation

4.6.1 Introduction

This section will give an overview of some improvements that can be done in multiparty computation. These improvements will focus on improvements using Shamir's secret sharing scheme, but similar improvements might also exist for other secret sharing schemes. For these improvements the number of players is denoted by n and the bit-length of a share is denoted by m.

4.6.2 Pseudorandom Secret Sharing

Many functions for multiparty computations, such as those in paper **B, C, F** require that the players have access to a source of random numbers. These random numbers must be secret shared but should be not known to any player. Creating secret shared random numbers can be done for all secret sharing schemes, but not all methods of creating random numbers will create perfectly random numbers. Using a scheme where pseudorandom secret shared values are created without interaction will greatly improve the efficiency of such protocols. For pseudorandom secret sharing see [CDI05].

4.6.3 Multiply and Reveal

Two variables $[a]$ and $[b]$ are secret shared and the function states that the two variables are to be multiplied together $[c] = [ab]$, before the variable c is revealed to all players. Using the standard model this will take $2n^2m$ bits of communication and two rounds of communication for the multiplication and revealing steps. This can be improved by multiplying and revealing in one step. To ensure that no information about $[a]$ or $[b]$ is revealed it is important to add a random or pseudorandom sharing of 0 to $[c]$ before revealing.

4.6.4 Lazy Shamir

Two vectors of variables $a_1, a_2, \ldots, a_k$ and $b_1, b_2, \ldots, b_k$ are secret shared and the function states that the inner product of the two vectors are to be computed.

$$S = \sum_{i=1}^{k} a_i b_i \tag{21}$$

The model states that this will take kn^2m bits of communication for the multiplication and addition step. The communication cost can be lowered to only n^2m bits of communication. This is done instead of sending shares after each multiplication. The shares are instead stored locally. The addition step is then done on the locally stored shares, and only the shares representing the sum are sent out. This trick can be done because each multiplication is actually a $2t$-threshold secret sharing (when the original secret sharing is a t-threshold secret sharing), and the shares are only sent out to reduce the sum from a $2t$ to a t-threshold secret sharing. Therefore it does not change the secret sharing scheme if the additions are done before sending or after sending the shares out.

5. A Complete Set of Operations

The operations sharing and revealing were introduced in Section 3. The operations addition and multiplication were introduced in Section 4. In addition to these four operations, a multiplication by -1 will result in a negation of

the original secret shared value. Together these five basic operations can form building blocks for constructing more complex operations. Papers **B** and **C** considers the "less-than" operation and **F** considers "bit-decomposition," but these are just two of the possible operations.

This section will explore in more detail how complex operations can be constructed from the set of five basic operations. For a comprehensive overview, the list of operations and statements will be based on lists of operators and statements used in a programming language. In particular the operations and statements given in this section are based on the Java programming language [Fla05]. The choice of programming language is arbitrary in that there are many programming languages that have the same abilities as Java to transform a function into a computer program that can be executed on a computer. The lists will be restricted so that for unary operators the operand must be a secret shared value and for binary operators only the case where both operands are secret shared values will be included in the lists. Operations where one operand is a secret shared value and one is a constant value will not be included although these operations are often much more efficient than operations where both operands are secret shared values. This section will not examine how efficient each operation is but only focus on how each operation can be implemented.

The list relies heavily on the operation *"bit-decomposition"*. This operation was first introduced in [DFK+06], and consists of transforming a secret shared variable into a list of variables, where each element in the list holds one bit of the original variable. For example, if the value 19 (expressed as 10011 in binary) is secret shared, then bit-decomposing this value creates an array of the secret shared elements $\{1, 0, 0, 1, 1\}$.

Operators that are short-hand methods of writing other operators, such as pre/post-increment/decrement (++, –), assignment with operator (*=, /=, %=, etc) and not equal (!=) are not included in the list. Less than or equal (<=), greater than (>), and greater than or equal (>=) can all be reformulated into operations using the less than (<) operator. "Do" loops can be reformulated to "while" loops, "for/in" collection iteration can be reformulated as "for" loops, and assertions can be rewritten as if statements. Computations involving float or double variables are also not included as these are best handled as Boolean circuits. (Circuits that only operate on the values 0 or 1.)

The tables follow standard notation for secret shared variables:

- $[a]$, $[b]$, $[c]$, etc. are secret shared variables. For Boolean operations it is assumed that the secret shared variables are Boolean variables.

- $max([a])$ denotes the maximum possible value in $[a]$. This is used in operations where the size of $[a]$ is not publicly known. For example in creating arrays of size $max([a])$ all players would have to create an array with the maximum possible number of array elements. Also if the size of the array is important to remember, then variable $[a]$ will have to be

Operator	Operation performed	Multiparty computation
.	object member access	internal
instanceof	type comparison	internal
=	assignment	internal
(type)	cast	internal
[[a]]	array element access	see comments
$(args)$	function (method) invocation	internal (args must be a fixed set of variables)
new [a]	object creation	internal
new [[a]]	array creation	create new (max[a]), see below
$-[a]$	negation (unary minus)	basic operation
$[a] + [b]$	addition	basic operation
$[a] - [b]$	subtraction	[a]+(-[b])
$[a] * [b]$	multiplication	basic operation
$[a]/[b]$	division	see comments
$[a]\%[b]$	modulo	$[a] - ([a]/[b])[b]$
[a]&&[b]	Boolean AND	$[a][b]$
[a]||[b]	Boolean OR	$[a][b] + [a] + [b]$
![a]	Boolean NOT	$1 - [a]$
$[a]^\wedge[b]$	Boolean XOR	$[a] + [b] - 2[a][b]$
&[a]	bitwise AND	bit-decompose([a]) and Boolean AND on the bits
|[a]	bitwise OR	bit-decompose([a]) and Boolean OR on the bits
$^\llcorner a]$	bitwise XOR	bit-decompose([a]) and Boolean XOR on the bits
[a]	bitwise compliment	bit-decompose([a]) and Boolean NOT on the bits
$[a] << [b]$	left shift	$[a]2^{[b]}$
$[a] >> [b]$	signed right shift	bit-decompose([a]) and shift by [b] places or bit-decompose both and use Boolean circuit
$[a] >>> [b]$	unsigned right shift	$[a]/(2^{[b]})$
$[a] == [b]$	equal	[DFK+06]
$[a] < [b]$	less than	see included papers

Table 1. List of operations

remembered. For example this can be handled by creating an additional array of the same size. The second array contains elements which are secret shared values which are either 1 or 0 corresponding to elements within or outside the scope of the original array.

- bit-decompose([a]) - the variable a is split into a list of l secret shared variables, where l is the maximum bit length of a.

The tables contain the following short hand explanations:

Basic operation The operation is one of the five basic operations (sharing, revealing, addition, multiplication and negation).

Internal Internal operations, such as function calls (method invocation) and the scope of variables, can be handled internally by each player. They have an essential function in the programming language as they are used to structure and simplify the program. On the other hand these

Statement (syntax)	Purpose	Multiparty computation
expression (expr)	side effects	internal
statements	group statements	internal
;	do nothing	internal
label:statement	name a statement	internal
variable	declare a variable	internal
if ([a]) [b] else [c]	conditional	[b](1-[a]); [c][a]; (see below)
switch ([a])	conditional	rewrite as if statements
while ([a])	loop	perform ordinary loops until [a] is true, then perform dummy loops until max([a])
for (init; test; update)	for statement	see comments
break	exit block	internal
continue	restart loop	internal
return	end method	internal
synchronized	critical section	internal
throw	throws exception	internal
try	handle exception	internal

Table 2. List of statements

operations do not affect the multiparty computation being performed or the messages sent between the players, and the computer program can in theory be rewritten to avoid these operations.

Dummy statements Using programming language elements such as "if," "while", and "for" can create the need for dummy statements. Dummy statements are operations that leave the secret shared variables unchanged. Dummy statements can for example be handled by introducing a secret shared "dummy flag", this variable is set to 1 if it is not a dummy statement and 0 if it is a dummy statement. For example, the statement "if ([a]) [b]++ else [c]–;" can be rewritten as "[b] = [b] + 1([a]); [c] = [c] - 1([1-a]);", where [a] functions as a dummy flag. Rewriting expressions this way avoids the need to reveal any information about [a].

See comments For those operations and statements that need a longer explanation.

The multiparty operations and variables marked in the tables with *see comments* are described further here.

Array element access, array element creation Accessing arrays with an secret shared size can be challenging, but is possible as long as the maximum number of elements $max([a])$ is known, where $[a]$ is a secret shared variable. For example see the description for $max([a])$.

Division Multiplication of two secret shared values is a basic operation. The same cannot be said for integer division of secret shared variables ($[a]/[b]$). One way of computing integer division is to "bit-decompose"

$[a]$ and $[b]$ and then compute the integer division as a Boolean circuit. A recent paper by Dahl, Ning and Toft [MDT12] presents a protocol which requires a logarithmic number of multiplications in logarithmic number of rounds.

`for` statements For statements consists of three parts called init, test, and update. If the secret shared variable is only in the init or update part of the "for" statement, then the "for" statement can be executed as an internal computation. On the other hand, if there are secret shared variables in the test part of the "for" statement, then it has implications on the multiparty computation. The "for" statement should then be run as normal until the test becomes true, thereafter only dummy "`for`" loops should be run until the maximum number of iterations is reached.

6. Software Frameworks for Multiparty Computations

This section will first give an overview of current implementations of multiparty computation. This section will also examine some of the challenges that are faced when implementing multiparty computation. The Subsection 6.2 examines the fact that the computations are computed over finite fields with secret shared information, where it is impossible to detect overflows. Subsection 6.3 examines the problem of timing. The final two subsections examine how Shamir's secret sharing can be implemented in practice and which operations that are implemented in VIFF.

6.1 Frameworks

In this thesis, these implementations of multiparty computation will be called frameworks as they provide programmers with the basic building blocks and an API (application programming interface) for constructing and executing multiparty computation programs, while at the same time, removing much of the complexity associated with multiparty computation.

It is interesting to note that although the general theory of multiparty computation was published in 1988 [GMW87, CCD88, BOGW88] no known software frameworks were developed until 2005 [BDJ+05], when the Secure Multiparty Computation Language was created. Although the FairPlay project [MNPS] was created in 2004, the framework only worked for two-party computation [MNPS04]. It was not until FairPlayMP in 2008 that the system was extended for more than two players. The field of practical applications of multiparty computations has been developing rapidly since then and there are currently six known frameworks. These are:

- The FairPlay project which started at Hebrew University of Jerusalem and the University of Haifa in Israel in 2004. Fairplay [MNPS] is a

system for secure two-party computation. This was later expanded into a multiparty computation setting with FairplayMP [BDNP08] in 2008.

- Secure Multiparty Computation Language (SMCL) [MP09] is a domain specific programming language for secure multiparty computation. It was developed as part of the SIMAP project (Secure Information Management and Processing project) in 2005.

- Sharemind [oTCa] is another project aiming to be an efficient and easily programmable platform for developing privacy-preserving computations [BLW08]. It is currently being developed at the University of Tartu and AS Cybernetica in Estonia. The first practical version was shown in 2007.

- The Virtual Ideal Functionality Framework (VIFF) [Tea09] was created at the University of Aarhus in Denmark, where Martin Geisler started the work in 2008 as an improvement to the SMCL.

- The SecureSCM project [pro] attempts to realize secure computation protocols for collaborative supply chain management. It was started sometime in 2008. The programming language they have created is called the L1 Language [SKM10].

- TASTY [oTCb] is a Tool for Automating (i.e., describing, generating, executing, benchmarking, and comparing) efficient Secure Two-partY computation protocols using combinations of garbled circuits and homomorphic encryption techniques. The program was developed in 2010 at the System Security Lab at Ruhr-Univeristy Bochum and a description of the framework can be found in [HKS+10].

- The JavaMPC framework created by the author of this thesis. The first working version was developed during the Autumn of 2009. This builds on the ideas from VIFF, while trying to create a more flexible environment for programmers on the Java platform. Implementation was stopped after two iterations of the program where developed early in 2010.

The different frameworks have very different design structures. Some of the different aspects have been included in Table 3. The table does not give a comprehensive overview of the different frameworks, but it describes some of the most important differences. Special compiler refers to the fact that the multiparty computations are written in a different programming language than the what the underlying framework is written in. The framework thus complies the multiparty computations before the computations can be computed.

Name	Written in	Network layer	Special complier	Players	Notes
Fairplay	Java	Java sockets	Yes(SDFL)	2	Circuit based
FairplayMP	Java	Java sockets	Yes(SDFL2.1)	3+	Circuit based
SMCL	C++	Unknown	Yes(SMCL)	3+	Used for auction
Sharemind	C++	RakNet	Yes/No	3(only)	Shared database
VIFF	Python	Twisted	No	2,3+	Open source
SecureSCM	Java	Unknown	Yes(L1)	2+	[SKM10]
Tasty	Pyton	Unknown	Yes	2	Circuit based
JavaMPC	Java	Netty/Mina	No	3+	Under development

Table 3. Comparison of multiparty computation frameworks

6.2 Simulated Integer Arithmetics

For security reasons most secret sharing schemes are computed over finite fields. All current software frameworks work with each share representing one value over a finite field $\mathbb{Z}_p$, where p is a large prime. On the other hand, the problem setting will normally be with computations over the integers $\mathbb{Z}$. This cognitive gap can be bridged by letting the computations be over *simulated integer arithmetics*, where the players simulate computations over the infinite field $\mathbb{Z}$ by using computations over finite fields $\mathbb{Z}_p$. This is analogous to the arithmetic in modern computers, but this cognitive gap might lead to situations where there is an overflow. An overflow is a situation where a value becomes larger than the string of bits that represents and stores the value. For example, a single byte can only store values between 0 and 255. Therefore if we try to add 254 and 3 the result will be 1 when stored in a single byte. Overflow was a significant problem in earlier computers as they had short register lengths. Modern computers, on the other hand, use larger register sizes so overflow is no longer such a big problem. Moreover, the arithmetic software will detect and report overflow situations.

The reason for emphasizing overflow in multiparty computations is that it is not possible to add overflow warnings in multiparty computation. In addition, the input values are secret and each operation is usually costly. The prime p will therefore have to be chosen in such a way that the finite field $\mathbb{Z}_p$ has to be larger than all possible inputs, intermediate values, and final values that might arise from all possible inputs. This could lead to a very large value for p, which in turn affects efficiency because more bits will have to be transmitted for each share. It is also important to verify that the inputs are within the given bounds otherwise some player might supply inputs that will result in future overflow situations.

6.3 Multiparty Coordination

The five basic operations are not particularly difficult to implement in a framework once we identify an efficient method to do this, nor is it difficult to

set up communication channels between the players (computers). The challenge comes from the requirement that programmers should be able to program essentially any function by using the framework. This means that the framework has to be able to interpret essentially any algorithm, and turn that algorithm into a set of basic multiparty computation operations and coordinate the computation of these basic operations among the players as efficient as possible. Algorithms for ordinary functions may produce programs with ten to a hundred thousand operations. A programming error may result in a wrong answer rather than an error report. The communicated messages will either have to be sent in a well-defined sequence, or each operation will have to be labeled. The reason for this will be shown in Algorithm 1:

Algorithm 1 Possible timing problem

```
    Share a, b, c, d
    x = ab
    y = cd
    Reveal x
5:  Reveal y
```

If the messages are not labeled, or the messages are not sent and received in the correct sequence, then the four secret shared variables might be shared in the wrong order, the multiplications might be done in the wrong order, or the values might be revealed in the wrong order. If a message or operation labeling is not done, then all operations have to be carried out in a strict order across the computer system, and a computer will likely spend a significant amount of time waiting for input from the other computers.

Labeling, on the other hand, increases the efficiency because the players can work asynchronously, by continuing the computation as soon as as the necessary input is available. On the other hand, each message will have to be labeled uniquely. Creating an algorithm that enforces unique labels on all messages is a challenge in and of itself, and is a topic that is not discussed in the multiparty computation literature.

6.4 Basic Operations

This subsection will focus on the details of a multiparty computation framework implementing Shamir's secret sharing scheme (see Section 3.4). There are a myriad of ways that such a program can be implemented. So this subsection will focus on all the common factors that all frameworks possess.

At the basic level a framework has to handle two basic types of variables, which are "values" and "secret shared values", also called "shares". The "values" are often expressed as integers over some finite field, as most secret sharing schemes are computed over a finite field. The "shares" are values representing a secret value shared among the participants. Each participant will at some

point have a representation of the secret shared value and this representation will also take mostly the form of an integer over a finite field.

Section 5 listed five basic types of operations that have to be handled by the framework, these are sharing, revealing, addition, multiplication and negation. Tables 4 and 5 summarize these operations, where n is the number of participants in the multiparty computations and communication is the amount of messages that have to be sent.

Operation	Input variable	Output variable	Communication
Sharing	value	share	n
Revealing to one	share	value	n
Revealing to all	share	value	n^2
Negating	value	value	local
Negating	share	share	local

Table 4. Unary operations in a secret sharing framework

Operation	First input	Second input	Output	Communication
Addition	value	value	value	local
Addition	share	value	share	local
Addition	share	share	share	local
Multiplication	value	value	value	local
Multiplication	share	value	share	local
Multiplication	share	share	share	n^2

Table 5. Binary operations in a secret sharing framework

6.5 Additional Operations

The five basic operations are necessary to implement in a framework, but the framework needs more functionality to be useful in practice. Additional operations are needed. The following list of operations can be identified in the VIFF framework.

7. Summary of Papers

This section gives a summary of the papers included in part two of this thesis. The summary consists of an introduction to the papers and an overview of how the papers relate to each other. Then each paper is summarized separately with a short description, some remarks about the results, and a description of my contribution to each paper.

Operation	Type	Description
Function calls	flow	$[z] = f([x], [y], \ldots)$
For loops	flow	$for(0 \text{ to } n)\{\ldots\}$
If statements	flow	$[z]?[x] : [y] \equiv [x(1-z) + yz]$
While loops	flow	$while(x)\{\ldots\}$
(Pseudo-)random secret shared value	value	$[z] = Rand()$
Broadcast	value	$x = \text{Broadcast}(x, player)$
Equality	operator	$[x] = [y]$
Comparison	operator	$[x] < [y]$
Splitting value into bits	operator	$[z]_B = \text{Split}([z]), [z] = \sum [z]_i 2^i$
Fan-in multiplication	operator	$\prod [z_i]$
Boolean logic	operators	and, or, not, etc.

Table 6. Additional useful operations for a multiparty computation software framework

7.1 The Starting Point

My initial work involved examining security mechanisms for sensor and ad-hoc networks. This work did not result in any papers except from the work mentioned in paper **A**. Over time my focus shifted to multiparty computation which has been a central component in all of the included papers.

The work on sensor and ad-hoc networks was beneficial and has guided my later work on multiparty computation, as both multiparty and ad-hoc networks are decentralized systems. The advantage of decentralized systems is that there is no single point of failure nor any single node that all players must trust. The disadvantage is increased complexity and the use of communication. The decentralized nature of these systems also make them vulnerable to malicious players, which can ruin the system for everyone else if any players collude.

The difference between the two fields of study is that in ad-hoc networks we normally deal with physical entities that have well-defined positions and we have physical limitations on how they communicate with each other, but in multiparty computation these restrictions are not present. In multiparty computations, however, there are restrictions on how information can be communicated between the players.

7.2 The Thesis Papers

This thesis includes six published papers and a short unpublished paper, ranging from my first publication to NIK (Norsk Informatik Konferanse) in 2006 to my final publication in ICISC (International Conference on Information, Security and Cryptography) in 2009. The papers are labeled from **A** to **F** chronologically according to when they were published. All six papers discuss multiparty computation, and explore several different aspects of the field. One group of papers focus on improving algorithms for multiparty com-

putations, while another group focus on practical applications of multiparty computations.

Paper **B**, **C** and **F** are concerned with theoretical improvements to multiparty computation algorithms, in particular the "less-than" and "bit-decomposition" operation. Although the focus of these papers is narrow, these operations are among the most useful operations. Therefore any such improvements in operations could have vast implications for the efficiency of multiparty computation in practice. Paper **B** and **C** focus on the "less-than" operator. Paper **C** is a direct improvement of the proposal of paper **B**. Paper **F** focuses on the "bit-decomposition" operation, where one secret shared value is decomposed into a vector of secret shared values. This operation is important because there is no simple algorithm yet found which extracts individual bits from a secret shared value, except by using a "bit-decomposition" algorithm.

Paper **A** and **E** are oriented toward practical applications and examine possible uses for multiparty computation. Paper **A** considers an ad-hoc network, where some nodes can calculate their position based on the private location information in other nodes, while maintaining the privacy such that no node needs to reveal its position to the others. Paper **E** proposes an e-voting scheme with distributed trust and multiparty computation for counting votes in an election.

Paper **D** investigates how a distributed RSA key generation algorithm can be carried out by multiparty computation, and implements it using the software framework VIFF. The result is a program where three players can generate a RSA key pair without any of them ever knowing the secret key. The algorithms were tested for efficiency and further improvements were proposed.

7.3 Paper A

This paper considers a network of nodes, where a majority of nodes know their current geographical position and want to keep that information private. The nodes that do not know their geographical position can compute their position based on the distance between nodes and the geographical position of other nodes. It is assumed that the distance between nodes can be computed by using some method such as delay timing or signal strength. The paper shows how this can be accomplished using multiparty computations for three nodes in the one-dimensional case and four nodes in the two-dimensional case. The calculations can also be extended to multiple nodes and to the three-dimensional case. I was the sole author on this paper and presented it at the NIK conference in November 2006.

7.4 Paper B

This paper considers the comparison operator $a < b$, where a and b are secret shared values. Improving the "less-than" operator could enable a significant

increase in the efficiency when evaluating functions using multiparty computation. This paper improves the "less-than" operator by roughly a factor of two or three in comparison to previous papers in the literature. The algorithm can work for any secret sharing scheme because the algorithm only assumes that there are primitives for addition and multiplication of secret shared values in addition to some method of generating random secret shared values. The paper also explores the possibility of moving some of the computations to a pre-computing step. Pre-computing is where the players can do multiparty computations, but do not have the available data to compute on. This paper was a joint effort between Tomas Toft and myself. The idea was based on discussions between the authors, and both authors contributed equally to the writing process. The work was presented at the ICITS conference in 2007.

7.5 Paper C

This paper considers the "less-than" operator $a < b$, where a and b are secret shared values. This paper improves upon the comparison operation by roughly a factor of two or three in comparison to paper **B**. This algorithm can also work for any secret sharing scheme because it too only assumes that there are primitives for addition and multiplication of secret shared values in addition to some method of generating random secret shared values. I was the sole author of this paper and I came up with the idea and wrote the paper myself. I received feedback and comments from the people listed in the acknowledgment section. This paper was presented at the SECRYPT conference in 2009.

7.6 Paper D

This paper was based on results from a Master's thesis topic I proposed and co-supervised. The paper presents a distributed RSA key generation protocol for three players, and the multiparty computation is implemented using the VIFF framework. The key generation protocol is based on an algorithm proposed by Boneh and Franklin [BF97]. We compared our results of efficiency and security with earlier work on the same algorithm. Our paper proposes further improvements. I contributed the main ideas of the paper, and co-supervised Atle Mauland in his programming and writing of his Master's thesis, and I took an active role in authoring the joint paper. This paper was presented at the NISK conference in 2009.

7.7 Paper E

This paper proposes an e-voting system which ensures a high degree of privacy and anonymity of votes in any given election where the system is used. The tallying process is split between multiple parties computing the final tally using multiparty computations. The goal of this paper was to propose a system that ensures a distribution of roles and responsibilities, in such a way

that no single role or tally could cheat or corrupt the voting process without being detected. This paper began as a result of examining the 2011 E-voting project in Norway. This paper was a joint effort between Md. Abdul Based and myself, where we both contributed equally with ideas and writing of the paper.

7.8 Paper F

This paper considers bit-decomposition of a value a, where a is a secret shared value. The value a should be split into secret shared bits of information such that $[a] = \sum_{i=0}^{n} [a_i]$, where $a_i \in \{0, 1\}$. This paper presents an algorithm which is linear $\mathcal{O}(\ell)$ in the number of multiplications. This is an improvement upon earlier constant round bit-decomposition algorithms, which are nearly linear $\mathcal{O}\ell log(\ell)$ in the number of multiplications. My main contribution to this collaboration was to come up with the idea which made this algorithm possible. This idea is similar to the idea for the algorithm proposed in paper **C**. I wrote an early draft version of this paper, but most of the writing was done by Tomas Toft. This paper was presented at the ICISC conference in 2009.

Bibliography

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *In Advances in Cryptology - Proceedings of CRYPTO 2002*, pages 417–432. Springer-Verlag, 2002.

[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. `http://eprint.iacr.org/`.

[BDJ+05] Peter Bogetoft, Ivan B. Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications. Technical Report RS-05-18, June 2005.

[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.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology – CRYPTO 97*, pages 425–439. Springer-Verlag, 1997.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS '08: Proceedings of the 13th European Symposium on Research in Computer Security*, pages 192–206, Berlin, Heidelberg, 2008. Springer-Verlag.

[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*, pages 136–147, 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.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005.

[CGH00] Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In *In Advances in Cryptology EUROCRYPT 2000*, pages 190–206. Springer-Verlag, 2000.

[Cyb] Cybernetica. Sharemind deployment and performance whitepaper. `http://sharemind.cyber.ee/files/whitepapers/sharemind-deployment-and-performance-whitepaper.pdf`.

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 120–127, London, UK, 1988. Springer-Verlag.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136, London, UK, 2001. Springer-Verlag.

[DT06] Ivan Damgard and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. Cryptology ePrint Archive, Report 2006/044, 2006. `http://eprint.iacr.org/`.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.

[Fla05] David Flanagan. *Java In A Nutshell, 5th Edition*. O'Reilly Media, Inc., 2005.

[FP09] Oriol Farras and Carles Padro. Ideal hierarchical secret sharing schemes. Cryptology ePrint Archive, Report 2009/141, 2009. `http://eprint.iacr.org/`.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178. ACM, 2009.

[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.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography.

In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, 1998.

[HKS+10] Wilko Henecka, Stefan Kgl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: Tool for automating secure two-party computations. Cryptology ePrint Archive, Report 2010/365, 2010. `http://eprint.iacr.org/`.

[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.

[KN06] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 2006.

[MDT12] Chao Ning Morten Dahl and Tomas Toft. On secure two-party integer division. In *Financial Cryptography and Data Security 2012*, 2012. To be published.

[MNPS] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. The fairplay project. `http://www.cs.huji.ac.il/project/Fairplay/home.html`.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.

[MP09] SIMAP Secure Information Management and Processing. Secure multiparty computation language, 2009. `http://www.brics.dk/SMCL/`.

[MWB99] M. Malkin, T. Wu, and D. Boneh. Experimenting with Shared Generation of RSA keys. In *In Proceedings of Symposium on Network and Distributed System Security (SNDSS*, 1999.

[oTCa] University of Tartu and AS Cybernetica. `http://sharemind.cs.ut.ee/`.

[oTCb] University of Tartu and AS Cybernetica. `http://code.google.com/p/tastyproject/`.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *In Advances in Cryptology EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[pro] SecureSCM project. `http://www.securescm.org/`.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SKM10] Axel Schroepfer, Florian Kerschbaum, and Guenter Mueller. L1 - an intermediate language for mixed-protocol secure computation. Cryptology ePrint Archive, Report 2010/578, 2010. `http://eprint.iacr.org/`.

[Tea09] VIFF Developement Team. Viff, the virtual ideal functionality framework, 2009. `http://viff.dk/`.

[Tho09] Rune Thorbek. *Linear Integer Secret Sharing*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2009.

[Tof07] Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2007.

[Tof09] Tomas Toft. Solving linear programs using multiparty computation. In *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 90–107. Springer Berlin / Heidelberg, 2009.

[vDGHV09] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009. `http://eprint.iacr.org/`.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213, New York, NY, USA, 1979. ACM.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

II

INCLUDED PAPERS

Paper A

Multi-party Secure Position Determination

Tord Ingolf Reistad

At Norsk informatikkonferanse 2006 (NIK 06)
http://www.himolde.no/nik06/articles/12-Reistad.pdf

Molde, Norway, November 20-22, 2006

MULTI-PARTY SECURE POSITION DETERMINATION

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Abstract We consider the problem of calculating the geographical position of nodes in a wireless network, where the privacy of location data is highly valued. A solution is presented using a multi-party computation, where the secret inputs are the position of anchor nodes and distances between nodes.

1. Introduction

Mobile ad-hoc networks (MANET) are wireless mobile devices (nodes) that cooperatively form a network without central infrastructure. Each node cooperates by being involved in routing and forwarding information between neighbors. Thus, an ad-hoc network allows devices to create a network without prior coordination or configuration.

In a non-homogeneous networks some nodes know their geographical position, and distance between nodes can be estimated. This information can be used to calculate the position of additional nodes in the network. The position data thus generated can in turn be used for a wide variety of applications, such as route tracking, location based services and many more.

On the other hand privacy of location data might be very important. This because ad-hoc networks might be implemented in future generations of mobile phones. Ad-hoc network can also be integrated into other mobile devices such as PDA's, digital cameras and laptops that are worn or carried around.

The location data can be kept private with the use of multi-party calculations. Multi-party calculations are distributed calculations done by multiple parties. Each value that should be private is split up into shares, such that an single share does not give any information about the value itself. Calculations can be performed on these shares, and the answer can be revealed by recombining shares representing the answer.

Our contribution is to show how multi-party computations can be performed on location data, to improve privacy in ad-hoc networks. This is part of

ongoing research, therefore only the protocols for simple location calculations in one and two dimensions are given.

1.1 Related work

Multi-party computations were introduced by Yao [1] and fundamental results obtained by Ben-Or, Goldwasser, and Wigderson [2], as well as Chaum, Crepau, and Damgård [3]. An integral part of any multi-party computation are secret sharing schemes. We will use Shamir's secret sharing scheme which was introduced in [4]. Damgård et al. [5] has shown novel techniques for calculating comparison and bit extraction.

The paper is organized as follows. In the following section, we describe the model, in section 3 we fix the notation for the article. We then recall the basics of a multi-party computation in section 4, present our solution in section 5, and discuss conclusion and future work in section 6.

2. Model

Consider an ad-hoc network of communicating nodes some of which know their geographical position, henceforth called *anchor nodes*. While others would like to compute their own geographical position and will be referred to as *floater nodes*. Both sets of nodes would like their geographical position to remain secret.

We present a protocol that uses multi-party computations to provide the floater nodes with their geographical position, while ensuring the privacy of location data of all honest participants. We assume that the communicating parties can determine the distance between them. This can for example be done by examining the loss of signal strength or by calculating the time the signal takes to propagate. We also assume there exists private channels to each node e.g. each node has a public RSA key, this to ensure that shares will only be known to the correct recipients.

In the protocols we describe, we make the assumption that the positions of nodes are in a two dimensional plane. This is a good approximation for long range ad-hoc networks, while greatly simplifying the computations and reducing the communication complexity.

We also assume that there are sets of $k \geq 3$ anchor nodes that are in direct communication range with at least some floater nodes in the network, and the anchor nodes are willing to participate in the computation. When these floater nodes find their geographical position they can in turn act as anchor nodes to other floater nodes.

3. Notation

All computations on secret shares will be performed over a finite field $\mathbb{F}$. This finite field is assumed to be chosen large enough such that no overflow occurs within the field - i.e. all computation on the inputs is equiv-

alent to computation over integers. The nodes will be denoted by capital letters A, B, C, and P, their position will be denoted by the coordinates $(x_A, y_A), (x_B, y_B), (x_C, y_C)$, and (x_P, y_P), respectively. The distance between nodes A and B will be denoted by $|AB|$.

The shares for a secret s will be denoted by $[s]_i$ where i identifies the recipient of the share.

4. Multi-party computation

Suppose m players $P_1, \ldots, P_m$ owning secret inputs $x_1, \ldots, x_m \in \mathbb{F}$, respectively, would like to compute a function $y = f(x_1, \ldots, x_m)$ without revealing more about $x_1, \ldots, x_m$ than what can be inferred from the output y.

They achieve this by first distributing *shares* of their input values to each other by using, for instance, Shamir's secret sharing scheme. Then they perform all operations given by the function f on the distributed shares and finally recombine the shares to obtain the output y.

Let a, b be secrets with shares $[a]_i$, $[b]_i$, $1 \leq i \leq m$, and let $c \in \mathbb{F}$. Since the function f can be written as a rational function in $x_1, \ldots, x_m$, only the following operations need to be carried out.

- Linear Combination

  Shamir's secret sharing scheme is linear, that is, shares for linear combinations of secrets are equal to the corresponding linear combination of shares and can be computed by the participants without any interaction. Thus, $[a + cb]_i = [a]_i + c[b]_i$.

- Multiplication

  The multiplication of two secrets a and b can be done by the following interactive protocol, described in [6].

  Each player i computes the value $h_i = [a]_i[b]_i$, splits h_i into shares $[h_i]_j$ and distributes the shares $[h_i]_j$.

  Each player can then compute a share of the product ab by using the following equation.

  $$[ab]_i = \sum_{j=1}^{m} \lambda_j [h_j]_i \tag{1}$$

  where λ_j is the first row of the inverse of the Van der Monde matrix $[i^j]_{1 \leq i \leq m, 0 \leq j \leq m-1}$.

- Multiplicative Inverse

  We use the protocol given in [7]. To compute $1/b$ the players first create a random number R as follows. Each player i distributes shares $[r_i]_j$ of a random number r_i and adds up all received shares to obtain $[R]_i = \sum_{k=1}^{m} [r_k]_i$, so that $R = \sum_{i=1}^{m} [R]_i$.

The shares for the value bR are calculated using the multiplication protocol and then bR is revealed. The shares for the inverse of b are then calculated as

$$\left[b^{-1}\right]_i = (bR)^{-1}[R]_i. \tag{2}$$

5. Position calculations

To simplify the exposition, we begin by solving the one dimensional analogue of our problem.

5.1 One-dimensional calculations

If a node P wants to know its position in a one-dimensional world it only needs to contact two anchor nodes A and B. In what follows, P needs to know the distances $|AP|$ and $|BP|$, while A and B only need to know their positions x_A and x_B, respectively. A, B, and P can then carry out the following protocol which allows P to learn its position x_P. All secrets will be shared using a $(2, 3)$–threshold linear secret sharing scheme.

A and B distribute shares of their positions x_A and x_B, P distributes shares of $|AP|$ and $|BP|$.

Shares for the following values are then calculated by each party.

$$\begin{aligned} x_{A1} &= x_A + |AP| & \qquad x_{B1} &= x_B + |BP| \\ x_{A2} &= x_A - |AP| & \qquad x_{B2} &= x_B - |BP| \end{aligned}$$

e.g. the shares $[x_{A1}]_i$ are calculated from the sum $[x_A]_i + [|AP|]_i$, for all nodes $i \in A, B, P$

Thereafter A, B and P calculate the following function and the resulting shares are sent to P which will enable P to learn its position.

$$x_P = \frac{x_{A1}x_{A2} - x_{B1}x_{B2}}{(x_{A1} + x_{A2}) - (x_{B1} + x_{B2})} \tag{3}$$

The division can be done in the field only if x_P is known to be an integer value. Otherwise division in the field will not give the same answer as division in over integers. A solution to integer division is to use the methods in [5] to get the bits and do integer division over the bits.

5.2 The 2-dimensional case

The two dimensional case involves at least four parties, namely three anchor nodes A, B, C, and one floater node P.

The three circles described by the following equations intersect, by construction, in the point (x_P, y_P).

$$\begin{aligned}(x - x_A)^2 + (y - y_A)^2 &= |AP|^2 \\ (x - x_B)^2 + (y - y_B)^2 &= |BP|^2 \\ (x - x_C)^2 + (y - y_C)^2 &= |CP|^2\end{aligned}$$

From each pair of circles we can construct a line passing through their intersection. The equations for these lines can be written as

$$\begin{aligned}2(x_A - x_B)x_P + 2(y_A - y_B)y_P &= x_A^2 - x_B^2 + y_A^2 - y_B^2 + BP^2 - AP^2 \\ 2(x_A - x_C)x_P + 2(y_A - y_C)y_P &= x_A^2 - x_C^2 + y_A^2 - y_C^2 + CP^2 - AP^2 \\ 2(x_B - x_C)x_P + 2(y_B - y_C)y_P &= x_B^2 - x_C^2 + y_B^2 - y_C^2 + CP^2 - BP^2\end{aligned}$$

If no two nodes are close by each other, it suffices to solve any pair of the above equations for (x_P, y_P). Thus the above equations can be simplified to the following linear system.

$$\begin{aligned}a_1 x_P + b_1 y_P &= c_1 \\ a_2 x_P + b_2 y_P &= c_2\end{aligned}$$

The solution of the linear system is then obtained by the following functions

$$x_P = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - b_1 a_2} \qquad y_P = -\frac{c_1 a_2 - c_2 a_1}{a_1 b_2 - b_1 a_2}$$

Again the division can generally not be done over the field, so the shares for the values $(c_1 b_2 - c_2 b_1)$, $(c_1 a_2 - c_2 a_1)$ and $(a_1 b_2 - b_1 a_2)$ are sent to P. P recombines the shares to get the values and does the divisions over integers to obtain its position (x_P, y_P).

In practice finding the geographical position of P in 2 dimensions can be done with 3 rounds of communication. In the first round all the variables are distributed. The second round the multiplication is done in parallel. The third round consists of sending the shares for the answers to P. With 4 nodes the privacy is preserved as long as 2 nodes do not cooperate.

This idea can easily be generalized to higher dimensions.

6. Conclusion and future work

In this paper we have given an introduction into multi-party computation over Shamir's secret sharing scheme. We have also shown how these methods

can be used to compute geographical position using simple algorithms for calculating location in one and two dimensions. The equations are quite simple and do not perform so well when anchor points are close together or in a line, but they improve privacy.

To improve the accuracy of the calculations and to compute geographical position in ad-hoc networks which have fewer anchor nodes, one can use better algorithms. A great deal of work has been done on localization, e.g. Strang et al. [8]. Localization in wireless ad-hoc and sensor networks can be found in e.g. [9] and [10]. But multi-party computations require many rounds of communication, e.g. calculating if $a > b$ costs 114 rounds of communication as proposed by Damgård et al. [5]. Therefore further research is needed to get more round efficient computations, and find algorithms that are well suited to multi-party computations.

Homomorphic public-key systems and threshold homomorphic public-key systems could be explored as an alternative to multi-party computations. A overview of such systems is given in [11].

Also for larger networks a subset of nodes could together be used as a form of distributed trusted third party. This could reduce the computational burden for the overall system.

6.1 Acknowledgments

We would like to thank Sasa Radomirovic for many helpful comments throughout the writing process.

Bibliography

[1] A. Yao. Protocols for secure computation. In IEEE, editor, *23rd annual Symposium on Foundations of Computer Science, November 3–5, 1982, Chicago, IL*, pages 160–164. IEEE Computer Society Press.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the twentieth annual ACM Symposium on Theory of Computing*, 1988 ACM Press., pages 1–10.

[3] D. Chaum, C. Crepeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM Symposium on Theory of Computing*, 1988 ACM Press., pages 11–19.

[4] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[5] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. *Proceedings of the third Theory of Cryptography Conference TCC 2006*, pages 285–304, 2006.

[6] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified VSS and fast-track Multiparty Computations with applications to Threshold Cryptography, 1998.

[7] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In ACM, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing: Edmonton, Alberta, Canada, August 14–16, 1989*, pages 201–209, ACM Press.

[8] Gilbert Strang and Kai Borre. *Linear algebra, geodesy, and GPS*. Wellesley-Cambridge Press, Wellesley, MA, USA, 1997.

[9] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks (Amsterdam, Netherlands: 1999)*, 43(4):499–518, November 2003.

[10] A. Savvides, H. Park, and M. Srivastava. The n-hop multilateration primitive for node localization problems. *Mobile Networks and Applications*, 8(4):443 – 451, aug 2003.

[11] Kristian Gjøsten. Homomorphic public-key systems based on subgroup membership problems. *Proceedings of MyCrypt 05 volume 3715 of LNCS*, pages 314–327, 2005.

Paper B

Secret Sharing Comparison by Transformation and Rotation

Tord Ingolf Reistad and Tomas Toft

In Preproceedings, International Conference on Information Theoretic Security 2007 (ICITS 07)

Madrid, Spain, May 25-28, 2007

SECRET SHARING COMPARISON BY TRANSFORMATION AND ROTATION

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Tomas Toft
University of Aarhus, Dept. of Computer Science,
DK-8200 Aarhus N, Denmark.[*]
tomas@daimi.au.dk

Abstract Given any linear secret sharing scheme with a multiplication protocol, we show that a given set of players holding shares of two values $a, b \in \mathbb{Z}_p$ for some prime p, it is possible to compute a sharing of ρ such that $\rho = (a < b)$ with only eight rounds and $29\ell + 36\log_2(\ell)$ invocations of the multiplication protocol, where $\ell = \log(p)$. The protocol is unconditionally secure against active/adaptive adversaries when the underlying secret sharing scheme has these properties. The proposed protocol is an improvement in the sense that it requires fewer rounds and less invocations of the multiplication protocol than previous solutions.

Further, most of the work required is independent of a and b and may be performed in advance in a pre-processing phase before the inputs become available, this is important for practical implementations of multiparty computations, where one can have a set-up phase. Ignoring pre-processing in the analysis, only two rounds and 4ℓ invocations of the multiplication protocol are required.

1. Introduction

In multiparty computation (MPC) a number of parties $P_1, \ldots, P_n$ have *private* inputs for some function that they wish to evaluate. However, they are mutually mistrusting and do not wish to share their inputs with anyone. A great deal of work has been done on unconditionally secure constant round MPC with honest majority including but not limited to [1, 9, 4, 10], indeed it has been demonstrated that any function can be computed securely using circuit based protocols.

[*]Supported by Simap

However, when considering concrete applications, it is often more efficient to focus on secure arithmetic e.g. in the field $\mathbb{Z}_p$ for some odd prime p, which may then be used to simulate integer arithmetic. Unfortunately, many such applications require non-arithmetic operations as well; this provides motivation for constructing specialized, efficient, constant-rounds protocols for primitive tasks.

Access to the binary representation of values allows many operations to be performed relatively cheaply. Although constant-round bit-decomposition is possible as demonstrated by Damgård et al. [5], it is less efficient than desired. Primitives may be used repeatedly and any improvement of these lead to improvements in the overall applications. This work focuses on one such primitive, namely comparison (less-than-testing) of secret shared values, i.e. obtaining a sharing of a bit stating whether one value is less than another without leaking any information.

Related work and contribution. Much research has focused on secure comparison in various settings as it is a quite useful primitive, though focus is often on some concrete application e.g. auctions. Often the setting differs from the present [2, 7, 13]. Through bit-decomposition, Damgård et al. provided the first constant rounds comparison in the present setting [5] – this required $\mathcal{O}(\ell \log(\ell))$ secure multiplications where $\ell = \log(p)$. Comparison was later improved by Nishide and Ohta [11] who reduced the complexity to $\mathcal{O}(\ell)$ multiplications.

A common streak in all of the above solutions is that the binary representation of the values is considered. Thus, unless a radically different approach is taken, improving on the $\mathcal{O}(\ell)$ bound does not seem feasible. The present work builds on sub-protocols and ideas of [5] and [11], but also uses ideas from [6] which considers comparison of bitwise stored values based on homomorphic encryption in a two-party setting.

Based on these papers, we construct a protocol with the aim of reducing the constants hidden under big-$\mathcal{O}$. In particular, when the protocol is split into pre-processing (secure computation independent of the inputs) and online computation, the online round complexity is extremely low; this split has not been considered in the related literature.

Table 1 compares the present solution to those of Damgård et al. and Nishide and Ohta. Type A refers to comparison of arbitrary values $[a], [b] \in \mathbb{Z}_p$, while R denotes values of restricted size, $[a], [b] < \lfloor \frac{p}{4} \rfloor$. When using $\mathbb{Z}_p$ to simulate integer computation, it is not unreasonable to choose p a few bits larger to accommodate this assumption.

With regard to the restricted comparison, in contrast to earlier papers (and the general comparison of Table 1), it is assumed that two rather than four attempts are needed to generate random bitwise shared values. When generating such values, the probability of failing may be as high as $1/2$ implying that multiple attempts may be needed. Earlier work used a Chernoff bound

Table 1. Complexities of comparison protocols

Presented in	Type	Rounds		Multiplications	
		overall	online	overall	online
[5]	A	44	37	$184\ell \log_2(\ell) + 209\ell$	$21\ell + 56\ell \log_2(\ell)$
[11]	A	15	8	$279\ell + 5$	$15\ell + 5$
This paper	A	10	4	$153\ell + 432 \log_2(\ell) + 24$	$18\ell + 8$
[5]	R	44	37	$171\ell + 184\ell \log_2(\ell)$	$21\ell + 56\ell \log_2(\ell)$
[11]	R	13	6	$55\ell + 5$	$5\ell + 1$
This paper	R	11	2	$24\ell + 26 \log_2(\ell) + 4$	$4\ell + 1$
This paper	R	8	2	$27\ell + 36log_2(\ell) + 5$	$4\ell + 1$

to bound the number of required attempts by a factor of four, ensuring that overall failure was negligible in the number of generated values.

In this paper we assume only a factor of two *plus* κ attempts, where κ is a security parameter much smaller than the number of values to be generated. It can be shown that in this case, the probability of failure (encountering too many failures) is negligible in κ. An alternative way of viewing the issue is that the random element generation is rerun. This does not compromise security, however, we only obtain an *expected* constant-rounds solution (with double round complexity), however, this only affects pre-processing. The issue only occurs when p may be arbitrary; if it is chosen sensibly, e.g. as a Mersenne prime, the failure probability may be reduced to negligible in the bit-length of p implying even greater efficiency.

The structure of this article. Sections 3 and 4 introduce the setting as well as a number of primitives required. Most of these are well-known, and are included in order to provide detailed analysis of our protocol. Section 5 takes a high-level view of the computation required; it reduces the problem of comparison to a more manageable form similar to previous work. Section 5 introduces the DGK comparison protocol, while Sect. 4 shows how the DGK algorithm can be used to create random bitwise shared elements. Sections 7, 8 and 9 modifies DGK to avoid leaking information. Finally Sect. 10 gives an overall analysis and conclusion.

Acknowledgments. The authors would like to thank Jesper Buus Nielsen for discussions as well as comments on the article. Further, the anonymous reviewers are thanked for their helpful comments, and Prof. Ivan Damgård and Prof. Stig Frode Mjølsnes for their support.

2. Preliminaries

We assume a linear secret sharing scheme with a multiplication protocol allowing values of the prime field $\mathbb{Z}_p$, $\ell = \lceil \log(p) \rceil$, to be shared among n parties.

As as example, consider Shamir's scheme along with the protocols of Ben-Or et al. (or the improved protocols of Gennaro et al.) [12, 3, 8]. The properties of the scheme are inherited, i.e. if this is unconditionally secure against active/adaptive adversaries then so are the protocols proposed. In addition to sharing values and performing secure arithmetic of $\mathbb{Z}_p$, the parties may reveal (reconstruct) shared values, doing this ensures that the value becomes known by *all* parties.

We use $[a]$ to denote a secret sharing of $a \in \mathbb{Z}_p$. Secure computation is written using an infix notation. For shared values $[a]$ and $[b]$, and constant $c \in \mathbb{Z}_p$, computation of sums will be written as $[a] + c$ and $[a] + [b]$, while products will be written $c[a]$ and $[a][b]$. The initial three follow by the linearity of the scheme, while the fourth represents an invocation of the multiplication protocol.

Sharings of bits, $[b] \in \{0, 1\} \subset \mathbb{Z}_p$ will also be considered. Boolean arithmetic is written using infix notation, though it must be realized using field arithmetic. Notably xor of two bits is constructed as $[b_1] \oplus [b_2] = [b_1] + [b_2] - 2[b_1][b_2]$ which is equivalent.

Values may also be *bitwise shared*, written $[a]_B$. Rather than having a sharing of a value itself, sharings of the bits of the binary representation of a are given, i.e. $[a_0], \ldots, [a_{\ell-1}] \in \{0, 1\}$ such that

$$[a] = \sum_{i=0}^{\ell-1} 2^i [a_i]$$

for $\ell = \lceil \log(p) \rceil$, with the sum being viewed as occurring over the integers. Note that $[a]$ is easily obtained from $[a]_B$ by the linearity of the scheme.

When considering complexity, the focus will be on communication. Computation will be disregarded in the sense that polynomial time suffices. Similar to other work, focus will be placed on the number of invocations of the multiplication protocol as this is considered the most costly of the primitives. Addition and multiplication by constants require no interaction and is considered costless. The complexity of sharing and revealing is seen as negligible compared to that of multiplication and is ignored.

It is assumed that invocations of the multiplication protocol parallelize arbitrarily – multiplications are executed in parallel when possible. Round complexity is formally rounds-of-multiplications; rounds for reconstruction are disregarded as in other work.

3. Simple Primitives

This section introduces a number of simple primitives required below. Most of these sub-protocols are given in [5] but are are repeated here in order to provide a detailed analysis as well as for completeness. Most of these are related to the generation of random values unknown to all parties. It is important

to note that these may fail, however, this does not compromise the privacy of the inputs – failure simply refers to the inability to generate a proper random value (which is detected). Generally the probability of failure will be of the order $1/p$, which for simplicity will be considered negligible, see [5] for further discussion.

Random element generation. A sharing of a uniformly random, unknown value $[r]$ may be generated by letting all parties share a uniformly random value. The sum of these is uniformly random and unknown to all, even in the face of an active adversary. The complexity of this is assumed to be equivalent to an invocation of the multiplication protocol.

Random non-zero values. A uniformly random, non-zero value may be obtained by generating two random values, $[r]$ and $[s]$ and revealing the product. If $rs = 0$ the protocol fails, however if not, $[r]$ is guaranteed non-zero and unknown as it is masked by $[s]$. The complexity is three multiplications in two rounds.

Random bits. The parties may create an uniformly random bit $[b] \in \{0, 1\}$. As described in [5] the parties may generate a random value $[r]$ and reveal its square, r^2. If this is 0 the protocol fails, otherwise they compute $[b] = 2^{-1}((\sqrt{r^2})^{-1})[r] + 1)$ where $\sqrt{r^2}$ is defined such that $0 \leq \sqrt{r^2} \leq \frac{p-1}{2}$. The complexity is two multiplications in two rounds.

Unbounded Fan-In Multiplications. It is possible to compute prefix-products of arbitrarily many non-zero values in constant rounds using the method of Bar-Ilan and Beaver [1]. Given $[a_1], \ldots, [a_k] \in \mathbb{Z}_p^*$, $[a_{1,i}] = [\prod_{j=1}^{i} a_j]$ may be computed for $i \in \{1, 2, \ldots, k\}$ as follows.

Let $r_0 = 1$ and generate k random non-zero values $[r_1], \ldots, [r_k]$ as described above, however, in parallel with the multiplication of $[r_j]$ and the mask $[s_j]$, compute $[r_{j-1}s_j]$ as well. Then $[r_{j-1}r_j^{-1}]$ may be computed at no cost as $[r_{j-1}s_j] \cdot (r_j s_j)^{-1}$. Each $[a_j]$ is then multiplied onto $[r_{j-1}r_j^{-1}]$ and all these are revealed. We then have

$$[a_{1,i}] = \prod_{j=1}^{i} (a_j r_{j-1} r_j^{-1}) \cdot [r_j] \ .$$

Privacy of the $[a_j]$ is ensured as they are masked by the $[r_j^{-1}]$, and complexity is $5k$ multiplications in three rounds. Regarding the preparation of the masking values as pre-prossessing, the online complexity is k multiplications in a single round.

Constructing a unary counter. Given a secret shared number $[a] < m < p-1$ for some known m, the goal is to construct a vector of shared values

$[V]$ of size m containing all zeros except for the a'th entry which should be 1. This may be constructed similarly to the evaluation of symmetric Boolean functions of [5].

Let $[A] = [a+1]$ and for $i \in \{0, 1, \ldots, m-1\}$ consider the set of functions: $f_i : \{1, 2, \ldots, m+1\} \to \{0, 1\}$ mapping everything to 0 except for $i+1$ which is mapped to 1. The goal is to evaluate these on input A. By Lagrange interpolation, each may be replaced by a *public* m-degree polynomial over $\mathbb{Z}_p$. Using a prefix-product, $[A], \ldots, [A^m]$ may be computed; once this is done, the computation of the entries of $[V]$ is costless. Thus, complexity is equivalent to prefix-product of m terms.

4. A High-level View of Comparison

The overall idea behind the proposed protocol is similar to that of other comparison protocols, including that of [11]. The problem of comparing two secret shared numbers is first transformed to a comparison between shared values where sharings of the binary representations are present; this greatly simplifies the problem. The main difference between here and previous solutions is that we consider inputs of marginally bounded size: $[a], [b] < \lfloor \frac{p-1}{4} \rfloor$. This assumption reduces complexity but may be dropped; the section concludes with a sketch of the required alterations allowing a general comparison as in previous works. Once the problem is transformed to comparison of bitwise represented values, the novel secure computation of the following sections may be applied.

The goal is to compute a bit $[\rho] = ([a] < [b])$. The transformation of the problem proceeds as follows, first the comparison of $[a'] = 2[a] + 1 < \frac{p-1}{2}$ and $[b'] = 2[b] < \frac{p-1}{2}$ rather than that of $[a]$ and $[b]$ is considered. This does not alter the result, however, it ensures that the compared values are not equal, which will be required below.

As $[a'], [b'] < \frac{p-1}{2}$, comparing them is equivalent to determining whether $[z] = [a'] - [b'] \in \{1, 2, \ldots, (p-1)/2\}$. In turn this is equivalent to determining the least significant bit of $2[z]$, written $[(2z)_0]$. If $z < (p-1)/2$, then $2z \in \mathbb{Z}_p$ is even; if not then a reduction modulo p occurs and it is odd.

The computation of $[(2z)_0]$ is done by first computing and revealing $[c] = 2[z] + [r]_B$, where $[r]_B$ is uniformly random. Noting that $[(2z)_0]$ depends only on the least significant bits of c and $[r]_B$ (c_0 and $[r_0]$) and whether an overflow (a modulo p reduction) has occurred in the computation of c. The final result is simply the xor of these three Boolean values.

It can be verified that determining overflow is equivalent to computing $c < [r]_B$; the remainder of the article describes this computation. Privacy is immediate and follows from the security of the secret sharing scheme. Seeing c does not leak information, as $[r]_B$ is uniformly random then so is c. The full complexity analysis is postponed until Sect. 10.

Unbounded $[a]$ and $[b]$. If it is not ensured that $[a], [b] < \lfloor \frac{p-1}{4} \rfloor$ then further work must be performed in order to perform a comparison. Similarly to [11] bits $[t_=] = [a = b]$, as well as $[t_a] = [a < (p-1)/2]$, $[t_b] = [b < (p-1)/2]$, and $[t_\delta] = [a - b < (p-1)/2]$ may be computed. Based on these, the final result may be determined by a small Boolean circuit which translates to a simple secure computation over $\mathbb{Z}_p$. The result is either immediate (if $[a] = [b]$) or may be determined by the three latter bits.

5. The DGK Comparison Protocol

In [6] Damgård et al. proposed a two-party protocol for comparison based on a homomorphic encryption scheme. They consider a comparison between a publicly known and a bitwise encrypted value with the output becoming known. Though their setting is quite different, ideas may be applied in the present one. Initially, we consider comparison of two unknown, ℓ-bit, bitwise shared values $[a]_B$ and $[b]_B$, determining $[a] < [b]$ is done by considering a vector $[E] = [e_{\ell-1}], \ldots, [e_0]$ of length ℓ with

$$[e_i] = [s_i] \left(1 + [a_i] - [b_i] + \sum_{j=i+1}^{\ell-1} ([a_j] \oplus [b_j]) \right) \tag{1}$$

where the $[s_i]$ are uniformly random non-zero values.

Theorem 2 (DGK)**.** *Given two bitwise secret shared values $[a]_B$ and $[b]_B$, the vector $[E]$, given by equation 1 will contain uniformly random non-zero values and at most one* 0*. Further, a* 0 *occurs exactly when $[a] < [b]$.*

Proof. The larger of two values is characterized by having a 1 at the most significant bit where the numbers differ. Letting m denote the most significant differing bit-position, the intuition behind the $[e_i]$ is that $[e_{\ell-1}], \ldots, [e_{m+1}]$ will equal their respective $[s_i]$, and $[e_{m-1}], \ldots, [e_0]$ will also be uniformly random and non-zero, as $1 \leq \sum_{j=i+1}^{\ell-1} ([a_j] \oplus [b_j]) \leq \ell$ for $i < m$. Finally, $[e_m]$ will then be 0 exactly when $[b_m]$ is 1, i.e. when $[b]$ is larger than $[a]$, otherwise it is random as well. □

Though the non-zero elements of vector $[E]$ are random, the location of the zero leaks information. In [6] this is solved by the two-party nature of the protocol – the $[e_i]$ are encrypted, with one party knowing the decryption key, while the other permutes them. The multiparty setting here requires this to be performed using arithmetic.

6. Creating Random Bitwise Shared Values

Based on the comparison of Sect. 5, a random, bitwise shared value $[r]_B$ may be generated such that it is less than some k-bit bound m. This is accomplished by generating the k bits $[r_{k-1}], \ldots, [r_0]$ and verifying that this

represents a value less than m. Noting that information is only leaked when a 0 occurs in the $[e_i]$, ensuring that this coincides with the discarding of $[r]_B$ if it is too large suffices. Based on the above, the k $[e_i]$ are computed as

$$[e_i] = [s_i] \left(1 + (m-1)_i - [r_i] + \sum_{j=i+1}^{k-1} ((m-1)_j \oplus [r_j]) \right) \tag{2}$$

where $(m-1)_i$ denotes the i'th bit of $m-1$. By Theorem 2 this will contain a 0 exactly when $m - 1 < [r]_B \Leftrightarrow [r]_B \geq m$.

The complexity of generating random bitwise shared values consists of the generation of the k bits and the k masks, $[s_{k-1}], \ldots, [s_0]$, plus the k multiplications needed to perform the masking. Overall this amounts to $4k$ multiplications in three rounds as the $[r_i]$ and $[s_i]$ may be generated in parallel (and the $[s_i]$ need not be verified non-zero, this increases the probability of failure marginally). The probability of success depends heavily on m, though it is at least $1/2$. Note that this may be used to generate uniformly random elements of $\mathbb{Z}_p$.

7. Avoiding Information Leaks

Though Sect. 5 has reduced the problem of general comparison to that of comparing a public value to a bitwise shared one, $c < [r]_B$, two problems remain with regard to the DGK-protocol in the present setting. First off, the result is stored as the existence of a 0 in a vector $[E]$ of otherwise uniformly random non-zero values, rather than as a single shared bit $[\rho] \in \{0, 1\}$ which is required to conclude the computation. Second, the position of the 0 in $[E]$ (if it exists) leaks information.

Hiding the result. The former of the two problems may be solved by "masking the result" thereby allowing it to be revealed. Rather than attempting to hide the result, the comparison will be hidden. Formally, a uniformly random value $[s] \in \{-1, 1\}$ is generated, this is equivalent to generating a random bit. The $[e_i]$ of equation 1 are then replaced by

$$[e_i] = [s_i] \left(1 + (c_i - [r_i])[s] + \sum_{j=i+1}^{\ell-1} (c_j \oplus [r_j]) \right) \quad .$$

As $[z]$ is non-zero, we have $c \neq [r]_B$, and thus the desired result is the one obtained, xor'ed with $-2^{-1}([s] - 1)$; if the comparison was flipped ($[s] = -1$) so is the output. Note that the observation made here implies that when considering comparison, it suffices to consider public results: Shared results are immediately available by randomly swapping the inputs.

Hiding the location of the *0*. In order to hide the location of the 0, the $[e_i]$ must be permuted, however, as the non-zero entries are uniformly random, it suffices to *shift* $[E]$ by a unknown, random amount $v \in \{0, 1, \ldots, \ell - 1\}$. I.e. constructing a vector $[\tilde{E}]$ with $[\tilde{e}_i] = [e_{(i+v) \bmod \ell}]$.

Shifting an ℓ-term vector by some unknown amount $[v]$ may be done by encoding $[v]$ in a *shifting-vector* $[W]$ of length ℓ with entries

$$[w_i] = \begin{cases} 0 & i \neq [v] \\ 1 & i = [v] \end{cases}$$

Note that this is *exactly* the unary counter of Sect. 4. Given this, an entry of the shifted vector may be computed as:

$$[\tilde{e}_i] = [e_{(i+v) \bmod \ell}] = \sum_{j=0}^{\ell-1} [e_j] \cdot [w_{(j+i) \bmod \ell}] \ , \tag{3}$$

however, this implies quadratic complexity overall. Thus, rather than shifting the $[e_i]$, the $[\tilde{e}_i]$ will be computed based on already shifted values, i.e. we will consider $[\tilde{r}_i] = [r_{(i+v) \bmod \ell}]$ and $[\tilde{c}_i] = c_{(i+[v]) \bmod \ell}$ along with shifted sums of xor's (the masks $[s_i]$ need not be shifted). This leaves us with two distinct problems:

1 Obtain the shifted bits of $[r]_B$ and c.

2 Obtain the shifted sums of the xor'ed bits

The solutions to these problems are described in the following sections.

8. Shifting Bits

The first thing to note is that securely shifting *known* values is costless, though naturally the result is shared. This is immediate from equation 3, thus the shifted bits of c, $[\tilde{c}_{\ell-1}], \ldots, [\tilde{c}_0]$, are available at no cost.

Regarding $[r]$, shifting its bits is as difficult as shifting the $[e_i]$. Thus an alternative strategy must be employed, namely generating "shifted" bits and constructing the *unshifted* $[r]$ from these. Let $[v]$ denote the shifting-value and $[W]$ the shifting vector, and compute an ℓ-entry bit-vector $[K]$ as

$$[k_i] = 1 - \sum_{j=0}^{i} [w_j] \tag{4}$$

at no cost. Clearly $[k_i] = 1$ for $i < [v]$ and $[k_i] = 0$ for $i \geq [v]$. $[r]$ may then be computed based on random "shifted" bits as

$$[r] = [2^{-v}] \cdot \left(\sum_{i=0}^{\ell-1} 2^i [\tilde{r}_i] \left(1 + [k_i](2^\ell - 1) \right) \right) \ ,$$

where $[2^{-v}]$ is computed by the costless $\sum_{i=0}^{\ell-1} 2^{-i}[w_i]$. It can be verified that the above is correct, assuming that the bits represent a value in $\mathbb{Z}_p$. $[2^{-v}]$ performs the "unshifting," while the final parenthesis handles "underflow."

It remains to verify that the bits represent a shifted value of $\mathbb{Z}_p$. This may be done by performing the exact same computation as needed for the full comparison, i.e. shifting the bits of $p-1$, computing the shifted sums of xors (as described below) and masking these. The shifting value $[v]$ is "hard-coded" into $[r]$, but it may be "reused" for this, as no information is leaked when no 0 occurs, i.e. when $[r] < p$. When information is leaked, the value is discarded anyway.[1]

Generating $[r]$ and the shifted bits $[\tilde{r}_i]$ involves creating a uniformly random value $[v] \in \{0, 1, \ldots, \ell-1\}$ as described in Sect. 4. This requires 3 rounds and $8\log_2(\ell)$ multiplications, as a factor of two attempts are needed when p may be arbitrary. $[W]$ may be computed from $[v]$ using $5\log_2(\ell)$ multiplications, but only one additional round as pre-processing parallelizes. The shifted bits of $[r]$ and the masks $[s_i]$ may be generated concurrently using 3ℓ multiplications; these masks will not be ensured non-zero as this is unlikely to occur, this results in a slightly larger probability of aborting. Checking $[r]_B < p$ adds four rounds and 4ℓ multiplications, providing a total of 8 rounds, $7\ell + 13\log_2(\ell)$ multiplications. A factor of two attempts of this implies $14\ell + 26\log_2(\ell)$ multiplications in 8 rounds per random value generated.

Computing $[r]$ based on the shifted bits uses an additional round and 1 multiplication as the $[\tilde{r}_i k_i]$ are needed for the verification of $[r]$ anyway. Concurrently, a sharing of the least significant bit of $[r]$ may be obtained using ℓ multiplications as described by equation 3. This value is needed for the overall computation.

9. Shifting the Sums of Xor's

It remains to describe how to "shift" the sums of the xor's, $\sum_{j=i+1}^{\ell-1} [r_j] \oplus c_j$. The shifted xor's $[\tilde{c}_i \oplus \tilde{r}_i]$ may be computed directly based on the previous section. The sum, however, consists of the terms from the *current* index to the *shifted* position of the most significant bit, which may or may not "wrap around." Focusing on a single $[\tilde{e}_i]$, we consider two cases: $i < [v]$ and $i \geq [v]$. Naturally, $[v]$ is unknown, however, the correct case is determined by $[k_i]$ of above.

When $i < [v]$ the terms have been shifted more than i, thus we must sum from $i+1$ to $[v]-1$, this equals:

$$[\sigma_i^{<}] = \sum_{j=i+1}^{\ell-1} [k_j] \cdot ([\tilde{r}_j \oplus \tilde{c}_j]) \quad .$$

Multiplying by the $[k_j]$ ensures that only the relevant terms are included. For $i \geq [v]$ the terms are shifted less than i, thus the most significant bit does not

pass the i'th entry. The sum must therefore be computed from i to $\ell - 1$ *and* from 0 to $[v] - 1$. Viewing this as the sum of all terms *minus* the sum from $[v]$ to i we get

$$[\sigma_i^{\geq}] = \left(\sum_{j=0}^{\ell-1} [\tilde{r}_j \oplus \tilde{c}_j] \right) - \left(\sum_{j=0}^{i} (1 - [k_j]) \cdot [\tilde{r}_j \oplus \tilde{c}_j] \right) \ .$$

Thus, as the computation of the two candidates $[\sigma_i^{<}]$ and $[\sigma_i^{\geq}]$ for each position i is linear, computing all sums simply require selecting the correct one for each entry based on $[k_i]$:

$$[k_i][\sigma_i^{<}] + (1 - [k_i])\,[\sigma_i^{\geq}] \ .$$

All this requires 3 rounds and 4ℓ multiplications, however, noting that

$$[k_j] \cdot ([\tilde{r}_j] \oplus [\tilde{c}_j]) = [k_j \tilde{r}_j] + ([k_j] - 2[k_j \tilde{r}_j])\,[\tilde{c}_j] \ ,$$

the $[k_j \tilde{r}_j]$ is pre-prossessed. This reduces the online complexity of calculating the sums to 1 round and 3ℓ multiplications.

10. Overall Analysis and Optimizations

Pre-processing. Initially a uniformly random value $[v] \in \{0, 1, \ldots, \ell - 1\}$ must be created. This value is used to construct a random shifting vector $[W]$ of length ℓ. Concurrently the *shifted* bits of $[r]$ are generated. It must then be verified that $[r] < p$. Once this is done the values $[r]$ and $[r_0]$ may be computed. In addition, the non-zero masks $[s_i]$, the sign bit $[s]$ must be generated, and $[s_i \cdot s]$ and $[s_i \cdot k_i]$ computed in order to reduce online complexity as described below.

The overall cost of creating the $[v]$, $[W]$ and shifted bits $[\tilde{r}_i]$ of $[r]$ is eight rounds, $14\ell + 26\log_2(\ell)$ multiplications. In addition, creating $[r]$, $[r_0]$, $[s_i]$, $[s]$, $[s_i \cdot s]$, and $[s_i \cdot k_i]$ adds another round and $6\ell + 3$ multiplications (as $[k_i \tilde{r}]$ are already computed). The total pre-processing complexity is therefore nine rounds $20\ell + 26\log_2(\ell) + 3$ multiplications.

Round complexity may be further reduced as $[v]$ and $[W]$ may be created in parallel, also $[r]$ and $[r_0]$ may be computed before $[r] < p$ is verified. Additionally, the verification of $[r] < p$ may be reduced by one round at the cost of ℓ additional multiplications by masking the sums of xor's and the remainder of Eq. 2 separately. The complexity of pre-processing is thereby changed to 6 rounds, though $23\ell + 36\log_2(\ell) + 4$ multiplications are required.

Online. The computation of $c = [r] + 2\left((2[a]+1) - 2[b]\right)$ where $[a], [b] < \lfloor \frac{p}{4} \rfloor$ is costless. The intuition behind the online computation of the $[\tilde{e}_i]$ is

$$[\tilde{e}_i] = [s_i]\left(1 + ([\tilde{c}_i] - [\tilde{r}_i])\,[s] + \left([k_i][\sigma_i^{\geq}] + (1-[k_i])\,[\sigma_i^{<}]\right)\right) \ ,$$

however, this is rephrased to reduce online round complexity:

$$[s_i] + [s_i \cdot s]\,([\tilde{c}_i] - [\tilde{r}_i]) + \left(([s_i \cdot k_i])[\sigma_i^{\geq}] + ([s_i] - [s_i \cdot k_i])\,[\sigma_i^{<}]\right) \ . \qquad (5)$$

This implies two rounds and 4ℓ multiplications: First $[\sigma_i^{<}]$ and $[\sigma_i^{\geq}]$ are determined using ℓ multiplications, then three parallel multiplications are performed for each of the ℓ instances of Eq. 5. Letting e denote if a 0 was encountered in the $[\tilde{e}_i]$, the final result is

$$[\rho] = [r_0] \oplus c_0 \oplus \left(\left(-2^{-1}([s]-1)\right) \oplus e\right) = [r_0 \oplus (-2^{-1}([s]-1))] \oplus c_0 \oplus e \ ,$$

where the multiplication needed parallelizes with the above. Indeed it is even possible to perform this in pre-processing, though it adds another round.

The overall complexity is therefore $24\ell + 26\log_2(\ell) + 4$ multiplications in 11 rounds or 8 rounds $27\ell + 36\log_2(\ell) + 5$ if the pre-processing round complexity is optimized as described above.

Bibliography

[1] Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Rudnicki, P., ed.: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, New York, ACM Press (1989) 201–209

[2] Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., Toft, T.: Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications. BRICS Report Series RS-05-18, BRICS (2005) `http://www.brics.dk/RS/05/18/`.

[3] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for noncryptographic fault-tolerant distributed computations. In: 20th Annual ACM Symposium on Theory of Computing, ACM Press (1988) 1–10

[4] Cramer, R., Damgård, I.: Secure distributed linear algebra in a constant number of rounds. In Kilian, J., ed.: Advances in Cryptology - Crypto 2001, Berlin, Springer-Verlag (2001) 119–136 Lecture Notes in Computer Science Volume 2139.

[5] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi, S., Rabin, T., eds.: Theory of Cryptography. Volume 3876 of Lecture Notes in Computer Science (LNCS)., Springer (2006) 285–304

[6] Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for on-line auctions. In: ACISP 07: 12th Australasian Conference on Information Security and Privacy. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany (2007) Forthcoming.

[7] Fischlin, M.: A cost-effective pay-per-multiplication comparison method for millionaires. In Naccache, D., ed.: Topics in Cryptology – CT-RSA 2001. Volume 2020 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2001) 457–471

[8] Gennaro, R., Rabin, M., Rabin, T.: Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM Press (1998) 101–111

[9] Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st Annual Symposium on Foundations of Computer Science, Las Vegas, Nevada, USA, IEEE Computer Society Press (November 12–14, 2000) 294–304

[10] Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Proceedings of ICALP 2002, Berlin, Springer-Verlag (2002) 244–256 Lecture Notes in Computer Science Volume 2380.

[11] Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: PKC 2007: 10th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany (2007)

[12] Shamir, A.: How to share a secret. Communications of the ACM **22**(11) (1979) 612–613

[13] Schoenmakers, B., Tuyls, P.: Practical two-party computation based on the conditional gate. In Lee, P.J., ed.: Advances in Cryptology – ASIACRYPT 2004. Volume 3329 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2004) 119–136

Paper C

Multiparty Comparison, An improved multiparty protocol for comparison of secret-shared values

Tord Ingolf Reistad

In Proceedings of SECRYPT 2009, International conference on security and cryptography

Milano, Italy. July 6-10, 2009

MULTIPARTY COMPARISON, AN IMPROVED MULTIPARTY PROTOCOL FOR COMPARISON OF SECRET-SHARED VALUES

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Abstract Given any linear secret sharing scheme with a multiplication protocol, we show that a set of players holding shares of two values $a, b \in \mathbb{Z}_p$ for some prime p (written $[a]$ and $[b]$), it is possible to compute a sharing $[result]$ such that $[result] = ([a] < [b])$. The protocol maintains the same security against active/adaptive adversaries as the underlying secret sharing scheme.

1. INTRODUCTION

In the millionaire problem [Yao, 1982] two or more millionaires want to know which one of them is richer without revealing each other's wealth. This was one of the first protocols for multiparty computation, and illustrates the challenges in multiparty protocols in practical applications. As the goal of the protocol between the millionaires should not only keep the input private, but also compute the result quickly.

In the millionaire problem each player (millionaire) knows his own wealth, which is the input to that is sent to the protocol. In a more generalized setting the inputs might not be explicitly known to any single party, but resulting from a linear combination of secret inputs by several parties, or from an earlier intermediate result.

In general a multiparty computation (MPC) is a computation between a number of parties $P_1, \ldots, P_n$. These parties have *private* inputs $x_1, \ldots, x_n$ and they want to compute some function f on these inputs, where $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ such that P_i learns y_i but nothing else. The players are mutually mistrusting and do not wish to share their inputs.

When considering concrete applications, the function f is often computed using Shamir secret sharing scheme over a field $\mathbb{Z}_p$ for some large odd prime p. The function is computed using *simulated integer arithmetic*. This means that although the values are points in a finite field, they can be used as integer values in $\mathbb{Z}$. This is analogous to current day computers where each value is

stored as a limited set of bits. The difference is that in the multiparty protocol there is no way to efficiently check for overflow, the prime p will therefore have to be chosen such that all simulated integer values are less than p for all possible inputs.

Although there exists efficient protocols for addition and multiplication of secret shared values, most functions or algorithms f require additional operators such as comparison and equality checking. This motivates us in constructing specialized efficient protocols that realizes such operators. These operator primitives may be used repeatedly and any improvement in the implementations of the primitive operators will lead to a similar improvement in an overall application.

Comparison in a multiparty computation setting could be solved by circuit based computations. All inputs would then be split into bits. Comparison between inputs would then be easy, but this would make addition and multiplication much more complicated so the overall effect would be slower computer programs.

This work describes a protocol for one such primitive operator, namely comparison (inequality-testing) of secret shared values. That is, given two secret shared values $[a]$ and $[b]$, a secret shared bit $[result] = [a < b]$ is obtained stating whether one value is larger than the other without leaking any information. This protocol is more efficient than all previously published constant round protocols for comparison.

Section 2 discusses this work in relation to previously published papers. Sections 3 and 4 introduce the model as well as a number of primitives required. Most of these are well-known, and are included in order to provide detailed analysis of our protocol. Section 5 Gives first a high-level view of the protocol and then discusses the details. Finally Section 6 gives the conclusion and further improvements.

2. RELATED WORK

Research on multiparty computation (MPC) has mostly focused on either primitives for multiparty computations or on some concrete applications such as actions [Bogetoft et al., 2005, Fischlin, 2001, Schoenmakers and Tuyls, 2004]. This paper considers neither of these cases. It assumes underlying primitives for multiparty computations with addition and multiplication of field elements. It also does not consider concrete applications as comparison of elements are just one piece in a framework for multiparty computations.

Damgård et al. provided the first constant rounds comparison in the present setting [Damgård et al., 2006], the protocol relied on bit-decomposition of values this approach required $\mathcal{O}(\ell \log(\ell))$ secure multiplications where $\ell = \log(p)$. Comparison was later improved by Nishide and Ohta [Nishide and Ohta, 2007] who reduced the complexity to $\mathcal{O}(\ell)$ multiplications.

The most recent solutions have concentrated on a binary representation of the values being compared. Thus, unless a radically different approach is

taken, improving on the $\mathcal{O}(\ell)$ bound does not seem feasible. The present work builds on sub-protocols and ideas of [Damgård et al., 2006], [Nishide and Ohta, 2007] and [Reistad and Toft, 2007] and aims at reducing the constants hidden under big-$\mathcal{O}$. These costs hidden under big-$\mathcal{O}$ are becoming more relevant as multiparty is implemented and used in practical applications [Bogetoft et al., 2008].

Presented in	Type	Rounds		Multiplications	
		overall	online	overall	online
[Damgård et al., 2006]	A	44	37	$184\ell \log_2(\ell) + 209\ell$	$21\ell + 56\ell \log_2(\ell)$
[Nishide and Ohta, 2007]	A	15	8	$279\ell + 5$	$15\ell + 5$
[Reistad and Toft, 2007]	A	12	4	$84\ell + 78 \log_2(\ell) + 17$	$18l + 5$
This paper	A	8	2	$58.5\ell + 33$	4.5ℓ
[Damgård et al., 2006]	R	44	37	$184\ell \log_2(\ell) + 165\ell$	$21\ell + 56\ell \log_2(\ell)$
[Nishide and Ohta, 2007]	R	13	6	$49\ell + 5$	$5\ell + 1$
[Reistad and Toft, 2007]	R	10	2	$17\ell + 26 \log_2(\ell) + 4$	5ℓ
[Reistad and Toft, 2007]	R	8	2	$20\ell + 36log_2(\ell) + 6$	5ℓ
This paper	R	6	0	$7,5\ell + 11$	$1,5\ell$

Table 1. Complexities of comparison protocols

Table 1 compares the solution presented in this paper to those of Damgård et al., Nishide and Ohta, and Reistad and Toft. Type A refers to comparison of arbitrary values $[a], [b] \in \mathbb{Z}_p$, while R is for restricted values $[a], [b] < \lfloor \frac{p}{4} \rfloor$ and a prime $p = 2^\ell - c$ where c is a small integer; when using $\mathbb{Z}_p$ to simulate integer computation, it is not unreasonable to choose p in such a way to accommodate these two assumption. Furthermore to give the protocols equal footing we assume that all protocols use the same underlying protocols see section 4. E.g. the protocol for creating random bitwise shared values are created using the same protocol. For comparison of arbitrary values in arbitrary fields it is assumed that the underlying protocol will have to be run two times for a total of 8ℓ multiplications. For a well chosen p and restricted values this complexity can be assumed to be close to 2ℓ. As the test becomes more efficient and the probability of having to run the underlying protocol twice becomes insignificant.

A distinction is also made between online complexity and pre-processing complexity. Pre-processing are all the computations that can be made independent of the private inputs. Online computations are all those computations that can only be made once the private inputs are available.

3. MODEL

We assume a linear secret sharing scheme that allows for multiparty addition and a multiplication of secret shared values, to be shared among $n > 2$ parties. The security properties of the secret sharing scheme are inherited, i.e. if the

secret sharing scheme is unconditionally secure against active/adaptive adversaries then so is the protocols proposed. As as example, consider Shamir's scheme along with the protocols of Ben-Or et al. (or the improved protocols of Gennaro et al.) [Shamir, 1979, Ben-Or et al., 1988, Gennaro et al., 1998].

The communication model is that there exist authenticated private channels between each pair of parties. The model assumes that in addition to sharing values and performing secure arithmetic on $\mathbb{Z}_p$, the parties may reveal (reconstruct) shared values. Revealing a secret shared value ensures that the value becomes known by *all* parties.

We use $[a]$ to denote a secret sharing of $a \in \mathbb{Z}_p$ among the n parties, where p is an ℓ-bit prime ($\ell > 7$). The operators are written using an infix notation. For shared values $[a]$ and $[b]$, and constant $c \in \mathbb{Z}_p$, computation of sums will be written as $[a] + c$ and $[a] + [b]$, while products will be written $c[a]$ and $[a][b]$. The first three operator are computed locally, while the fourth operator represents an invocation of the multiplication protocol.

Sharings of bits, $[b] \in \{0, 1\} \subset \mathbb{Z}_p$ will also be considered. Boolean arithmetic is written using infix notation, though it must be realized using field arithmetic. Notably xor of two bits is constructed as $[b_1] \oplus [b_2] = [b_1] + [b_2] - 2[b_1][b_2]$ which is equivalent.

Values may also be *bitwise shared*, written $[a]_B$. Rather than having a sharing of a value itself, $[a]$, sharings of the bits of the binary representation of a are given, i.e. $[a_0], \ldots, [a_{\ell-1}] \in \{0, 1\}$ such that

$$[a]_B = \sum_{i=0}^{\ell-1} 2^i [a_i] \tag{1}$$

for $\ell = \lceil \log(p) \rceil$, with the sum being viewed as occurring over the integers. Note that $[a]$ is easily obtained from $[a]_B$ as it is a linear combination.

Similarly a publicly known value c can be split into bits, where c_i represents the i'th bit of c. c_0 is the least significant bit of c.

Complexity. When considering complexity, the focus of the analysis will be on communication. Similar to other work, focus will be placed on the number of invocations of the multiplication protocol as this is considered the most costly of the primitives. It is assumed that invocations of the multiplication protocol parallelize and multiplications are executed in parallel when possible. When we say "one round" we mean one round of arbitrary many invocations of the multiplication protocol, all performed in parallel among the parties.

Multiplication by constants and addition require no interaction and is therefore considered cost-less. The complexity of sharing and revealing is seen as negligible compared to that of multiplication. Rounds for reconstruction are disregarded as in other work.

4. SIMPLE PRIMITIVES

This section introduces a number of simple primitives required below. Most of these sub-protocols are given in [Damgård et al., 2006] but are repeated here in order to provide a detailed analysis as well as for completeness. Most of these are related to the generation of random values unknown to all parties. It is important to note that these may fail, however, this does not compromise the privacy of the inputs – failure simply refers to the inability to generate a proper random value (which is detected). Generally the probability of failure will be of the order $1/p$, which for simplicity will be considered negligible, see [Damgård et al., 2006] for further discussion.

Random element generation. A sharing of a uniformly random, unknown value $[r]$ may be generated by letting each party share a uniformly random value $r_i \in \mathbb{Z}_p$. The sum $[r] = \sum r_i \mod p$ of these is uniformly random and unknown to all, even in the face of an active adversary. The complexity of this is assumed to be equivalent to one invocation of the multiplication protocol.

Random bits. The parties may create a uniformly random bit $[b] \in \{0, 1\}$. The parties may generate a random value $[r]$ and reveal its square, r^2. If this is 0 the protocol fails, otherwise they compute $[b] = 2^{-1}((\sqrt{r^2})^{-1})[r] + 1)$ where $\sqrt{r^2}$ is defined such that $0 \leq \sqrt{r^2} \leq \frac{p-1}{2}$. The complexity is two multiplications in two rounds.

Creating Random Bitwise Shared Values. The parties may create a uniformly random bitwise shared value $[r]_B$ less than some m, $k = \lceil log_2(m) \rceil$, as described in [Reistad and Toft, 2007]. This is accomplished by generating k bits $[r_{k-1}], \ldots, [r_0]$ and verifying that this represents a value less than m. The verification is done by creating a vector with elements $[e_i]$ and revealing them. Note that this protocol will be used to generate uniformly random bitwise shared elements of $\mathbb{Z}_p$ (done by setting $m = p$).

$$[e_i] = [s_i]\Big(1 + (m-1)_i - [r_i] + \sum_{j=i+1}^{k-1} ((m-1)_j \oplus [r_j])\Big) \tag{2}$$

The notation $(m-1)_i$ denotes the i'th bit of $m-1$ and $[s_i]$ are random shares in $\mathbb{Z}_p$. Revealing the shares $[e_i]$ gives a vector that will contain a 0 element only when $m - 1 < [r]_B \Leftrightarrow [r]_B \geq m$ for proof see [Reistad and Toft, 2007]. Note that this protocol will leak information about the individual bits $[r_i]$ if there is a 0 element, but this coincides with discarding the bits $[r_i]$.

The complexity of generating random bitwise shared values consists of the generation of the k bits and k masks, in addition there are needed k multiplications to perform the masking. Overall this amounts to $4k$ multiplications in three rounds as the $[r_i]$ and $[s_i]$ may be generated in parallel. For added

efficiency the $[e_i]$ where $(m-1)_i = 1$ do not need to be calculated or revealed, as they are guaranteed to be non-zero. For $m = 2^k - c$, where c is a small integer the complexity can be reduced to $2k + 2\log(c)$.

Additional Improvement. The probability that the protocol above fails to return a bitwise secret shared value, depends upon m. The worst case scenario is $m = 2^k + 1$, which makes the probability of failing equal to $1/2$. For 5 additional multiplications and no extra rounds the probability of restarting can be lowered to an upper bound of $1/4$ for all m. This is done by noting that if the two highest order bits of m are 10, then the protocol fails if the products of the two highest bits of $[r]_B$ is 1. The solution is therefore to create 2 or sets of candidates for highest order bits. The two bits in each of the candidates can be multiplied together and revealed, and the candidate only used if the product is 0. This introduces no additional rounds of communication as $[t_{k-1}t_{k-2}]$ can be computed in parallel with computing the bits.

$$[r_{k-1}r_{k-2}] = 4^{-1}\left(\frac{[t_{k-1}t_{k-2}]}{\sqrt{t_{k-1}^2}\sqrt{t_{k-2}^2}} + \frac{[t_{k-1}]}{\sqrt{t_{k-1}^2}} + \frac{[t_{k-2}]}{\sqrt{t_{k-2}^2}} + 1\right) \tag{3}$$

Unbounded Fan-In Multiplications. It is possible to compute prefix-products of arbitrarily many non-zero values in constant rounds using the method of Bar-Ilan and Beaver [Bar-Ilan and Beaver, 1989]. Given $[a_1], \ldots, [a_k] \in \mathbb{Z}_p^*$, $[a_{1,i}] = [\prod_{j=1}^{i} a_j]$ may be computed for $i \in \{1, 2, \ldots, k\}$ as follows.

Let $u_0 = 1$ and generate k random non-zero values $[u_1], \ldots, [u_k]$, however, in parallel with the multiplication of $[u_j]$ and its mask $[v_j]$, compute $[u_{j-1}v_j]$ as well. Then $[u_{j-1}u_j^{-1}]$ may be computed at no cost as $[u_{j-1}v_j] \cdot (u_j v_j)^{-1}$. Each $[a_j]$ is then multiplied onto $[u_{j-1}u_j^{-1}]$ and all these are revealed. We then have:

$$[a_{1,i}] = \prod_{j=1}^{i} (a_j u_{j-1} u_j^{-1}) \cdot [u_i] \tag{4}$$

Privacy of the $[a_j]$ is ensured as they are masked by the $[u_j^{-1}]$, and complexity is $5k$ multiplications in three rounds. Regarding the preparation of the masking values as pre-processing, the online complexity is k multiplications in a single round.

5. THE COMPARISON PROTOCOL

The comparison protocol consists of 3 sub protocols, the first sub-protocol transforms the comparison $[a] < [b]$ into a comparison $[r]_B > c$, where $[r]_B$ is a random secret shared value and c is a value known to all parties. This transformation is done so the other two protocols can work on individual bits.

The second sub-protocol transforms the comparison $[r]_B > c$ into a single $[x]$ this share $[x]$ is made in such a way that $[x] < \sqrt{4p}$ for all $[r]_B$ and c and the least significant bit of $[x]$ is equal to the result of the comparison test between $[r]$ and c.

The third sub-protocol extracts the least significant bit of $[x] < \sqrt{4p}$ efficiently. Once the least significant bit of $[x]$ is extracted then the bit representing the answer to the original comparison between $[a] < [b]$ is computed with two xor's.

5.1 First Transformation

When $[a], [b] < \frac{p-1}{2}$ and $[z] = [a] - [b]$, then $[a] < [b]$ is equivalent to determining the least significant bit of $2[z]$, written $[(2z)_0]$. The least significant bit of $[(2z)_0]$ is found by computing and revealing c, where

$$[c] = 2[z] + [r]_B = 2([a] - [b]) + [r]_B \tag{5}$$

c_0 is the least significant bit of c, $[r_0]$ is the least significant bit of $[r]_B$ and $[r]_B$ is a random bitwise shared value. We then have the following equation:

$$\big([a] < [b]\big) = [(2z)_0] = c_0 \oplus [r_0] \oplus \big([r]_B > c\big) \tag{6}$$

With this first transformation the comparison between $[a]$ and $[b]$ has then been transformed to the problem of comparing $[r]_B > c$. For more detail on the transformation see [Reistad and Toft, 2007], and for unbounded values of $[a]$ and $[b]$ see [Nishide and Ohta, 2007].

The cost of this transformation is the creation of one secret shared value and one secret shared xor. The calculation of the secret shared value can be done in 3 rounds in parallel with other pre-processing and the xor adds one online multiplication in one round.

Privacy follows from the security of the secret sharing scheme. Revealing c does not leak information, because $[r]_B$ is a uniformly random secret shared value. Therefore the distribution of c is also uniformly random.

5.2 Computing X

Theorem 1:. Given a random secret shared value $[r]_B$ and a publicly known value c. Let the secret shared value $[x]$ be constructed as:

$$[x] = \sum_{i=0}^{l-1} [r_i](1 - c_i)2^{\sum_{j=i+1}^{l-1} c_j \oplus [r_j]} \tag{7}$$

The least significant bit of $[x]$, written $[x_0]$ is equal to the value $[r_i]$, where i is the most significant bit where $[r_i] \neq c_i$. Or in other words $[x_0]$ is equal to the boolean statement $([r]_B > c)$.

Proof:. Starting from the most significant bits we see that $[r_i](1-c_i)$ and the sum $\sum_{j=i+1}^{l-1} c_j \oplus [r_j]$ are 0, as long as $[r_i] = c_i$. Beginning with the highest order bit where $[r_i] \neq c_i$, the expression $[r_i](1-c_i)$ is equal to $[r_i]$. For lower order bits after the first one where $[r_i] \neq c_i$, the sum $\sum_{j=i+1}^{l-1} c_j \oplus [r_j]$ becomes non-zero. Therefore all lower terms except the first one where $[r_i] \neq c_i$ become either. 0 or some value 2^k, where $k < 1$. □

For the greatest efficiency an even more complex form for the function for $[x]$ will be used. The function $[x]$ is essentially the same only that for each summation not one but two bits of $[r_i]$ and c_i are considered at the same time. Therefore $[r_i](1-c_i)$ will be expressed as $r_i > c_i$ for 2 bits at a time, and $c_j \oplus [r_j]$ will only return 1 if both pair of bits are equal. This ensures that $[x] < 2^{\frac{l+1}{2}}$.

This functions which compares two bits at a time is more efficient. In addition it also avoids a problem in the next protocol. As that the function $[x]$ as stated above has a worst case bit length of ℓ.

In the 2-bit version each pair of secret shared bits have to be multiplied together at a cost of $0,5\ell$ multiplications. This also computes the two bit version of $[r_i](1-c_i)$ and $c_j \oplus [r_j]$. Computing $2^{\sum_{j=i+1}^{l-1} c_j \oplus [r_j]}$ can be done with fan-in multiplications in $2,5\ell$ multiplications and the final multiplication costs $0,5\ell$. The protocol therefore takes a total of $3,5\ell$ multiplications and 3 rounds (as the random values for fan-in multiplications can be done in pre-processing). Privacy of computing $[x]$ is immediate as no value is revealed.

5.3 Extracting the Least Significant Bit

Extracting the last bit of a secret shared value $[x]$, where $[x] < \sqrt{4p}$, can be done efficiently. First a bitwise secret shared random variable $[s]_B$ is created. The the value d is calculated and revealed, where

$$[d] = [s]_B + [x] \tag{8}$$

From this we can easily see that the least significant bit of $[x]$, written as $[x_0]$ is equal to $[s_0] \oplus d_0$, when $d > \sqrt{4p}$.

To create a generalized version for finding $[x_0]$, $[s]_B$ will have to be split into 3 parts $[s]_B = 2^{l-1}[s_{l-1}] + 2^{l-2}[s_{l-2}] + [\hat{s}]_B$. Note that two bits of information are needed from $[s]_B$ as in a worst case scenario p might be $2^{l-1}+1$ and $[s]_B$ might be $2^{l-1}-1$.

We then have that $[\hat{s}]_B + [x] < 2^{l-2} + \sqrt{4p} < p$, for all p. As $[\hat{d}] = [\hat{s}]_B + [x]$ never will need a modulo p reduction.

$$[x_0] = [\hat{s}_0]_B \oplus [\hat{d}_0] \tag{9}$$

The value $[\hat{d}_0]$ cannot be revealed, as this would leak information about $[x]$, on the other hand there are only 4 possible values for $[\hat{d}]$ based upon the value

d, and the shares $[s_{l-1}]$ and $[s_{l-2}]$. Therefore following equation need only one secret shared multiplication to compute $[\hat{d}_0]$: (Note $(d < 2^{l-1})$ returns 1 if $d < 2^{l-1}$).

$$\begin{aligned}[\hat{d}_0] =&(1 - [s_{l-1}] - [s_{l-2}] + [s_{l-1}s_{l-2}])d_0 \\ &+ ([s_{l-2}] - [s_{l-1}s_{l-2}])(d_0 \oplus (d < 2^{l-2})) \\ &+ ([s_{l-1}] - [s_{l-1}s_{l-2}])(d_0 \oplus (d < 2^{l-1})) \\ &+ ([s_{l-1}s_{l-2}])(d_0 \oplus (d < (2^{l-1} + 2^{l-2}))\end{aligned}$$

Going from $[\hat{d}_0]$ to $[x_0]$ adds one multiplication, and going from $[x_0]$ to $[[a] < [b]]$ also add one more multiplication. Therefore once d is known it takes 3 multiplications in 2 rounds to compute the original comparison. To save rounds all combinations of the secret shared values $[s_{l-1}]$, $[s_{l-2}]$, $[s_0]$ and $[r_0]$ can be pre-computed once $[r]_B$ and $[s]_B$ are known. This is done with 11 multiplications and 2 rounds that run in parallel with previous computations. The privacy argument is the same as for the first transformation.

6. CONCLUSION AND FURTHER WORK

We have presented a protocol for comparison in constant rounds that is significantly faster than previous versions. As the most time consuming part of a MPC program is in many cases the comparison protocol, the efficiency of the comparison protocol will directly affect the speed of the program.

The comparison protocol in [Reistad and Toft, 2007] has be implemented in the VIFF framework (http://viff.dk/) therefore the implementation of the protocol in this paper will show in practice how efficient the two protocols are compared to each other.

Further improvements in the online computation can also be done as the computation of $2^{\sum_{j=i+1}^{l-1} c_j \oplus [r_j]}$ when computing $[x]$ in equation 7 can be split into three multiplications.

- $\prod_{j=i+1}^{l-1}(1 + [r_j])$ can be pre-processed.
- $\prod_{j=i+1}^{l-1}(1 + c_j)$ can be computed online without multiparty multiplications.
- The last that product $\prod_{j=i+1}^{l-1}(1 - 3 \cdot 4^{-1}(c_j[r_j]))$ can be pre-computed in such two shares are created and only one of them is opened depending upon c_j.

This will result in the same amount of pre-computations, but will not require any online computations only revealing values.

ACKNOWLEDGEMENTS

The author would like to thank the anonymous reviewers for their comments, Stig Frode Mjølsnes and Harald Øverby for useful discussions on the article and Tomas Toft for comments and fruitful discussions about MPC comparison.

Bibliography

[Bar-Ilan and Beaver, 1989] Bar-Ilan, J. and Beaver, D. (1989). Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Rudnicki, P., editor, *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209, New York. ACM Press.

[Ben-Or et al., 1988] Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press.

[Bogetoft et al., 2008] Bogetoft, P., Christensen, D., Dåmgard, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J., Nielsen, J., Nielsen, K., Pagter, J., Schwartzbach, M., and Toft, T. (2008). Multi-party computation goes live. *Cryptology ePrint Archive, Report 2008/068*.

[Bogetoft et al., 2005] Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., and Toft, T. (2005). Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications. BRICS Report Series RS-05-18, BRICS. `http://www.brics.dk/RS/05/18/`.

[Damgård et al., 2006] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J., and Toft, T. (2006). Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi, S. and Rabin, T., editors, *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science (LNCS)*, pages 285–304. Springer.

[Fischlin, 2001] Fischlin, M. (2001). A cost-effective pay-per-multiplication comparison method for millionaires. In Naccache, D., editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 457–471. Springer-Verlag, Berlin, Germany.

[Gennaro et al., 1998] Gennaro, R., Rabin, M., and Rabin, T. (1998). Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111, New York, NY, USA. ACM Press.

[Nishide and Ohta, 2007] Nishide, T. and Ohta, K. (2007). Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC 2007 International Workshop on Theory and Practice in Public Key Cryptography*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany.

[Reistad and Toft, 2007] Reistad, T. and Toft, T. (2007). Secret sharing comparison by transformation and rotation. In Preproceedings *ICITS*, International Conference on Information Theoretic Security 2007

[Schoenmakers and Tuyls, 2004] Schoenmakers, B. and Tuyls, P. (2004). Practical two-party computation based on the conditional gate. In Lee, P. J., editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, Berlin, Germany.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

[Yao, 1982] Yao, A. (1982). Protocols for secure computation. In *Proceedings of the twenty-third annual IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society.

Paper D

Realizing Distributed RSA Key Generation using VIFF

Atle Mauland, Tord Ingolf Reistad, Stig Frode Mjølsnes

At Norsk informasjonsikkerhetskonferanse 2009 (NISK 09)

Trondheim, Norway, November 24-25, 2009

REALIZING DISTRIBUTED RSA KEY GENERATION USING VIFF

Atle Mauland
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
atle.mauland@gmail.com

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Stig Frode Mjølsnes
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
sfm@item.ntnu.no

Abstract In this experimental work, we have implemented and analyzed the performance of a protocol for multiparty RSA key generation. All players that take part in the key generation protocol get a share of the resulting private key, while no player gets any information about the prime numbers used.

We have implemented a distributed program based on an algorithm proposed by Boneh and Franklin in 1997. The implementation is based on VIFF, a new software framework for realizing multiparty computations. The program is analyzed for key lengths up to 4096 bits. We demonstrate the efficiency of shared RSA key generation employing VIFF and real network communication. We are also able to propose improvements with respect to security and performance compared to the previous published protocol.

1. Introduction

Consider the situation where an encrypted text should be decrypted only if all players agree. An example of this is the voting application, where all talliers have to agree that the voting is finished before the counting of the ballots can begin. This implies that if the voting was done by encrypted ballots then the talliers must first decrypt the ballots.

The ballots can either be encrypted using a symmetric or an asymmetric key system. The advantage of using a symmetric key system is that key is just a random string. The key can then easily be split into shares using any secret sharing scheme. The problem is that the secret key must be used in the encryption process. Therefore, th symmetric key system must find a solution so that no single party gets hold of the secret key.

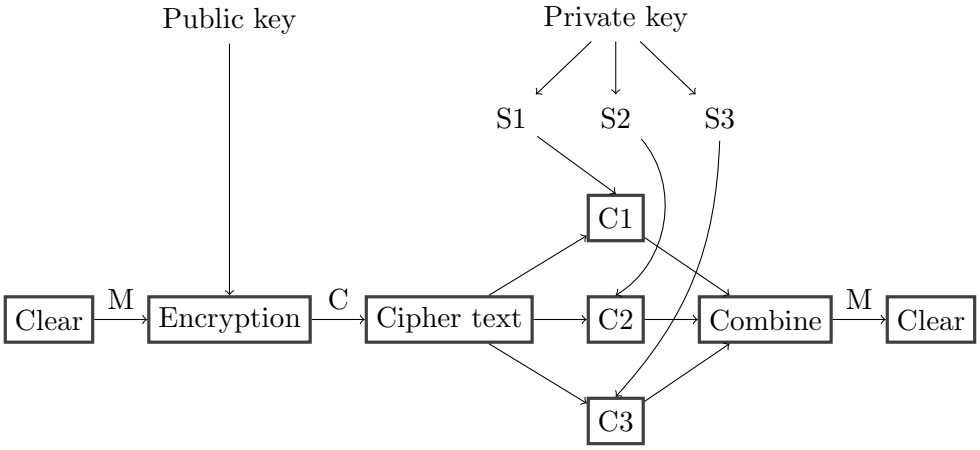

Figure 1. Encryption and decryption using a shared private key

This is not a problem for asymmetric cryptography because the encryption key can be public. This is shown in Figure 1, but for an asymmetrical key scheme such as RSA, one has to be generate a private/public key pair. This is nontrivial if no single party should get hold of the private key, but the private key should be split into shares. The public/private key pair can be computed by using a trusted third party. The trusted third party then shares the private key and publishes the public key. Again there has to be a solution so that the trusted third party really is trusted and nobody gets hold of the private key or any other information that can be used to break the public key crypto system.

Our contribution. In this *experimental work* we use multiparty computation to compute a function without the need for a trusted third party. Multiparty computation is a system whereby multiple parties can compute a common function based on private input. More specifically multiparty computation is here used to compute a public/private key pair for RSA, without any single party ever knowing the primes or the private key.

The algorithm used is based on the algorithm proposed by Boneh and Franklin [BF97], and a more detailed version by Malkin, et al. [MWB99]. Based on our experimental insights, we propose 2 *security improvements* to the algorithms, and some improvements related to the runtime. The algorithm by Algesheimer, et al. [ACS02] has not been considered, because that

algorithm switches between different integer sharing and sharing over finite fields. This switching is not easily computed in our system.

Multiparty computation was introduced by Yao in 1982 [Yao82]. The VIFF framework uses Shamir's secret sharing scheme [Sha79] and the multiplication protocol given in [BOGW88]. The VIFF framework provides a platform for computing with secret shared values.

The structure of this article. Section 2 gives an outline of the algorithm used and our improvements are discussed in Section 3. We then give an overview of the VIFF framework in Section 4, before examining the performance of the distributed protocol in Section 5. Finally Section 6 gives the conclusion and further work.

2. The Distributed RSA Protocol

Although the RSA algorithm [RSA78] is well known, a short summary will be given for completeness.

RSA key generation. Two primes p and q are chosen. The public exponent N is calculated as $N = pq$. In addition we need to compute $\phi(N) = (p-1)(q-1)$ and select an integer e, such that $gcd(\phi(N), e) = 1, 1 < e < \phi(N)$. The private exponent d is computed as $d = e^{-1} \mod \phi(N)$. The public key is the pair of values e and N and the private key is the pair of values d and N. $\phi(N)$ and the primes p and q are discarded.

Encryption and decryption. Given a plain text M such that $M < N$ the cipher text C is given as $C \equiv M^e \mod N$ and the decryption is $M \equiv C^d \mod N$.

2.1 The Distributed Protocol

The number of players contributing to the computation of the distributed algorithm is k, where $k \geq 3$. The distributed RSA protocol is taken from [BF97] and given here for completeness. The protocol can be split into five parts: First a candidate prime is generated, then this prime is tested if it divisible by a small prime, this is called trial division. This first trial division is performed on secret shared values. When 2 candidate primes have been found, the value N is revealed and trial division is performed on N. After that a biprimality test is performed on N. The biprimality test is a test to verify that N is a product of only 2 primes. Finally if all the previous tests are successful the private key is computed.

As seen from the description the value N is revealed early. This is before it is known if it is a product of two primes or not, because it is faster to compute on a publicly known value N than it is to compute on secret shared values. An overview of the distributed algorithm is shown in Figure 2.1.

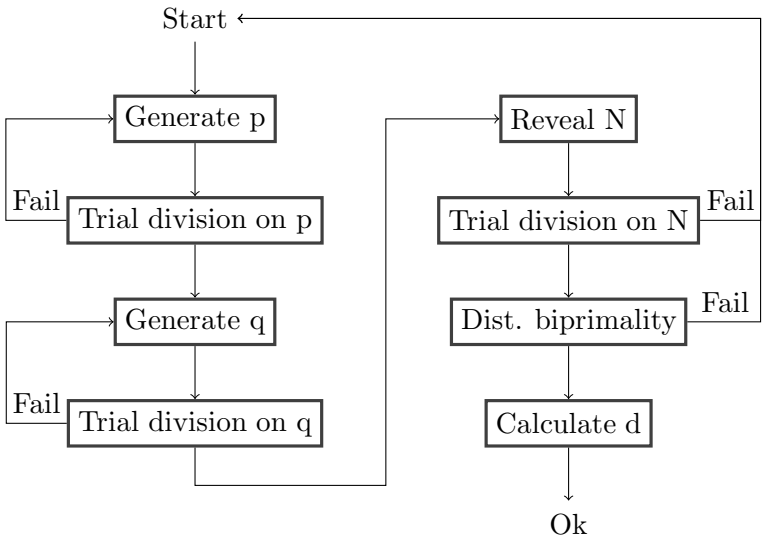

Figure 2. Distributed RSA

Generating Candidate Primes. The generating of candidate primes and first trial division is the same both primes the algorithm will only be explained for one prime called q. The distributed biprimality test only works on Blum[1] integers. Therefore player 1 chooses a random value $q_1 \equiv 3 \mod 4$ and all other players choose a random value $q_i \equiv 0 \mod 4$. These values are secret shared and the value q is computed as $q = \sum_{i=1}^{k} q_i$.

Distributed Trial Division. The parties perform distributed trial division to determine if q is divisible by a small primes less than a boundary B_1 using multiparty computations. In Boneh and Franklin [BF97] this test is done by each party picking a random value r_i in addition to q_i. Then the following products are revealed:

$$qr = (\sum_{i=1}^{k} q_i)(\sum_{i=1}^{k} r_i) \mod l \tag{1}$$

where l are all small primes less than some bound B_1. To avoid rejecting a good candidate for q because r is divisible by l the product is computed twice with two different sets of r_i's. An improvement for this test will be shown in Section 3.

Trial Division on N. When both candidate primes p and q have passed the distributed trial division step, then the value $N = (p_1 + p_2 + \ldots + p_k)(q_1 + q_2 + \ldots + q_k)$ is computed and revealed to all players. As N is the product of two large candidate primes p and q, it should not be divisible by any other

primes. The players therefore do a more comprehensive trial division on the revealed N locally to check that N is not divisible by any small prime in the domain $[B_1, B_2]$ for some fixed B_2 (typically much larger than B_1). If N is divisible by a small prime up to B_2, this test is declared a failure and the whole key generation protocol restarts by the players picking new values for the candidate primes.

Distributed Biprimality Test. After the two initial division tests, N is not divisible by any small prime number up to the boundary B_2. The next test is a distributed probabilistic biprimality test which consists of 4 steps (for proof of correctness see [BF97]).

Step 1:. The players agree on a random $g \in \mathbb{Z}_N^*$. This is done by one of the players picking a random g and revealing it to the other players.

Step 2:. The players compute the Jacobi symbol for g over N. If $(\frac{g}{N}) \neq 1$ then the distributed biprimality test is restarted with a new g.

Step 3:. The players computes $v = g^{\phi(N)/4} \mod N$. This is done by player 1 computing $v_1 = g^{(N-p_1-q_1+1)/4} \mod N$. The rest of the players compute $v_i = g^{-(pi+qi)/4} \mod N$. Next, all players secret share their values of v_i. The value v is then computed and revealed. The equation for computing v is:

$$v = \prod_{i=1}^{k} v_i \mod N \tag{2}$$

Once v is revealed, the players check if:

$$v = \pm 1 \mod N \tag{3}$$

If $v = \pm 1 \mod N$ then the test fails the parties declare that N is not a product of two distinct primes, the protocol is then restarted from the beginning by picking new values for p and q.

Step 4:. There are two ways of implementing step 4, we have used the alternative step of [BF97], this is shown below. This alternative step requires very little calculations, but there is a bit more communication between the players. This step tests if $gcd(N, p + q - 1) > 1$. The players cannot reveal their private p_i and q_i, therefore each player also picks a random number $r_i \in \mathbb{Z}_N^*$. They then compute z such that

$$z = (\sum_{i=1}^{k} r_i)(-1 + \sum_{i=1}^{k}(p_i + q_i)) \mod N \tag{4}$$

Next z is revealed, and the players check if $gcd(z, N) > 1$. If so, N is rejected, and the protocol is restarted from the beginning. If N is actually a product of two distinct prime numbers, it will pass this test with a high probability.

If N passes this test, then N is declared to be the product of two distinct primes and can be used, all that is left is then to compute the public and private exponent.

Calculate Exponents. Once N is found, the next step is to find e and d that form the public key and private key respectively. There are two options regarding the public exponent e, it can either be set to a standard (small) RSA exponent such that no calculations are required, or it can be calculated, and therefore vary from key to key. We have chosen to use a static $e = 2^{16}+1$, simplifying the computations somewhat. The calculation of $d = \sum_{i=1}^{k} d_i$ needs to be performed in a distributed manner, where each player will only learn its own d_i. Traditionally, the gcd algorithm is used to find an inverse of $e \mod \phi$, where $\phi = \phi(N)$, but as modular arithmetic is very expensive in multiparty computation, this would slow down the computations excessively. On the other hand setting $\phi_1 = N - p_1 - q_1 + 1$ and $\phi_i = -p_i - q_i$ we have the following equation:

$$\phi = \sum_{i=1}^{k} \phi_i = N - \sum_{i=1}^{k} (p_i + q_i) + 1 = N - p - q + 1 \tag{5}$$

We can then compute e using the following 3 steps:

1 Compute $\varsigma = \phi^{-1} \mod e$

2 Set $T = -\varsigma + 1$. Observe that $T \equiv 0 \mod e$.

3 Set $d = T/e$. $d = e^{-1} \mod \phi$ since $de = T$ and $T \equiv 1 \mod \phi$.

Since each player knows $l_i = \phi_i \mod e$ the players can together compute and reveal the value $l = \sum_{i=1}^{k} l_i$. The value $\varsigma = l^{-1} \mod e$ is then known. Therefore each player now can calculate:

$$d_i = \lfloor \frac{-\varsigma \phi_i}{e} \rfloor \tag{6}$$

The private key d can then be calculated as $d = (\sum_{i=1}^{k} d_i) + r$, where $0 \leq r < k$. The value r is obtained by player 1 by doing one trial decryption. This is done by player 1 first picking a message m and computing $c = m^e \mod N$. Player 1 distributes the value c to all players who return to player 1 the value $m_i = c^{d_i} \mod N$. Now all player 1 does is to compute the following for all possible values of r:

$$m = (\prod_{i=1}^{k} m_i) c^r \mod N \tag{7}$$

At last player 1 updates his value d_1 by setting $d_{1,new} = d_{1,old} + r$, for the value r that decrypted the message correctly.

The distributed RSA protocol is now complete, the values N and e are known to all players and each player has a share of the value d. Note that using a static e makes the protocol very efficient, but some bits of the key is leaked to all the players. The leakage happens when calculating $l = \phi \mod e$ and the trial decryption process where r is determined. This leaks a total of $\log_2 e + \log_2 k$ bits. This step can however be conducted such that no bits are leaked by using an arbitrary public exponent (see [BF97]) or doing the computations of d_i secret shared, so no information is revealed before after player 1 has updated his value d_1. These modifications make the protocol somewhat slower, but eliminates the leakage of information.

3. Improvements

Our first improvement is in the trial division of the primes. Instead of secret shared computing the modulus of a small prime l, we compute with shares over $\mathbb{Z}_\pi$, where π is a large prime ($\pi >> N$). If the value that we are performing trial division on is q, then each player computes and secret shares the value $t_i = q_i \mod l$. Adding the shares together results in

$$t = \sum_{i=1}^{k} t_i = sl + (q \mod l), \text{ where } 0 \leq s < k \tag{8}$$

The value u that is revealed is then:

$$u = t(t-l)(t-2l)\cdots(t-(k-1)l)r \tag{9}$$

Where r is a random value. If q is divisible by l then the revealed value u becomes 0, otherwise it will be a random value in the field $\mathbb{Z}_\pi$. This improves the run time, by ensuring that the probability of discarding a good candidate is minimal. In addition, it also improves the security in that the revealed value u does not leak any information about q.

The second weakness is found in step 4 of the biprimality test. The step checks the integers that fall into case 4 in the proof in [BF97], testing whether $gcd(N, p+q-1) > 1$, see Equation 4. For efficiency the reduction $\mod N$ must be done after z is revealed. But this leaks information about $(-1+p+q)$ if z can be factored.

To avoid this possible information leakage, we propose the following equation, which is more efficient and more secure than Equation 4.

$$z = (\sum_{i=1}^{k} r_i)(-1 + \sum_{i=1}^{k}(p_i + q_i)) + r_2 N \tag{10}$$

Here r_2 is a shared random value $0 \leq r_2 < \mathbb{Z}_\pi/N - 2kN^{\frac{3}{2}}$. The factor $r_2 N$ does not modify the value $z \mod N$, but hides the prime factors of z.

4. The Virtual Ideal Functionality Framework

VIFF [18] was started by Martin Geisler in March 2007. VIFF was started with the aim of creating a software framework for doing multiparty computations in an easy, efficient and secure manner, programmed using Python and Twisted for high flexibility. Python is a very flexible, high level programming language with support for object-oriented programming. Twisted is an event-driven network programming framework written in Python that abstracts the low-level socket communication away for the programmer, which allows the programmer to implement efficient asynchronous network applications in an easy way.

VIFF follows the modularity principle, it has a very small, well-defined core, which contains only the absolute necessary functionality for doing simple multiparty computations. On top of this core, several modules are available, which broaden the functionality of the framework, and thus making it more useful in several different scenarios.

Basically, the core in VIFF consists of 3 principles (or classes): Share, Runtime and Fields. Shares are objects that will contain share values sometime in the future. The Runtime includes logic and mathematical methods for performing arithmetic and other mathematical expressions using only shared values. The field classes ensures that values calculated in the Runtime are correct according to the actual values by doing modular arithmetics and maintaining the mapping between shared values and the real integer values.

The choice to use Python as programming language for VIFF has a good reason. Python has very good support for what is called lambda (anonymous) functions. These functions are created at runtime, and does not need to be defined elsewhere in the code. Lambda functions can schedule operations for share values before the share values actually are evaluated. When the share values used in a lambda function are ready, the lambda function is calculated. By using this method, a very efficient scheduling tree can be created, which will complete in bottom-up manner, as soon as the leaf nodes contains values, the parent node can be calculated. By using such a tree structure, VIFF can schedule many operations in parallel, which is a crucial point to be able to make secure multiparty computations efficient.

VIFF is released under the GNU LGPL license making it free to download and modify. Applications that use VIFF can also be built applications without having to license them under the GPL.

# bits	Rounds	Avg.time		Ratio	Min(s)	Max(s)
32	100	1.75 s	0.03 min	N/A	0.30	6.54
64	100	3.08 s	0.05 min	1.76	0.48	9.67
128	100	15.20 s	0.25 min	4.94	0.77	87.17
256	100	58.28 s	0.97 min	3.83	0.67	294.77
512	100	226.55 s	3.78 min	3.89	1.04	1326.16
1024	100	1956.69 s	32.61 min	8.64	7.04	8861.80
2048	10	7252.28 s	120.87 min	3.71	9.51	20713.43
4096	1	132603.92 s	36.83 h	18.28	-	-

Table 1. The time used to generate a distributed RSA key over LAN.

5. Performance Testing

5.1 Equipment

The equipment used is three computers which are connected via a 10 Mbit/s wired LAN. The specifications on the three computers are as follows:

- HP Compaq DC7900, Intel Core 2 Duo processor clocked at 3 GHz, 3.5 GB memory, Windows XP SP3.
- Dell Optiplex GX270, Intel Pentium 4 processor clocked at 2.6 GHz, 1 GB memory, Windows XP SP3.
- Dell Optiplex GX270, Intel Pentium 4 processor clocked at 2.6 GHz, 1 GB memory, Windows XP SP3.

All three computers have been used when testing over LAN, while the HP Compaq DC7900 computer has been used to test locally with all player on the same computer.

5.2 Key Generation

The key generation part measures the average time needed to generate a valid key. In this paper the average is found by performing the key generation protocol 100 times (rounds) for the smaller keys and take the average of all rounds. A smaller number of key generations have been done for for the largest keys as they are very time consuming to generate. The testing is conducted for key sizes from 32-bit to 4096-bit, using SSL on all tests. Key sizes less than 1024 bits are generally considered insecure, and testing these are purely to get an overview of the increase in time needed to generate valid keys as the keys get larger.

The results from the LAN test are presented in Table 1. The ration in column 5 is the average time divided by the average time for the previous row.

The first thing to notice from Table 1 is that the average time for generating a 1024-bit distributed RSA key over LAN is 32.6 minutes, ranging from 7

# bits	Rounds	Avg.time		Ratio	Min(s)	Max(s)
32	100	0.66 s	0.01 min	N/A	0.07	3.23
64	100	1.54 s	0.03 min	2.33	0.09	10.47
128	100	7.90 s	0.13 min	5.13	0.61	47.75
256	100	26.72 s	0.45 min	3.38	1.10	139.88
512	100	124.89 s	2.08 min	4.67	3.07	703.02
1024	100	835.48 s	13.92 min	6.69	4.94	3753.72
2048	10	6165.49 s	102.76 min	7.38	19.46	13128.69
4096	1	10431.07 s	173.85 min	1.69	-	-

Table 2. The time used to generate a distributed RSA key on one computer.

seconds as the fastest to 8861 seconds (148 minutes) as the slowest. This is quite a lot of time, and it excludes several scenarios for use of a distributed RSA key. This also implies that the time required to create a 2048-bit and a 4096-bit distributed RSA key, become impractical for some applications.

Also notice that a steady ratio of approximately 4 applies to the low length keys (up to 512 bits), this is an expected ratio when generating a distributed RSA key. The probability of a randomly picked number near N being prime is approximately $1/\ln(N)$. Doubling the number of bits in N means squaring the maximum value of both p and q. Which in turn decreases the probability of picking a prime to about half. Consequently the probability of picking 2 primes at the same time is given as:

$$\frac{1}{\ln(p)}\frac{1}{\ln(q)} \approx \frac{1}{(\ln(p))^2} \tag{11}$$

The total ratio is therefore expected to approximately decrease by a factor of 4.

The larger ratios for the larger key sizes (1024-bit and up) indicate that there are several significant reasons for the increased ratio. The calculations must be performed on larger numbers, requiring more memory, more network traffic is generated (potentially exceeding the limit of maximum packet size) and a larger $\mathbb{Z}_\pi$ must be used to be able to represent all the shared values.

The results from testing on a single computer are presented in Table 2. This test was performed on HP Compaq DC7900. While performing computations on a single computer invalidates the security of a distributed RSA key algorithm. This tests are performed to evaluate the impact of network traffic on the algorithms. These local results are much faster than the LAN tests as the communication is done locally on different ports instead of via the wired LAN.

Compared to the LAN tests, all key sizes takes approximately half the time to conduct locally instead of over the LAN. The 1024-bit key size is actually even better, performed in approximately 43% of the time needed over LAN. The ratio are reasonable close to 4 also in this test, up to key size of 1024 bits.

Function name	LAN	Local
Trial division p	39997	37152
Trial division q	39999	37137
Trial division N	6256	5812
Generating g	934	861
Checking v	467	431
Generating z	1	1
Generating d	1	1

Table 3. The average number of times each function is run

An interesting thing is the one 4096-bit key generated, was done in less than 3 hours compared to 37 hours to generate the 4096-bit key over LAN, but this only illustrates that the time consumption for such large keys can vary a lot.

The range between minimum time and maximum time in the key generation tests is quite big for all key sizes and for both LAN and locally. This is because of the time used to find both p and q as primes can vary greatly and this variance increases with key size.

Table 3 gives more details on which parts of the distributed RSA key generation protocol that are time consuming. The table gives the average number of times each of the functions in the implementation is run while generating a valid 1024-bit key.

The important thing to notice here is that the steps for key generation, up to the step for generating d, is very time consuming, and is conducted numerous times. On the other hand, once some candidates p and q have passed all the test up to the step for checking v, the rest of the steps are only conducted 1 time. This means that improvements on the run-time of the protocol should focus on the key generation step, and not so much on the step for calculating the exponents and doing decryption and signature.

The test results for decryption are shown in Table 4. Note that these results will also be valid as tests for distributed signature generation, because basically the same code is executed. The number of rounds for all key sizes is 20, which is enough to give a very good estimate for this test because the variance in each set of results is very small.

As can be seen, the times are measured in milliseconds, which means that once the key is generated, both decryption and signature can be used to more or less all possible scenarios because of the quick execution, even for large keys.

The ratio is increasing very slowly up to 1024 bits, using almost the same amount of time for 32-bit keys as for 512-bit keys. Again, the first leap is from 512 bits to 1024 bits, however this leap is not as big for decryption as for key generation. One reason for the lesser leap is that doing decryption and signature code is conducted one time only in any case, while for key generation the leap is affected by the accumulated value of many failed tries. The ratio leaps further to 2048 bits and 4096 bits increase even a bit more, but the

		LAN		Local	
# bits	Rounds	Avg.time	Ratio	Avg.time	Ratio
32	20	6.4 ms	N/A	3.3 ms	N/A
64	20	6.6 ms	1.0	3.4 ms	1.0
128	20	6.7 ms	1.0	3.4 ms	1.0
256	20	7.6 ms	1.2	4.1 ms	1.2
512	20	9.7 ms	1.3	5.0 ms	1.2
1024	20	20.2 ms	2.1	12.8 ms	2.6
2048	20	69.1 ms	3.4	53.2 ms	4.2
4096	20	560.7 ms	8.1	263.6 ms	5.0

Table 4. Decryption times

overall time needed is fairly low for all key sizes. It can also be seen that the time needed to locally compute decryption and signature is about half the time needed over LAN, which is essentially the same as was found for key generation.

Other Timing Measurements. Both [BF97] and [MWB99] state that 1024-bit keys are generated in approximately 90 seconds, this was done on much slower computers (clocked at 300 MHz) than what is used today . We will compare our results to those proposed by Malkin et al. [MWB99], as they give more data on their solution.

Their solution based on two independent components. One component abstract low level communications, and provides encrypted links between servers. This communication component was a COM package for Malkin et al. The same functionality is performed by a library called Twisted in VIFF. The second component is the high lever code. This was written in C Malkin et al., while in our program this was split between primitives given by the VIFF framework and aditional code written on top of VIFF.

The most significant difference was that Malkin et al. did not do trial division on p and q, instead they implemented a distributed sieving protocol. They reported that this single optimization resulted in a 10-fold improvement in running time and is probably most of the reason for our slower results.

This distributed sieving protocol works by first fixing the values p and q modulo 30 to ensure that the primes are not divisible by 2,3 and 5. They then computed a distributed sieving modulo $M = 7 \cdot 11 \cdot 13 \cdots p_l$, where p_l is the sieving bound.

The distributed sieving algorithm works by each player i picking a random integer r in the range $[1, \ldots, M]$. Then computing $r, r+1, r+2, \ldots, r+30$ and setting a_i to the first element that is not divisible by all the primes in M. If no element was found then a new r is chosen. The product $a = \prod_{i=1}^{k} a_i \mod M$ is then a random integer relatively prime to M. This multiplicative share is converted into an additive share so that each party obtains a value b_i

# bits	Total time MWB	Avg.time this paper	Ratio
512	0.15 min	3.78 min	25,2
1024	1.51 min	32.61 min	21,6
2048	18.13 min	120.87 min	6,67

Table 5. Comparison between MWB and our paper.

mod M. For details see [MWB99]. Finally each server picks another r_i in the range $[0, \frac{2^n}{M}]$ and sets $p_i = r_i M + b_i$.

One possible problem with their implementation of the distributed sieving algorithm is that it seems from their paper that they just calculated this sieving algorithm once for p and once for q. Then used that sieve for all candidates for p and q. This might create an attack by a player taking part in the distributed RSA key generation protocol. The attack is done on candidates for N that are not a product of 2 primes. Some of these candidates for N can be split into its prime factors. This gives information about possible values for $b_i \mod M$. Given enough candidates for N, it would be possible to get unique values for $b_i \mod M$ for the candidate N values, if the sieving algorithm was run only once. Therefore the eventual value N that would be used as a key would also have primes with known factors $b_i \mod M$.

For a comparison of the differences in timing see Table 5. The ratio is the difference between the two solutions. There was a constant factor of 20-25 between the two programs for both the 512 and 1024-bit solutions, but the factor falls to 6,67 for the 2048-bit solution. This is because the sieving time for 1024-bit solution and 2048-bits was almost the same. Therefore the sieving algorithm lost its ability to improve the solution for the 2048-bit, but our algorithm had a more constant ratio of 3.71 between the 1024-bit and 2048-bit solution. On the other hand if the sieving algorithm would have to run more times to avoid a security problem as mentioned above, the timing difference would not be so great.

6. Conclusion and Further Work

The main goal of this experimental work was to implement a fully functional distributed RSA protocol using secure multiparty computations in VIFF, and to measure its time efficiency. The distributed RSA protocol has been successfully implemented for three players in VIFF. This includes distributed key generation, decryption and signature. All of which are important features of a distributed RSA protocol. The implemented protocol allows three players to generate and use a distributed RSA key of arbitrary size in a secure manner.

Our solution is significantly slower than the solution in [MWB99], while the CPU processing speed has increased almost a 10-fold in the 10 years between the 2 implementations. The addition of a general multiparty computa-

tion library written in Python and no sieving algorithm, as opposed to special computer programs written in C and a sieving algorithm, makes the old implementation 20 times faster. On the other hand the sieving algorithm introduces a weakness in the distributed protocol if it is run only once for each prime. Therefore the 10-fold increase in speed from the sieving algorithm might not be possible with the same level of security.

Another supplementary goal of this experimental work, was to analyze the security of the original protocol. Two security weaknesses were found, both of which relate to the way a random number is used to secure a revealed answer. Both weaknesses could possibly reveal the private key to any of the players. The first weakness relates to the distributed trial division step, whereas the second weakness is regarding the alternative step in the biprimality test. Methods for avoiding both of the weaknesses have been described.

Future improvements are inspired by [MWB99]. Implementing some or all of these changes will definitely make the key generation process a lot faster, and therefore making it more useful. These have not been implemented as the implementation was completed as part of a master thesis which has constraints on the time that can be used for implementation.

- The most important improvement would be to implement the distributed sieving algorithm as in [MWB99] to improve the distributed trial division step. As they reported a 10-fold improvement in running time for this step alone when generating a 1024-bit key. It is however unclear if the same 10-fold improvements would be possible if the distributed sieving algorithm would have to be run once for each revealed value N.

- GMPY should be used to represent all the values in the VIFF program. Using GMPY instead of standard Python integers on all values in the program will greatly increase the efficiency of the protocol. This is estimated to result in a 10-20% speed up in key generation with this rather simple fix alone.

- Test several candidates in parallel by testing several values for p and q simultaneously. The nature of multiparty is not very efficient, given that the players are waiting at several synchronization points to receive shares from each other. By testing several candidates in parallel, each player normally has some calculations that can be done for at least one of the candidates, which decreases the idle time for each player, and thus improving the efficiency of the protocol.

- Implement a solution to perform parallel trial division on N. This is the idea of trying to compare N to many primes at the same time. This can be accomplished by computing $a = p_1 p_2 \cdots p_b$ for some bound b, then checking that $gcd(a, N) = 1$.

- Apply load balancing, which is the idea of balancing the calculations done for each player. The current protocol lets a specific player do some

calculation at some points, such as the calculation of the Jacobi symbol in the distributed biprimality test, which is always conducted by player 1, or the trial decryption process which is always calculated by player 3 in the implementation. The responsibility should rotate between all players, such that player i does the calculations every k time, where k is the total number of players. Applied together with testing several candidates in parallel, makes the workload for each player very uniform.

- The step for calculating the private exponent d should be implemented for arbitrary values of e, either using the method described in [BF97] or the method described in [CGH00]. This step will hardly affect the runtime for generating a valid key, but will increase the security of the protocol.

6.1 Acknowledgment

This paper is based on research done as part of the master thesis for Atle Mauland, with professor Stig Frode Mjølsnes as supervisor and PhD candidate Tord Ingolf Reistad as co-supervisor and researcher in the field of multiparty computations. The complete master thesis [Mau09] with source code will be available at http://daim.idi.ntnu.no/. We would also like to thank the Martin Geisler and the VIFF mailing list for help with the VIFF framework.

Bibliography

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *In Advances in Cryptology - Proceedings of CRYPTO 2002*, pages 417–432. Springer-Verlag, 2002.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology – CRYPTO 97*, pages 425–439. Springer-Verlag, 1997.

[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.

[CGH00] Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In *In Advances in Cryptology EUROCRYPT 2000*, pages 190–206. Springer-Verlag, 2000.

[Mau09] Atle Mauland. Realizing Distributed RSA using Secure Multiparty Computations, 2009.

[MWB99] M. Malkin, T. Wu, and D. Boneh. Experimenting with Shared Generation of RSA keys. In *In Proceedings of Symposium on Network and Distributed System Security (SNDSS*, 1999.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Tea09] VIFF Developement Team. Viff, the virtual ideal functionality framework. http://viff.dk/, 2009.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

Paper E

Internet Voting using Multiparty Computations

Md. Abdul Based, Tord Ingolf Reistad, Stig Frode Mjølsnes

At Norsk informasjonsikkerhetskonferanse 2009 (NISK 09)

Trondheim, Norway, November 24-25, 2009

INTERNET VOTING USING MULTIPARTY COMPUTATIONS

Md. Abdul Based
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
based@item.ntnu.no

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Stig Frode Mjølsnes
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
sfm@item.ntnu.no

Abstract We propose an electronic voting system where multiple independent parties cooperate to register the voter, acquire the ballots, and count the election result using multiparty computations. The aim of this system is to guarantee privacy of the voter and the ballot, public verifiability, and robustness against malicious authorities. The system is flexible in the sense that it is not bound to a single set of election rules, or public key cryptosystem.

Keywords: Internet voting, multiparty computations, voting system.

1. Introduction

Deploying an electronic voting system for general elections is becoming popular. Electronic voting systems have been used on a large scale in Brazil, India, the Netherlands, Venezuela, and the United States. Internet voting systems have also gained popularity and have been used for national elections in the United Kingdom, Estonia and Switzerland. Norway plans to test an Internet voting scheme in 2011.

This trend of electronic voting is coupled with the trend of deploying some sort of electronic identification scheme in many countries. The idea being that all interactions with the government can be handled over an Internet

connection: e-government. The idea of extending an electronic identification scheme to electronic voting use is often proposed without much thought to the security requirements of the new system.

The general concern for trusting voting systems is fundamental to acceptance. With a paper based system it can be open and intuitive for everyone. An Internet voting system is new and as such will be viewed with less confidence than a well established paper-based system.

We distinguish here between two kinds of Internet voting: polling station based voting and remote voting, say from home. The polling station based voting schemes are run like normal paper based elections except that the voting and subsequent counting are done electronically. Unless some paper based auditing trail is added to these schemes the counting process will be subject to suspicion [1]. In this paper we will focus more on voting from home.

By separating the different roles in the voting system the system can be more open to audit so that independent verifiers can ensure that no ballot was deleted or modified or added erroneously, and thereby increasing trust in the electronic voting system.

Our Contribution.. We present the design of an Internet voting scheme, where the voter will be equipped with a smartcard containing some private information. The proposed voting system splits the system into different roles in order to audit and verify that the election was performed correctly, without revealing anything about the individual voters or their choice.

The system enforces that only authorized voters can vote, and no single role can successfully cheat by removing, modifying or adding votes without it being detected. Only all the roles working together can compute the election result. The auditing system can log every message. The messages and the ballots are encrypted, so, there is no inherent security problem showing anyone the encrypted data.

The system uses multiparty computations for tallying ballots. Multiparty computations (MPC) is a computation where a number of parties $P_1, \ldots, P_n$ have *private* inputs $x_1, \ldots, x_n$ by which they compute some function f on these inputs, where $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ such that P_i learns y_i but nothing else. They are mutually mistrusting so no input is shared between the parties.

A multiparty computation based on a Shamir secret sharing scheme using a multiplication protocol [2] gives a versatile system where the counting algorithm can be tailored to different election rules [18]. The multiparty tallying will return the correct answer as long as 2/3 of the parties are honest (for active adversaries) [2].

The Structure of this Article.. Section 2 discusses this work in relation to previously published papers. The different roles of the voting system are

presented in Section 3. Section 4 describes the communication model between the parties. The protocols for ballot signing and encryption by different roles are described in Section 5, and the security analysis is done in Section 6. The limitations of the proposed system are discussed in section 7. Finally, Section 8 includes the summary of the work and future directions.

2. Background and Related Work

When it comes to actual system development, there are some systems that are developed by non-profit organizations [3, 4]. The United Kingdom [5], France, Netherlands have already partially introduced electronic voting in their areas. Estonia is the first country that introduced Internet voting for elections in March 2007 [6, 7]. A comprehensive report on the challenges and opportunities of electronic voting was issued by the Norwegian Ministry of Local Government and Regional Development in 2006 [8]. On the other hand, there is not much software freely available that can be purchased or downloaded to run a voting system electronically.

Iversen presented [10] how Zero-Knowledge Techniques could be used in computerized secret ballot election schemes. Tjøstheim worked on security analysis of electronic voting [9] in 2007. In [9], he presented a model for system-based analysis of voting systems. A voting system is presented by Chaum et al. in [11]. This system allows voters to become sure that whatever they see in the booth will be included correctly in the outcome. Mainly a rigorous and understandable model of requirements for election systems is presented in their work. They first formally state the properties of the system, and then prove them. A new verifiable and coercion-free voting scheme, namely Bingo Voting, is presented by Bohli et al. [12]. Their work is based on a trusted number generator to prevent electoral fraud, and to avoid coercion and vote buying. A variation of the Pret-a-Voter voting protocol that keeps the same ballot layout but borrows and slightly modifies the underlying cryptographic primitives from Punchscan is presented by Graaf [13]. This work is based on bit commitments. The author uses unconditionally hiding bit commitments to obtain unconditional privacy. Homomorphic elections, mixnet-based voting scheme, and electronic voting to support decision-making in local government are published in [14, 15, 16, 17].

In almost all of the above mentioned papers, the main focus was on ballot verification and tallying. Very few takes on the complete system with all roles. In this paper, we start our work with the registration of voter by a registrar, and ballot acquirer by already registered voter. We then show how the ballot is verified by the ballot acquirer, and becomes part of the counting process by the MPC talliers. Thus this work presents a voting scheme all the way from voter registration to ballot counting.

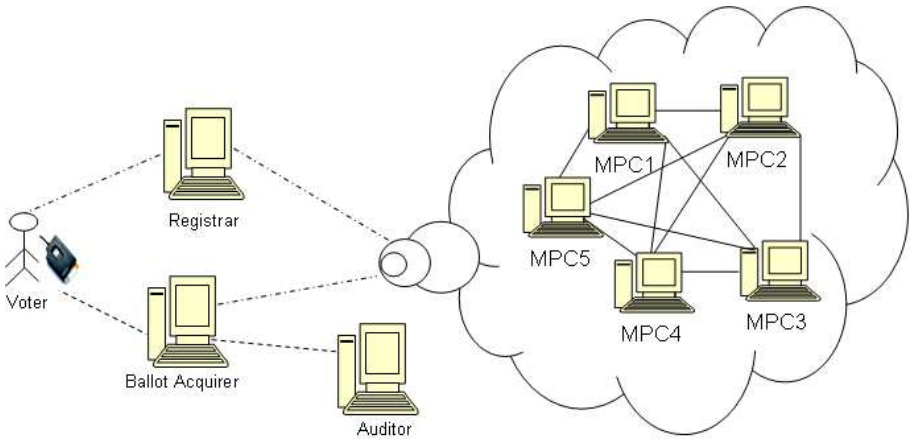

Figure 1. Internet Voting using MPC.

3. Roles in the System

The election system is organized as shown in Figure 1. The roles are as follows:

- **Voter** - A voter is one who is eligible to vote in the election. The voter holds a smartcard containing some private information that identifies the eligibility of the voter to the registrar. Each voter creates a private/public key pair and a pseudorandom identifier (*pid*). The public key of this pair and the *pid* are signed by the registrar so that the voter can use these in the voting system. When the voter has chosen the candidate to vote for, the choice is encoded, and the ballot is split into shares and sent to the ballot acquirer.

- **Registrar** - A registrar validates that the voters are eligible to vote and signs the public key supplied by the voter. It also checks that the *pid* given by the voter is unique.

- **Ballot Acquirer** - A Ballot Acquirer has the role of acquiring the ballots from the voters and checking their signature. When the ballot acquisition stage is over the ballot acquirer mixes all the votes and sends all the ballots to the MPC talliers.

  The communication could have been directly between the voter and the talliers, but adding a role that verifies that the vote is signed and acts as an anonymizing mix-net [15] makes for more distinct roles and makes the message transmission easier. It also eliminates the need for the talliers to coordinate to see that they have received the same information from each voter. Otherwise, if the voter sent the ballots directly to the talliers, the voter might send the ballots to some talliers and not to other talliers.

- **Auditor** - An auditor is an independent role that creates a public auditable record of the election system. The main role of the auditor in the proposed voting system is to keep the registrar and ballot acquirer honest so that it does not remove any ballot. That is, the auditor checks the behaviour of the ballot acquirer by monitoring all network traffic to/from the ballot acquirer.

- **MPC Talliers** - A group of talliers ($MPC_1, MPC_2, \ldots$) verifies that each individual ballot is correctly constructed and computes the result of the election using multiparty computations. In this voting system, no single tallier gets complete information about the individual votes, but together they reach a result that all talliers can agree upon.

4. Communication Model

The communication model is that of open unsecure channels between the different roles. The MPC talliers are the same role, but different parties, the communication between the them is done over authenticated and encrypted channels.

The following notation will be used: a is a message. K, K^{-1} are the public and private keys of K. $[a]_K$ is the message a encrypted with the key K. $|a|_{K^{-1}}$ is the message a and a signature of the hash of a using the private key K.

Each role is assumed to have a public key for encrypting messages to that role, the public keys are known by all roles. Each role is also assumed to have a private key to sign messages with. Although it might look like the same key is used for encryption and signing, the two keys should be different. They are just written as the same key for readability purposes.

There are many voters, but as there is no interaction between different voters. Therefore, the protocol is shown for only one voter. This voter has the public key V. The public key of the registrar is R, the public key of the ballot acquirer is A, the public key of the auditor is C, and the MPC talliers have different public keys $M_1, M_2, \ldots, M_n$, and one common key M.

5. Protocols

The following subsections will present the protocols between the roles:

5.1 Registration

Each voter will have to be registered before he or she can create a ballot in the election. The voter generates a new key public/private key pair K, K^{-1} and sends the public key K and a signed pseudorandom identifier $|pid|_{V^{-1}}$ to the registrar. (The voter first checks that the *pid* he wants to use is unique before signing it). The registrar returns $[|K, pid|_{R^{-1}}]_V$. That is, the registrar signs the key and the *pid*. This *pid* is later used in the verification of the votes. The protocol between the voter and registrar is shown in Figure 2.

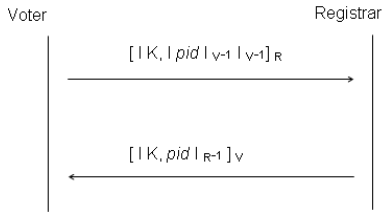

Figure 2. The Protocol between Voter and Registrar.

The registrar also sends a signed version of signed *pid* to the auditor $||pid|_{V^{-1}}|_{R^{-1}}$. This *pid* is striped of the signatures and signed by the auditor $|pid|_{C^{-1}}$ before being sent to the ballot acquirer. The reason for this is to keep the registrar from generating fake voters.

5.2 Voting

Any voting system can be implemented, but for the illustrating purpose, we will show the system where each voter first chooses his vote. In our election system, the voter is presented with a choice of candidates and gives a "one vote" to the candidate he or she chooses while giving a "zero vote" to all the other candidates. For example, if there are 4 candidates and the voter chooses candidate 3 then the vote would consist of the vector $[0, 0, 1, 0]$. To split this vector using Shamir secret sharing scheme, each element is seen as points on a polynomial. An element s would be shared using a polynomial of sufficient degree $f(x) = s + r_1x + r_2x^2 + \cdots \mod p$, where p is a large prime. The element $s = 1$ could with 5 talliers, be converted into the polynomial $f(x) = 1 + 2x + (-1)x^2$, where the coefficients 2 and -1 are chosen at random by the voter. MPC1 should receive the share $f(1) = 2$, MPC2 should receive the share $f(2) = 1$, and so on.

The voter encrypts the share by the public key of the tallier and signs it with the key K^{-1}. The *pid* is included in the vote. Thus if sh_1 is the share for MPC1, sh_2 is the share for MPC2, and so on, the ballot b should have the following form.

$$b = [pid]_M, [sh_1]_{M1}, [sh_2]_{M2}, \cdots$$

The ballot should be signed with the key K^{-1} and the key K should in turn be signed by the registrar. Therefore, the complete ballot should be on the form:

$$d = |b|_{K^{-1}}, |K, pid|_{R^{-1}}$$

Before the complete ballot is sent to the ballot acquirer it is signed by the voter and encrypted with the public key of the ballot acquirer. The message

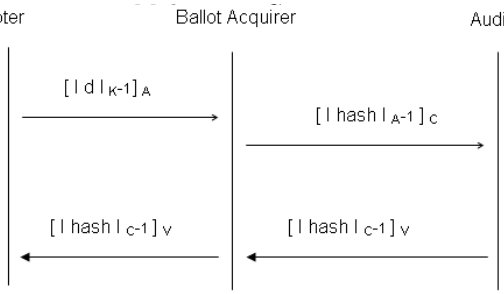

Figure 3. The Protocol between Voter, Ballot Acquirer, and Auditor.

sent from the voter to the ballot acquirer will be $[|d|_{K^{-1}}]_A$. The ballot acquirer verifies the *pid* against the list of *pid*s received from the auditor.

To verify that the ballot acquirer actually has received this ballot and does not silently drop it, the ballot acquirer sends a signed hash value of the vote to the auditor $[|hash|_{A^{-1}}]_C$. The auditor signs this hash and sends the value to the voter. The voter can therefore verify that the correct ballot was recorded by the ballot acquirer. The auditor notes how many votes were cast in the election. The auditor can also keep a correct score of how many voters have voted. The protocol between the voter, ballot acquirer, and auditor is shown in Figure 3. The auditor sends the hash to the voter through the ballot acquirer.

5.3 Sending the Ballot Batch to the MPC Talliers

The ballot acquirer waits until the election is over before mixing the ballots and sending the ballots onwards to the MPC talliers using the public key of the talliers. The ballots are signed by the ballot acquirer to verify their origin. The ballot acquirer also sends a copy of the hash values of the mixed ballots to the auditor so that the auditor can verify the hash values. This list of hash values is also verified by the talliers.

For the MPC talliers to verify that it was eligible voters, the registrar also sends some information to the MPC talliers. The information sent is the pair $[|K, pid|_{R^{-1}}]_M$. This ensures that the MPC talliers can verify that the correct *pid* was used in each ballot. This double sending will detect if the registrar or the ballot acquirer modifies the keys. The protocol between the ballot acquirer, registrar, and MPC talliers is shown in Figure 4.

5.4 Counting the Votes

The MPC talliers count the votes. First each tallier verifies the signature and sees that the key K matches the *pid*. The shares are then decrypted using the private keys M_1^{-1}, M_2^{-1}, and so on. The MPC talliers then have to see that the ballots were constructed correctly or not. As each tallier can only see one share of each ballot the talliers must work together to check the ballot.

For example, the MPC talliers should receive an array from each voter, where each element in the array is either 0 or 1, and the sum of the elements in

the array should be 1. It can be verified that this is the case without revealing anything about the vote, if the array is expressed as $b = [b_1, b_2, b_3, \ldots]$, and if the MPC talliers use the following equations:

$$\alpha_i = b_i^2 - b_i \tag{1}$$

$$\beta = \sum b_i \tag{2}$$

There is no loss of information about the ballot if α_i for all i and β are revealed. Here, $\alpha_i = 0$ for all i and $\beta = 1$, as long as the ballot is correctly constructed. The only loss of information about the ballot will occur for ballots that are not correctly constructed, but as such ballots have to be eliminated from the count there is no harm in revealing that the ballot is wrongly constructed and removing it form the tally.

6. Security Analysis

We want to achieve the following security properties of our proposed voting system:

Unlinkability. This property refers to the unlinkability between the voter identity and ballots by all parties except by the voter himself or herself. The only authorities that know the identity of the voter are the registrar and the auditor. The registrar receives public key and *pid* from the voter, but never sees the ballot sent by the voter (as shown in Figure 1 and Figure 2). So the registrar has no opportunity to link the voter identity to any ballot. The registrar only sees the *pid* and the hash of the ballots, so it has no way to combine this information.

The MPC talliers and ballot acquirer receive the ballot as $[|d|_{A^{-1}}]_M$ and $[|d|_{K^{-1}}]_A$ respectively (shown in Figure 3 and 4). This ballot is signed with the private key K. So, the MPC talliers and the ballot aquirer can not recognize the specific voter.

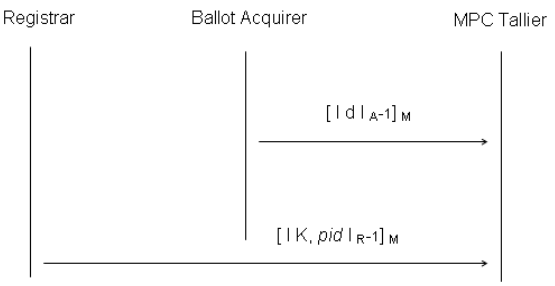

Figure 4. The Protocol between Ballot Acquirer, Registrar, and MPC Tallier.

Untraceability. Untraceabilty refers that neither the voter nor the ballot acquirer can add identifiable information to the ballot. In our proposed voting system, both of these parties sign the ballot and forward the signed ballots to the next party. This property supports the unlinkability property. Untraceability also holds for other parties in this voting system.

Only registered voters can vote. Our proposed voting system ensures that only the listed or registered voters can vote. Unless a voter receives a signed key from the registrar as shown in Figure 2, there is no way for the voter to send a ballot to the ballot acquirer that will be accepted. Thus, only the registered voters will have the opportunity to vote for a candidate.

The voter authenticates himself or herself to the registrar by generating a pair of private/public key (K, K^{-1}) and sending the public key K and a *pid* to the registrar, as shown in Figure 2. The registrar has the list of eligible voters, and it checks against its census list whether the voter is eligible or not. There is no way for a non eligible voter to vote, even the same voter cannot vote again. Because, the registrar will easily verify the signature of the voter and will detect the duplicate signature if any.

The voter signs the ballot using his/her private key value K^{-1} of the pair (K, K^{-1}), encrypts the ballot using the public key of the ballot acquirer, again signs it with private key, and sends this signed and encrypted $[|d|_{K^{-1}}]_A$ ballot to the ballot acquirer. So, the ballot acquirer can also verify the signature of the voter.

On the other hand, since the key K and *pid* are not connected to any voter, the registrar could generate keys K and *pid*s itself. This would be detected by the ballot acquirer because the *pid* would not match the list of *pid*s it received from the auditor. This is because the registrar would not be able to fake the signed *pid* value $||pid|_{V^{-1}}|_{R^{-1}}$ sent to the auditor.

The registrar could also create keys K and sign it with already registered *pid*s, but this would also be detected by the ballot acquirer if the voter voted, because there would exist two key pairs $|K, pid|_{R^{-1}}$ and $|K', pid|_{R^{-1}}$ with the same *pid*. This is something that should be stopped by the registrar, as the *pid* should be unique.

Voters can be uniquely distinguished. The registrar and the auditor maintain a list of the valid voters. The voters can either be uniquely distinguished by the key V or the signature V^{-1}. This information must not be connected to the ballots. When it comes to the ballots, the voters are also uniquely identified by the key K and the *pid*.

Only one ballot is part of the tallying process. The registered voters receive signed key with *pid* value from the registrar (shown in Figure 2). The key k and *pid* value are unique and part of the complete ballot sent to the MPC talliers by the ballot acquirer. One voter will receive only one signed key

with *pid* value. As shown in Figure 4, the registrar sends signed and encrypted K with *pid* to the MPC talliers. So, the MPC talliers can easily check this *pid* value such that only one ballot from each voter becomes part of the tallying process.

The ballot acquirer also verifies the signature of the key K and maintains a list of already verified signatures and *pid*s so that it can check the double voting by a voter.

Tallying starts after all acquired ballots have been received by the talliers. In our proposed voting system, the ballot acquirer mixes all the votes and sends all the ballots to the MPC talliers only when election is over. This ensures that tallying starts after all acquired ballots have been received by the talliers. The auditor also monitors the activities of the ballot acquirer, so the ballot acquirer has no way to drop any ballot.

No intermediate tallying result is revealed to any party. The MPC talliers verify each ballot and computes the result using multiparty computations. In this way no single tallier gets any information about the individual votes, but together they reach a result that all talliers can agree upon.

A casted ballot cannot be modified or deleted without detection. As described earlier, the auditor is monitoring the activities of the ballot acquirer. So if the ballot acquirer modifies or deletes any ballot, it will be easily detected by the auditor. Only the voter knows the key K, so only the voter can sign ballots with the key K^{-1}.

Confidentiality of the ballot. As we described earlier, the ballot acquirer receives the ballot $[|d|_{K^{-1}}]_A$ from the voter. Since the ballot is encrypted with the public key of the ballot acquirer, no one else will be able to decrypt this ballot. Moreover, as each individual share is encrypted with the MPC talliers′ public key, no one except the MPC talliers can decrypt these shares. Thus confidentiality of the ballot is achieved.

Validity of the ballot. The MPC talliers can verify the validity of the ballot. As mentioned earlier, the voter splits the ballot in some elements of a vector such that each element is either 0 or 1, and the summation of all elements of that vector is 1. Thus, in this example, MPC talliers receive a vector element b_i, the talliers check b_i^2 - $b_i = 0$. This condition holds only when b_i is either 0 or 1. The MPC talliers also check that $\sum b_i = 1$ for a valid ballot. Thus, the MPC talliers can verify the validity of a ballot without revealing any information about it.

Confirmation of vote to the voter. The ballot acquirer receives the ballot from the voter as $[|d|_{K^{-1}}]_A$. After receiving this ballot, the acquirer sends hash of this value (encrypted with the public key of the auditor) to the

auditor. The auditor signs and broadcasts it so that it can be received by the voter anonymously.

Robustness. The system presented in this paper provides no opportunity to a single role to modify or cheat a ballot. In other words, the system is 2-resilient to malicious ballot casting and t-resilient for tallying, where $t < n/3$, and n is the number of MPC talliers. The robustness of this voting system can be seen by the following:

- The voter has no chance to vote twice or a non eligible voter is not allowed to vote as described before, the voter also has no way to get two pairs of key K and *pid* signed by the registrar. (See only registered voters can vote).
- The registrar cannot send fake votes to the MPC talliers, as only the ballot acquirer sends ballots to the talliers, and the ballot acquirer signs the messages with its own private key. The registrar cannot imitate the voter (See only registered voters can vote).
- The ballot acquirer cannot send in fake votes as it is monitored by the auditor, it also does not have the private keys K^{-1} and R^{-1} needed to imitate the ballot acquirer or the voter.
- The auditor cannot silently drop *pid*s received from the registrar as this would be detected by the ballot acquirer when the voter voted. The auditor cannot fake the hash values as this would be detected by the voter.
- The MPC calculations are secure against passive security for up to 1/2 of the computations and 1/3 for active security.

Therefore there is no way for any single role or any individual tallier to cheat without being detected.

7. Limitations

The system has certain limitations as it cannot be secured against all kinds of attacks. Although the system is robust against any single role or tallier, it is not robust against 2 or more roles if they collude. For example, the registrar could issue credentials to fake voters; the *pid* is then accepted by the auditor without it being signed by a voter. This would enable the registrar to create an unlimited number of fake voters and vote for them.

Since the system focuses on voting from home, we are not considering the problem of voter coercion. The voter is identified by the smartcard, so there is no way for the system to link the smartcard to the actual voter. But, there is another problem of verifying that each voter only gets one smartcard and one key V, though this is outside the scope of this paper. We only assume that this key V comes from some sort of national identification cards.

The system is not receipt free as the voter gets a receipt from the auditor on the hash of the vote. This is because we found it more important to verify that the vote was actually received by the ballot acquirer and counted, than to provide a receipt free voting system.

We have also focused on detecting cheating by any role. We have not determined how such cheating should be handled, and how to restore confidence in the voting system if such cheating is found.

8. Conclusions and Future Work

This paper has shown a system for casting and counting votes, which provides authenticated voters to vote, provides ballot confidentiality, provides ballot integrity, ensures validity of ballot, and counts anonymous ballots without trusting any single role or tallier.

The system is versatile in that it accommodate different election rules. It also does not need expensive hardware and trusting a single server with doing the counting correctly. Thus most transactions can be logged and the logs made public, as there is no inherent security problem showing anyone the encrypted data.

The security of this system will be further analyzed. A working pilot test of this system can be constructed. The communication between voter, registrar, ballot acquirer and auditor can be realized using publicly available public key cryptosystems. The MPC Talliers can be implemented in the future using the Virtual Ideal Functionality Framework (VIFF) [18] or any other framework for multiparty computations.

Bibliography

[1] D. Sandler and D. Wallach: *Casting Votes in the Auditorium*. EVT'07: Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology, USENIX Association, (2007)

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson: *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*. Proceedings of the eighth annual ACM Symposium on Principles of distributed computing. Pages: 201 - 209, ISBN:0-89791-326-4, (1989)

[3] Web: *http://www.gnu.org/software/free/*. Access Date: September, (2009)

[4] Web: *http://www.adderpit.com/~sjaveed/coding/votebot.html*. Access Date: September, (2009)

[5] Web: *http://www.electoralcommission.org.uk/elections/pilots*. Access Date: September, (2009)

[6] Web: *http://www.foruminternet.org/telechargement/documents/reco-evote-en-20030926.pdf*. Access Date: September, (2009)

[7] European University Institute, Robert Schuman center for Advanced Studies, Report for the Council of Europe: *Internet Voting in the March 2007 Parliamentary Elections in Estonia*. July 31, (2007)

[8] Norwegian Ministry of Local Government and regional Development: *Report: Electronic voting- challenges and opportunities*. February, (2006)

[9] T. Tjøstheim, T. Peacock, and P.Y. A. Ryan: *A model for system-based analysis of voting systems*. Fifteenth International Workshop on Security Protocols, (2007)

[10] K. R. Iversen: *The Application of Cryptographic Zero-Knowledge Techniques in Computerized Secret Ballot Election Schemes*. Ph.D. dissertation, IDT-report, 1991:3, Norweigan Institute of Technology, February, (1991)

[11] D. Chaum, J. Graaf, P. Ryan, P. Vora: *High Integrity Elections*. Cryptology ePrint Archive, Report 2007/270. http://eprint.iacr.org/, (2007)

[12] J. Bohli, J. Muller-Quade, and S. Rohrich: *Bingo Voting: Secure and coercion-free voting using a trusted random number generator*, Cryptology ePrint Archive, Report 2007/162. http://eprint.iacr.org/, (2007)

[13] J. Graaf, Universidade Federal de Minas Gerais: *Merging Pret-a-Voter and PunchScan*, Cryptology ePrint Archive, Report 2007/269. http://eprint.iacr.org/, (2007)

[14] A. Acquisti: *Receipt-free homomorphic elections and write-in ballots*. Cryptology ePrint Archive, Report 2004/105. http://eprint.iacr.org/, (2004)

[15] R. Aditya, B. Lee, C. Boyd, and E. Dawson: *An efficient mixnet-based voting scheme providing receipt-freeness*. In Sokratis K. Katsikas, Javier Lopez, and Gunther Pernul, editors, TrustBus, volume 3184 of Lecture Notes in Computer Science, Pages 152-161. Springer, (2004)

[16] C. Bouras, N. Katris, V. Triantafillou: *An electronic voting service to support decision-making in local government*, Telematics and Informatics 20 (2003) 255-274, February 12, (2003)

[17] O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard and J. Stern *Practical Multi-Candidate Election System* Proceeding of the 20th ACM Symposium on Principles of Distributed Computing (PODC '01) Pages: 274-283, ACM Press, (2001)

[18] The Virtual Ideal Functionality Framework (VIFF) Web: *http://viff.dk/*.

Paper F

Linear, constant rounds Bit-decomposition

Tord Ingolf Reistad and Tomas Toft

ICITS 2009

2009

PAPER F: LINEAR CONSTANT-ROUNDS BIT-DECOMPOSITION

Tord Ingolf Reistad
Dept. of Telematics, Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tordr@item.ntnu.no

Tomas Toft
University of Aarhus, Dept. of Computer Science,
DK-8200 Aarhus N, Denmark.[*]
tomas@daimi.au.dk

Abstract When performing secure multiparty computation, tasks may often be simple or difficult depending on the representation chosen. Hence, being able to switch representation efficiently, may allow more efficient protocols.

We present a new protocol for bit-decomposition: converting a ring element $x \in \mathbb{Z}_M$ to its binary representation, $x_{(\log M)-1}, \ldots, x_0$. The protocol can be based on arbitrary secure arithmetic in $\mathbb{Z}_M$; this is achievable for Shamir shared values as well as (threshold) Paillier encrypted ones, implying solutions for both these popular MPC primitives. For additively homomorphic primitives (which is typical, and the case for both examples) the solution is constant-rounds and requires only $\mathcal{O}(\log M)$ secure ring multiplications.

The solution is only passively secure, however, active security can be achieved assuming the existence of additional primitives. These exist for both the above examples.

1. Introduction

Since Yao introduced the concept of secure multiparty computation (MPC) [Yao82] – evaluate a function on distributed, private inputs without revealing additional information on those inputs – it has been rigorously studied. Different approaches have been suggested, including garbled circuits [Yao86], secret sharing based approaches [BGW88, CCD88], and techniques relying on homomorphic, public-key cryptography, e.g. [CDN01].

[*]Supported by Simap

It has been demonstrated that any function can be evaluated by describing it as a Boolean circuit and providing the inputs in binary, e.g. stored as 0 and 1 in some field or ring over which the secure computation occurs. However, alternative representations may provide greater efficiency. If, for example, the the function consists entirely of integer addition and multiplication, this can be simulated with arithmetic over $\mathbb{Z}_M$, where M is chosen larger than the maximal result possible. With primitives providing secure computation in $\mathbb{Z}_M$ – e.g. based on Shamir sharing if M is chosen prime [Sha79, BGW88] – this is much simpler than using ring arithmetic to emulate the Boolean gates needed to "simulate" the integer computation.

Unfortunately, other operations become difficult when integers are stored as ring elements, e.g. division, modulo reduction, and exponentiation. To get the best of both worlds, a way of changing representation is needed. To go from binary to ring element is easy: it is a linear combination in the ring. The other direction is more difficult, in particular when adding the requirement that the solution must be constant-rounds.

Related work. The problem of constant-rounds bit-decomposition was first solved by Damgård *et al.* in the context of secret shared inputs, [DFK+06]; this was later improved by a constant factor by Nishide and Ohta [NO07]. Both solutions require $\mathcal{O}(\ell \log \ell)$ secure multiplications, where ℓ is the bit-length of the modulus defining the field. These solutions provided the same security guarantees as the inputs, i.e. to ensure active or adaptive security, it was sufficient to utilize secure arithmetic providing this.

Independently, Schoenmakers and Tuyls considered *practical* bit-decomposition of Paillier encrypted values, i.e. the cryptographic setting, where they obtained linear – *but non-constant-rounds* – solutions [ST06, Pai99]. (They also noted the applicability of [DFK+06] for teh Paillier based setting.) The solution of [ST06] was also secure against active adversaries, however, this needed additional "proofs of correctness" added to the basic protocol – in difference to the above solutions, secure arithmetic was not sufficient by itself.

A constant-rounds, *almost* linear solution was proposed by Toft [Tof09]. The problem of bit-decomposition was first reduced to that of postfix comparison (PFC) (using $\mathcal{O}(\ell)$ secure multiplications), which was then solved using $\mathcal{O}(\ell \cdot \log^{*(c)} \ell)$ multiplications. Similarly to [DFK+06] and [NO07], security was inherited from the primitives implying security against both active and adaptive adversaries immediately, both based on secret sharing as well on (threshold) Paillier encryption.

Contribution. We present a novel, constant-rounds, linear solution to the PFC problem, and hence also to that of bit-decomposition. The solution is applicable for arbitrary secure arithmetic in $\mathbb{Z}_M$,[1] i.e. it is applicable for both secret sharing as well as Paillier based primitives. However, in difference to [DFK+06, NO07, Tof09], perfect security cannot be provided, even if this is

guaranteed by the primitives; only statistical security is guaranteed. Further, we require $M > 2^{2(\kappa+\log n)}$ where κ is the security parameter and n the number of parties.

Similarly to [ST06], active security is *not* directly obtained from active security of the primitives. Active security is achievable when the parties can demonstrate that a provided input is less than some public bound. For both the Shamir based setting and the Paillier based setting, a constant-complexity protocol exists implying actively secure, constant-rounds, $\mathcal{O}(\ell)$ bit-decomposition protocols in these settings, where $\ell = \log M$.

An overview of this paper. Section 2 presents the model of secure computation used along with additional high-level constructs. Then in Sect. 3 the postfix comparison problem is introduced. The basic solution is presented in Sect. 4. Finally, the steps needed to achieve security against active adversaries are then discussed in Sect. 5, while Section 6 contains concluding remarks.

2. Secure Arithmetic – Notation and Primitives

We present our result based on abstract protocols. The model of secure computation is simply the arithmetic black-box (ABB) of Damgård and Nielsen [DN03]. It is described as an ideal functionality in the UC framework, and the present work can be used together with any realizing protocols. Naturally, the Paillier based protocols of [DN03] realize this functionality, but it can equally well be realized with perfect, active, and adaptive security with Shamir sharing over primes field $\mathbb{F}_M$ [Sha79] and the protocols of Ben-Or *et al.* [BGW88].

2.1 The Arithmetic Black-box

The arithmetic black-box allows a number of parties to securely store and reveal secret values of a ring, $\mathbb{Z}_M$, as well as perform arithmetic operations. Borrowing notation from secret sharing, a value, v, stored within the functionality will be written in square brackets, $[v]$. The notation, $[v]_B$ will be used to refer to a bit-decomposed value, i.e. it is shorthand for $[v_{\hat{\ell}-1}], \ldots, [v_0]$ of some bit-length $\hat{\ell}$. The ABB provides the following operations; we assume that it is realized using additively homomorphic primitives.

- **Input:** Party P may input a value $v \in \mathbb{Z}_M$. Depending on the primitives, this can mean secret share, encrypt and broadcast, etc.

- **Output:** The parties may output a stored $[v]$; following this, all parties know the now public v. This refers to reconstruction, decryption, etc.

- **Linear combination:** The parties may decide to compute a linear combination of stored values, $[\sum_i \alpha_i v_i] \leftarrow \sum_i \alpha_i [v_i]$. This follows immediately from the homomorphic property assumed above.

- **Multiplication:** The parties may compute products, $[v \cdot u] \leftarrow [v] \cdot [u]$ – this requires interaction, at least for the examples considered.

Note that secure computation is written using infix notation. Moreover, linear combinations and multiplications may be written together in larger expressions. Though further from the primitives, it improves readability as it emphasizes the intuition behind the secure computations performed.

Regarding complexity, we will only consider rounds and communication size. Note that this implies that linear combinations are considered costless. For rounds, it is assumed that the other primitives all require $\mathcal{O}(1)$ rounds, and that an arbitrary amount may be performed in parallel. Note that with abstract primitives, we can only count the number of sequential executions, not the actual number of rounds of a concrete realization. Under big-$\mathcal{O}$, the two are equivalent, though.

For communication complexity, the number of invocations of the primitives are simply counted. Moreover, rather than counting them individually, similarly to previous work they will simply be referred to collectively as *secure multiplications*. (An input from every party will be considered equivalent to a single multiplication – multiplication protocols typically require each party to provide at least one input.)

2.2 Complex Primitives

The protocols proposed will not be presented directly in the ABB model. A number of high-level primitives are simply listed – these are obtained from previous work.

Element inversion, constant-rounds multiplication, and prefix-products. Element inversion is possible using $\mathcal{O}(1)$ secure multiplications. This may be further used to obtain constant-rounds, unbounded fan-in multiplication of invertible elements. Both are due to Bar-Ilan and Beaver [BB89]. This may then be used to compute prefix-products: given an array of invertible values, $([v_0], \ldots, [v_m])$, compute $([p_0], \ldots, [p_m])$, where $[p_i] = \prod_{j=0}^{i} [v_j]$. See e.g. [DFK+06].

Random bit generation.. We require a protocol for generating a uniformly random bit which is unknown to all parties. If M is prime, this is achievable with $\mathcal{O}(1)$ secure multiplications, [DFK+06]. For non-prime M it can be achieved by letting each party share a uniformly random value in $\{1, -1\}$, computing the product of these, and mapping the result (which is still ± 1) to $\{0, 1\}$.

Note that when M is not prime, complexity is $\mathcal{O}(n)$ multiplications, where n is the number of parties. For simplicity, we disregard this factor in all complexity analyzes below. This can be viewed as assuming only a constant

number of parties. This problem is not exclusive to our work; the factor n occurs in *all* comparable solutions known to the authors.

Comparison. A protocol for comparing bit-decomposed values is required, i.e. we allow expressions of the form

$$[a > b] \leftarrow [a]_B \overset{?}{>} [b]_B.$$

A constant-rounds solution can be found in [DFK+06], complexity is $\mathcal{O}(\hat{\ell})$ multiplications in a constant number of rounds, where $\hat{\ell}$ is the bit-length of the inputs.

Non-bit-decomposed values must also be compared,

$$[a > b] \leftarrow [a] \overset{?}{>} [b].$$

This can be achieved with $\mathcal{O}(\ell)$ multiplications in $\mathcal{O}(1)$ rounds using e.g. [NO07].

Random, bit-decomposed element generation. Another requirement is the ability to generate a uniformly random, unknown element along with its bit-decomposition. This can be achieved by generating ℓ random bits and viewing these as the binary representation. Computing the value itself is a simple linear combination, while a comparison is used to verify that the value is indeed less than M. As the bits needed may be generated in parallel, this requires $\mathcal{O}(\ell)$ work in $\mathcal{O}(1)$ rounds.

Least significant bit (LSB) gate. Finally, the ability to extract the least significant bit of an $\hat{\ell}$-bit value, $[x]$, of bounded size will be needed. [ST06] describes a way to do this for Paillier encrypted values when there is sufficient "headroom" in the ring, $2^{\hat{\ell}+\kappa+\log n} < M$, where κ is a security parameter and n is the number of parties. The result is not limited to the case of Paillier encryption, but can be utilized with arbitrary realizing protocols.

The idea is that the parties initially generate a random, unknown bit m_0, and that each party P_k inputs a uniformly random $(\kappa + \hat{\ell} - 1)$-bit value, $[m^{(k)}]$ from which a random mask is computed,

$$[m] \leftarrow 2(\sum_{k=1}^{n} m^{(k)}) + [m_0].$$

Then $d = [x] + [m]$ is computed and output. By the assumption on the size of M, we have $d_0 = x_0 \oplus m_0$, where d_0 and x_0 are the least significant bits of d and x respectively. Thus, $[x_0]$ is easily obtained, $[x_0] \leftarrow d_0 + [m_0] - d_0[m_0]$.

As the ABB is secure by definition, only one potential leak exists: d. However, m_0 is unknown and uniformly random, so for any honest party, P_k,

$2 \cdot [m^{(k)}] + [m_0]$ statistically hides any information, and no adversary can learn anything, even when all but one parties are corrupt.

3. The Postfix Comparison Problem

The postfix comparison problem was introduced by Toft in [Tof09].

Problem 1 (Postfix Comparison [Tof09])**.** *Given two secret, $\hat{\ell}$-bit values $[a]_B = \left([a_{\hat{\ell}-1}], [a_{\hat{\ell}-2}], \ldots, [a_0]\right)$ and $[b]_B = \left([b_{\hat{\ell}-1}], [b_{\hat{\ell}-2}], \ldots, [b_0]\right)$, compute*

$$[c_i] = [a \bmod 2^i]_B \overset{?}{>} [b \bmod 2^i]_B$$

for all $i \in \{1, 2, \ldots, \ell\}$.

From that paper, we get the following lemma, which states that a protocol for solving the problem of bit-decomposition can be obtained from any protocol solving the PFC problem.

Lemma 1 ([Tof09])**.** *Given a constant-rounds solution to Problem 1 using $\mathcal{O}(f(\ell))$ secure multiplications, constant-rounds bit-decomposition may be achieved in complexity $\mathcal{O}(\ell + f(\ell))$.*

The proof is by construction, which we sketch, see [Tof09] for the full explanation.

To bit-decompose $[x]$, first compute $[x \bmod 2^i]$ for all i. Now, a bit, $[x_i]$, of $[x]$ can be computed using only arithmetic, $2^{-i}([x \bmod 2^{i+1}] - [x \bmod 2^i])$. To reduce $[x]$ modulo all powers of 2, first add a random, bit-decomposed mask, $[r]_B$, *over the integers*:

$$[c]_B = [x] + [r]_B.$$

Then reduce both c and r modulo 2^i (easy, as they are already decomposed) and simulate computation modulo 2^i using secure $\mathbb{Z}_M$ arithmetic:

$$[x \bmod 2^i] \leftarrow [c \bmod 2^i] - [r \bmod 2^i] + 2^i \cdot \left([r \bmod 2^i]_B \overset{?}{>} [c \bmod 2^i]_B\right).$$

The integer addition of $[x]$ and $[r]_B$ is achieved by computing and revealing $\tilde{c} = [x] + [r] \bmod M$ using ring arithmetic. We now have $c \in \{\tilde{c}, \tilde{c} + M\}$; both values are known, so it is merely a matter of securely choosing the bits of the relevant candidate. The comparisons of $r \bmod 2^i$ and $c \bmod 2^i$ for all i is a postfix comparison problem.

4. The New Constant-rounds Solution

The proposed solution is based on a a variation of a comparison protocol due to Reistad, [Rei]. The parts relevant for this paper along with the needed alterations are presented in Sect. 4.1. The improved postfix comparison protocol is then presented and analyzed in Sect. 4.2. The solution has a restriction,

which must be eliminated in order to obtain the final bit-decomposition protocol, this is described in Sect. 4.3. For the purpose of this section, assume that the parties are honest-but-curious.

4.1 The Modified Comparison of [Rei]

Let $[r]_B$ and $[c]_B$ be two $\hat{\ell}$-bit, bit-decomposed numbers to be compared. Further, let κ be a security parameter, let n be the number of parties, and assume that $2^{\hat{\ell}+\kappa+\log n} < M$. The overall idea for computing $[r > c]$ is to first compute a value, $[e_i]$, for each bit-position, i. The expression is written with intuition in mind; details on how to perform the actual computation follow below.

$$[e_i] \leftarrow [r_i](1 - [c_i])2^{\sum_{j=i+1}^{\hat{\ell}-1} [r_j] \oplus [c_j]}. \tag{1}$$

Note that $[e_i]$ is either 0 (when $[r_i]$ is not set, or when both $[r_i]$ and $[c_i]$ are set) or a distinct power of two strictly less than M – the computation can be viewed as occuring over the integers. Note also that $[e_i] = 1$ can only occur when i is the most significant differing bit-position. Thus, all values except at most one are even. And an odd value, 1, occurs only if r_i is set at the most significant differing bit-position, i.e. if $[r]_B$ is bigger than $[c]_B$.

Since at most one value is odd and this exists exactly when $[r]_B > [c]_B$, computing the least significant bit, $[E_0]$, of

$$[E] \leftarrow \sum_{i=0}^{\hat{\ell}-1} [e_i]$$

provides the desired result. In difference to [Rei], here this bit is determined with a LSB gate.

Security of this protocol is trivial: the arithmetic black-box can only leak information when something is deliberately output, and this only occurs in sub-protocols, which have already been considered.

For the computation of the $[e_i]$ above, $2^{\sum_{j=i+1}^{\hat{\ell}} [r_j] \oplus [c_j]}$ must be computed for every bit-position, i. This is done by first computing $[r_j \oplus c_j] \leftarrow [r_j] + [c_j] - [r_j][c_j]$ for each bit-position, j. Rewriting the exponentiation of Eq. (1) as

$$\prod_{j=i+1}^{\hat{\ell}-1} (1 + [r_j \oplus c_j]),$$

illustrates not only how to compute it for a single bit-position, it also allows it to be computed efficiently for *every* such position: it is simply a prefix-product with terms $1 + [r_j \oplus c_j]$ and the most significant bit-position first. This is computable in $\mathcal{O}(1)$ rounds since all terms are invertible – they are either 1 or 2, and M is odd.

4.2 Solving the PFCP with the Modified Comparison

Recall the PFC problem: we are given two $\hat{\ell}$-bit values, $[r]_B$ and $[c]_B$, and must compare all postfixes, i.e. all reduction modulo 2-powers. The above comparison cannot be applied naively at every bit-position, that would be too costly. Our goal is therefore to compute a value, $[E^{(k)}]$, for every bit-length, $k \in \{1, \ldots, \hat{\ell}\}$, equivalent to $[E]$ above. This suffices as the goal are the least significant bits of these values, and the LSB-gate requires only constant work.

Values similar to the $[e_i]$ above cannot be computed; there is a quadratic number of them. Instead the $[e_i]$ are computed as before. These are equivalent to the desired $[e_i^{(\hat{\ell}-1)}]$, and will be used in *all* the ensuing computation,

$$[\tilde{E}^{(k)}] \leftarrow \sum_{i=0}^{k-1} [e_i].$$

The computed values quite likely differ from the desired $[E^{(k)}]$. However, $[e_i]$ is only off from $[e_i^{(k)}]$ – which should have been used – by a factor of some two-power, $2^{\sum_{j=k}^{\hat{\ell}-1} [r_j] \oplus [c_j]}$. For any fixed k, this value is also fixed. Therefore $[\tilde{E}^{(k)}]$ is also simply "wrong" by a factor of this.

To "correct" $[\tilde{E}^{(k)}]$, first note that $[2^{\sum_{j=k}^{\hat{\ell}-1} r_j \oplus c_j}]$ has already been computed by the prefix-product. Further, the factor can be eliminated as it is invertible and element inversion is possible due to the protocol of [BB89]. I.e. the desired $[E^{(k)}]$ is securely computable.

$$[E^{(k)}] \leftarrow [\tilde{E}^{(k)}] \cdot [2^{\sum_{j=k}^{\hat{\ell}-1} r_j \oplus c_j}]^{-1}$$

At this point, invoking an LSB-gate on every $[E^{(k)}]$ provides the final result.

Correctness follows from the above discussion along with that of Sect. 4.1. Regarding security, the protocol clearly does not leak information. More values are output from the ABB, but this still occurs only in sub-protocols. Thus, no information is leaked.

We conclude with a complexity analysis of the protocol. Securely computing both the factors, $[2^{\sum_{j=k}^{\hat{\ell}-1} [r_j] \oplus [c_j]}]$, and the $[e_i]$ for all bit-positions, i, and bit-lengts, k, requires only $\mathcal{O}(\hat{\ell})$ secure multiplications in $\mathcal{O}(1)$ rounds. All that was required was the computation of $[r_j \oplus c_j]$ for every bit-position, the prefix-product, and the concluding computation for each $[e_i]$.

Computing the $[\tilde{E}^{(k)}]$ is costless at this point, while correcting them – computing the $[E^{(k)}]$ – requires $\mathcal{O}(\hat{\ell})$ work. One element inversion and one multiplication is needed per bit-length, and these may be processed in parallel. Similarly, the concluding LSB-gates are also $\mathcal{O}(1)$ work each and may also be executed concurrently.

Combining the above, it is clear that only $\mathcal{O}(1)$ rounds are needed in which $\mathcal{O}(\hat{\ell})$ secure multiplications must be performed. Thus, the following theorem is obtained.

Theorem 1. *There exists a protocol which solves postfix comparison problems of size $\hat{\ell}$ in $\mathcal{O}(1)$ rounds using $\mathcal{O}(\hat{\ell})$ secure multiplications of elements of $\mathbb{Z}_M$, for $M > 2^{\hat{\ell}+\kappa+\log n}$.*

4.3 Performing Bit-decomposition

It remains to apply Lemma 1 and Theorem 1 to obtain the main result of this paper. There is, however, still one problem to be solved. The PFC problem to solve is of size $\ell = \log M$, but to apply Theorem 1 we must have $M > 2^{\ell+\kappa+\log n}$; this is of course contradictory.

Assuming that $M > 2^{2(\kappa+\log n)}$, then the following variation of the above solution fixes the problem. The trick, taken from [Rei], consists of considering pairs of bit-positions rather than single bit-positions when computing the $[e_i]$. This results in half as many $[e_i]$, thereby halving the bit-length needed. I.e. the resulting modified $[E^{(k)}]$ have at least $\kappa+\log n$ bits of headroom in $\mathbb{Z}_M$, allowing the LSB-gate to be applied. This was the sole reason for the restriction on M. For simplicity it is assumed that ℓ is even in the following, where Eq. (1) is replaced by Eq. (2) which is computed only for the *odd* bit-positions, i.

First values, $[u_i]$, are computed,

$$[u_i] \leftarrow [r_i] \wedge (\neg[c_i]) \vee (\neg([r_i] \oplus [c_i])) \wedge [r_{i-1}] \wedge (\neg[c_{i-1}]).$$

Note that this is simply a comparison circuit for 2-bit numbers. Though somewhat complex, the expression translates readily to arithmetic.

$$[r_i](1-[c_i]) + (1 + [r_i] \cdot [c_i] - [r_i] - [c_i])[r_{i-1}](1-[c_{i-1}])$$

The $[u_i]$ are then used in the computation replacing Eq. (1). $[e'_i]$ is set to a 2-power exactly when the 2-bit position of $[r]_B$ is greater than that position of $[c]_B$, and the powers are smaller, as only the number of differing 2-bit blocks are "counted."

$$[e'_i] \leftarrow [u_i] \cdot 2^{\sum_{j=((i-1)/2)+1}^{\ell/2-1}([r_{2j}]\oplus[c_{2j}])\vee([r_{2j+1}]\oplus[c_{2j+1}])} \tag{2}$$

Again, the expression is slightly more involved than before, but it can also be translated to a prefix-product.

The smaller $[\tilde{E}^{(k)}]$ can now be computed, however, there are two cases. To compute the odd ones, values $[e'_i]$ are also needed for the even bit-positions.

$$[e'_i] \leftarrow [r_i](1-[c_i]) \cdot 2^{\sum_{j=(i/2)+1}^{\ell/2-1}([r_{2j}]\oplus[c_{2j}])\vee([r_{2j+1}]\oplus[c_{2j+1}])}$$

At this point we may compute

$$[\tilde{E}^{(k)}] \leftarrow \begin{cases} \sum_{i=0}^{k/2-1} [e'_{2i+1}] & \text{when } k \text{ is even} \\ [e'_{k-1}] + \sum_{i=0}^{(k-1)/2-1} [e'_{2i+1}] & \text{when } k \text{ is odd} \end{cases}$$

after which the incorrect powers of 2 can be eliminated and the LSB-gates applied as above. This solves the PFC problem such that $[x]_B$ can be determined. The result is summarized in the following theorem.

Theorem 2. *There exists a protocol which bit-decomposes a secret value, $[x]$ of $\mathbb{Z}_M$, to $[x]_B$ using $\mathcal{O}(\ell)$ secure multiplications in $\mathcal{O}(1)$ rounds. The protocol is statistically secure against passive adversaries when the arithmetic primitives are this. When they only provide computational security, then so does the present protocol.*

5. Active Security

As noted in the introduction, active security is not immediate. Even when actively secure protocols are used for the computation, problems occur when the parties are asked to share a random value from a domain different from $\mathbb{Z}_M$. For example, when M is not a prime, the parties must verify that inputs are really ± 1 during the bit-generation protocol. This is easily achieved with a zero-knowledge proof in the case of Paillier values, [DJ01]. The problem does not affect the solution based on Shamir sharing. There $\mathbb{Z}_M$ must be a field which implies that M is a prime. A second problem occurs in the LSB gates. It must be verified that the masks, $[m^{(k)}]$, are indeed of the specified bit-length. But these are the only problems.

By the definition of the arithmetic black-box, no adversary can do other harm. Thus, given a constant-work means of proving that a value is of bounded size (an interval proof), the solution can be made secure against active adversaries. There exists such proofs for both our examples.

As noted in [ST06], it is possible to demonstrate that a Paillier encryption contains a value within a specified range using the results of [Bou00, Lip03, DJ02]. The solution reveals no other information than the fact that indeed the value was from the desired range. Hence, active security is quite easily obtained in a Paillier based setting.

Regarding protocols based on Shamir sharing, such a proof is not immediate. It is, however, possible to obtain efficient range proofs by taking a detour through linear integer secret sharing (LISS). The solution follows directly from LISS, hence we only sketch it; see [Tho09] for a full explanation.

First off, LISS not only provides secret sharing of integer values, it can also form the basis for unconditionally and actively secure MPC. Further, it is possible convert a linear integer secret sharing to a Shamir sharing over $\mathbb{Z}_M$ simply by reducing the individual share modulo M. The solution is therefore to first share the $m^{(k)}$ using LISS, and for each of them demonstrate that it

Table 1. Complexity of constant-rounds bit-decomposition.

	Rounds	**Multiplications**
[DFK+06]	38	$94\ell \log \ell + 63\ell + 30\sqrt{\ell}$
[NO07]	25	$47\ell \log \ell + 63\ell + 30\sqrt{\ell}$
[Tof09]	$23 + c$	$(31 + 26c)\ell \cdot \log^{*(c)} \ell + 71\ell + 14c\sqrt{\ell}\log^{*(c)}(\ell) + 30\sqrt{\ell}$
This paper	18	$74\ell + 30\sqrt{\ell}$

is in the desired range using constant work, [Tho09]. Secondly, those secret sharings are then converted to Shamir sharings over $\mathbb{Z}_M$. This ensures that the Shamir shared value is of bounded size as required.

6. Conclusion

We have proposed a novel protocol for constant-rounds bit-decomposition based on secure arithmetic with improved theoretic complexity compared to previous solutions. The complexity reached – $\mathcal{O}(\ell)$ – appears optimal, as this is also the number of outputs, however, that this is the case is not immediately clear. Proving that $\Omega(\ell)$ secure multiplications is indeed a lower bound is left as an open problem.

Unfortunately, the present solution also has some "defects" compared to the previous ones. Firstly, "only" (at most) statistical security is guaranteed, rather than perfect. This still allows linear, constant-rounds, unconditionally secure bit-decomposition, though. That security cannot be perfect also implies that the underlying ring must be sufficiently large to accommodate the large, random elements needed for statistical security. I.e. the protocol is only applicable for large moduli.

A second, worse "defect" is that the basic solution does not provide out-of-the-box active security. This must be obtained through additional protocols, which of course increases complexity of the operations where these are needed. However as demonstrated, efficient, active security can be achieved quite readily for both Paillier based and Shamir sharing based settings.

We conclude by comparing the explicit complexity of our solution to that of previous ones, Table 1. Counting the exact number of secure multiplications provides a direct comparison for the case of passive security. It is noted that the proposed protocol not only improves theoretic complexity, it is also highly competitive with regard to the constants involved.

With regard to active security, however, the present solution must also take into account the proofs that the masks shared by parties are well-formed – these depend on the realizing primitives and are therefore not included in the overview.

Bibliography

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Piotr Rudnicki, editor, *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209, New York, 1989. ACM Press.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant dist ributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, 1988.

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807, pages 431–444, 2000.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19. ACM Press, 1988.

[CDN01] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045, pages 280–300, 2001.

[DFK+06] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryoptography*, volume 3876 of *mylncs*, pages 285–304. Springer, 2006.

[DJ01] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992, pages 119–136, 2001.

[DJ02] I. Damgård and M. Jurik. Client/server tradeoffs for online elections. In David Naccache and Pascal Paillier, editors, *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2274, pages 125–140, 2002.

[DN03] I. Damgård and J. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer Berlin / Heidelberg, 2003.

[Lip03] H. Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, volume 2894, pages 398–415, 2003.

[NO07] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC 2007: 10th International Workshop on Theory and Practice in Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2007.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, Berlin, Germany, 1999.

[Rei] T. Reistad. Multiparty comparison – an improved multiparty protocol for comparison of secret-shared values. To appear at SECRYPT 2009.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[ST06] B. Schoenmakers and P. Tuyls. Efficient binary conversion for paillier encrypted values. In *Advances in Cryptology – EUROCRYPT 2006*, Lecture Notes in Computer Science, pages 522–537. Springer-Verlag, Berlin, Germany, 2006.

[Tho09] R. Thorbek. *Linear Integer Secret Sharing*. PhD thesis, Aarhus University, 2009.

[Tof09] T. Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473, pages 357–371, 2009.

[Yao82] A. Yao. Protocols for secure computations (extended abstract). In *23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society Press, 1982.

[Yao86] A. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.