.
Aalto University
School of Science
Degree Programme in Security and Mobile Computing

Reimo Rebane

A Feasibility Analysis of Secure Multiparty Computation Deployments

Master's Thesis
Espoo, May 16, 2012

Supervisors: Professor Tuomas Aura, Aalto University
Sven Laur, Ph.D., University of Tartu
Instructor: Dan Bogdanov M.Sc., University of Tartu

Aalto University
School of Science

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF
MASTER'S THESIS

Author:	Reimo Rebane		
Title:	A Feasibility Analysis of Secure Multiparty Computation Deployments		
Date:	May 16, 2012	**Pages:**	59
Professorship:	Computer Science	**Code:**	T-110
Supervisors:	Professor Tuomas Aura Sven Laur, Ph.D.		
Instructor:	Dan Bogdanov M.Sc.		

Imagine a scenario where multiple companies hold valuable information and they want to combine their data for analysis that would benefit them all. In an honest world, the companies could do just that—combine their data. However, in the real world, they might not be able to share their data because of data privacy issues. A cryptographic solution to this problem would be to use secure multiparty computation (SMC). SMC is a useful tool for computing the result of an operation with the inputs of multiple parties, without revealing what the inputs were. As a result, we can perform computations on the data without disclosing it.

General multiparty computation is communication heavy and therefore its performance is network bound. One goal of this work is to create a mathematical model for predicting the performance of SMC protocols depending on the network parameters. The model is based on a set of experiments performed on the SHAREMIND SMC framework in our specialized cluster system.

We perform analysis of the constructed model and estimate the model parameters. To validate the model, we compare the predictions of the model with the actual algorithm execution time results on the cluster system. To see how the model performs on an alternative system, we deploy a SHAREMIND nodes in the cloud environment and perform the validation there.

In the last part of the work, we assess the feasibility of SMC in the cloud environment. The analysis is based on a sample secure survey scenario.

Keywords:	secure multiparty computation, performance modeling, cloud deployments
Language:	English

Acknowledgements

I would like to thank my supervisors and instructor for the guidance and feedback provided while writing this thesis. Their help was invaluable in constructing the performance model and analyzing it. Additionally, all the suggestions on the general direction of the work helped me to stay focused.

I would especially like to thank Dan Bogdanov for all the help regarding with the SHAREMIND framework, understanding how its protocols are structured and for providing me the tools and resources for performing this work.

Tartu, May 16, 2012

Reimo Rebane

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem statement

Imagine a scenario where multiple companies hold valuable information and they want to combine their data for analysis that would benefit them all. In an honest world, the companies could do just that—combine their data. However, in the real world, none of them can afford to make their data public because it could compromise their competitive advantage. One can easily find many similar real-world scenarios where there are privacy issues concerned with the data. Data privacy is also a very prominent issue when outsourcing the computing resources, for example, to cloud services. A cryptographic solution to this problem would be to use secure multiparty computation (SMC). SMC is a useful tool for computing the result of an operation with the inputs of multiple parties, without revealing what the inputs were. As a result, we can perform computations on the data without disclosing it.

There exist two main approaches how to perform SMC. First, circuit evaluation, which is based on computations on arithmetic or logic circuits and is CPU intensive (CPU-bound). Second, general multiparty computation, which relies more on the communication between the parties (network-bound). Currently, the more efficient systems in this field use the latter approach. The theoretical complexity of these systems is well known. However, for real-life deployments the theoretical results alone are not enough. In this work, we would like to study the practical performance of the network-bound general multiparty computation. Based on the published results, the SHAREMIND [5] SMC framework has shown the best performance and widest functionality among similar systems. Our results will be based on the SHAREMIND framework.

Modern cloud service providers allow for quick deployment of services on the web through resource virtualization. Virtualization supports the easy allocation of hardware resources. For the service provider, it grants better utilization of the resources, for the customers, it enables payment only for used resources and quick scaling. For many po-

tential services on the cloud, the data privacy issues could be solved with SMC. However, little research has been done about the feasibility of such deployments in the cloud environment.

One goal of this work is to create a mathematical model for predicting computational performance of SMC depending on the network parameters. The main parameters we are interested in are the bandwidth and latency of the network we operate on. The model is based on a set of experiments that are performed on the SHAREMIND framework. This model will be used to predict the resource usage of an SMC implementation in the cloud environment and therefore get an estimation of the cost of such deployment.

Furthermore, a feasibility analysis is conducted using a sample scenario, where a securely survey is conducted with reasonable volumes of data and amount of computation. Based on the sample scenario we can estimate the feasibility of a wide range of other scenarios.

1.2 Outline

In Section 2, we take a brief look at the background information required for reading the rest of the work. An overview of secure multiparty computation is given, after which we descibe a concrete SMC framework, namely SHAREMIND, and discuss how its protocols are structured. We cover the related work at the end of this section.

Section 3 describes our experimental setting. We cover the hardware and software we used, what was process of running our experiments and, finally, what kinds of data was gathered and how we gathered it.

In Section 4, we discuss the protocols in the SHAREMIND framework more thoroughly by dissecting some of the protocols, granting a better understanding on what has to be taken into account when constructing the performance models.

Section 5 presents the constructed protocol performance model. In this section we explain the reasoning behind the model, the experiments that were run to gather the data for analyzing this model and validate how well the model fits the experimental data.

In Section 6, we motivate the benefits of performing SMC in the cloud environment, set up a SHAREMIND installation in the cloud and see how well our model predicts the protocol performance in this setting.

Finally, Section 7 analyzes the economic feasibility of performing SMC in the cloud environment. We estimate the costs of a sample secure survey scenario.

1.3 Author's contribution

In this section we list the author's contributions to this work. In performing the experiments with the SHAREMIND framework, the author was responsible for setting up the

nodes in the cluster environment. The author did the scripting work related to gathering and aggregating the results. Even though we partially automated our experiments, it can still be noted that we run an approximate rough total amount of 1000 hours of experiments on our cluster. Some of the existing data gathering tools in SHAREMIND were extended by the author to allow for more precise statistics about the network.

In the model analysis, the author performed various tasks such as fitting the model to the gathered data and calculating the communication coefficients depending on the assumed network connection. Estimating the accuracy of the fit and the significance of the model coefficients, was also carried out by the author in the model validation process.

As with the cluster machines, the author was responsible for setting up the SHAREMIND configuration in the cloud environment. This required some research on the existing cloud service providers and their pricing models, available server configurations, data center locations and other aspects affecting the choice of the provider. The author also managed the experiments in the cloud and executed the validation of our model on the data gathered in these experiments.

Finally, in the feasibility analysis the author estimated the cost of performing certain operations in the cloud environment. An evaluation of the feasibility of SHAREMIND deployments in the cloud was made based on the estimates.

Chapter 2

Preliminaries

2.1 Secure multiparty computation

Secure multiparty computation is a method that allows multiple parties to compute the output of a function in a secure manner. Secure, in this context, means that the inputs and outputs remain private and that the correctness of the output is guaranteed. The security properties must hold even if some parties cheat. Additional security requirements might apply to certain applications of SMC.

More formally, parties $P_1, \ldots, P_n$ can compute any function f such that $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where x_i corresponds to the input and y_i to the output of party P_i. Each party only learns its input and output and nothing more (except what is implied by the computed function). The outputs $y_1, \ldots, y_n$ can be different for some functions and equivalent for others. The concept of SMC was first introduced by Yao [16] in 1982 as a solution to the Millionaires' problem, where two parties want to know who is richer without revealing the exact amount of their wealth.

Many secure multiparty computation schemes make extensive use of secret sharing. Secret sharing is a scheme for splitting a secret value between a number of parties, also called *agents*. The scheme works by having a party called the *dealer*, knowing the secret value, construct shares of the secret and privately distribute them to the *agents*. The secret can only be recovered if a qualified subset of *agents* combine their shares, otherwise no information is revealed about the secret. Secret sharing was independently proposed by Shamir [15] and Blakley [3] in 1979. Both solutions use a threshold scheme where a secret s, shared between any number of n parties $P_1, \ldots, P_n$, can be reconstructed by any subset of t parties, where $t \leq n$. We denote the shares of the secret s as $[\![s]\!] = [s_1, \cdots, s_n]$, where party P_i knows the share s_i. Homomorphic encryption can be used as an alternative to secret sharing when constructing SMC protocols. However, the secret sharing based protocols are currently shown to be more efficient [13].

To argue about the security of a SMC protocol, we have to model the capabilities

of an adversary. It is common to consider an adversary who can corrupt a number of parties [11]. The corruption process can be *static*, where the adversary corrupts a fixed set of parties chosen before the protocol starts, or *adaptive*, where the adversary can corrupt additional parties during the protocol based on the information received. The behavior of the corrupt parties during the protocol can be modeled in the *semi-honest* or in the *malicious* security model. In the *semi-honest* model, also known as *honest-but-curious* model, the adversary sees the entire view of the corrupted parties, but the corruption is passive and the corrupted parties must still follow the protocol. The system has to be secure against the combined knowledge of the corrupted parties. In the *malicious* model, the adversary, as before, sees the view of the corrupted parties. However, there are no restrictions to the behavior of the corrupted parties (they can send arbitrary messages or stop communication altogether).

We can also classify protocols based on the communication model used [11]. In the cryptographic model, the adversary can see all the messages sent over a channel, however, messages exchanged between honest parties cannot be modified. The security is based on a hardness assumption, that some operations are difficult to perform (for example the discrete logarithm assumption in a cyclic group). Alternatively, in the information-theoretic model, we assume pairwise secure channels, where the adversary can only see that some message was sent on the channel, without learning anything about the contents. Communication can be done in both synchronous and asynchronous way, although more restrictions apply for the latter case.

Similarly to the SMC protocols, we can look at secret sharing in the presence of a *semi-honest* or *malicious* adversary, where the adversary can corrupt a number of parties and tries to obtain the secret [10]. In the *semi-honest* model, a threshold secret sharing scheme is secure with at most $t - 1$ corrupted parties, where t is the threshold. In the *malicious* model, a stronger scheme is needed, because we have no control over the actions of the corrupted parties. Verifiable secret sharing was introduced by Chor, Goldwasser, Micali and Awerbuch [9]. It ensures that if a corrupt party provides an invalid share while reconstructing the secret, it is detected and the secret can still be reconstructed with simulating the corrupt party by the honest parties. VSS is secure against an adversary that has corrupted at most t parties, where $t - 1 < n/2$.

The classical results by Ben-Or, Goldwasser and Wigderson [2] and Chaum, Crépeau and Damgård [8] state that every function can be securely computed in the presence of an adaptive adversary if and only if the adversary has corrupted less than $n/2$ parties in the *semi-honest* model and less than $n/3$ parties in the *malicious* model. Better threshold can be obtained with additional assumptions (for example broadcast channels) [11].

2.2 Sharemind

SHAREMIND is a framework for performing computations on private data using SMC. It works in a client-server model, where the controller applications act as clients and SHAREMIND is the distributed application server. The controller applications can be implemented in multiple ways, for example as web applications or as dedicated desktop applications. The controller applications perform two tasks. First, they insert data to the distributed server by splitting the data into individual shares, using secret sharing, and transmitting the shares to the server nodes. Second, they request permitted computations on the inserted data and retrieve the results. The distributed server, in a typical deployment setting, runs on three separate machines, called (data) miners or computing nodes. These three machines perform the multiparty computation protocols, using the data shares. A high-level structure of the system in a deployment setting is illustrated in Figure 2.1.

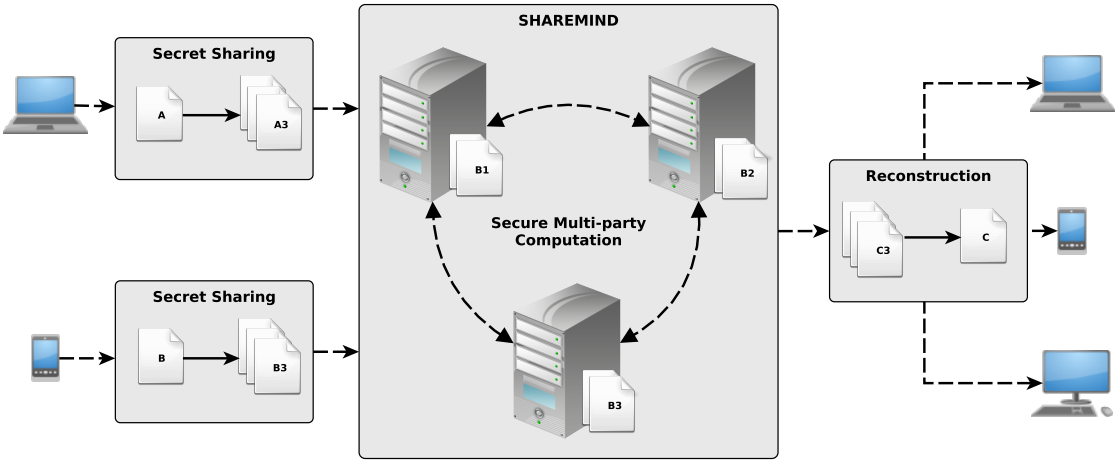


Figure 2.1: Deployment model of the SHAREMIND framework

The current version of SHAREMIND uses an n-out-of-n additive secret sharing scheme with 32-bit unsigned integer values, or more concretely, values in $\mathbb{Z}_{2^{32}}$. The shares $s_1, \ldots, s_n$ are constructed from the secret s by uniformly generating $s_1, \ldots, s_{n-1} \in \mathbb{Z}_{2^{32}}$ and computing $s_n = s - s_1 - \cdots - s_{n-1}$. The secret is reconstructed by adding the shares together in $\mathbb{Z}_{2^{32}}$. In SHAREMIND, the data is divided into three shares. This scheme has been shown to be information-theoretically secure.

The SMC protocols for the secure operations, that are implemented currently implemented in SHAREMIND, are listed in Table 2.1—values referred to as private are in practice secret shared. These basic operations can be combined to compute more complex functions. The implementation of algorithms is made possible for non-cryptographers

through the use of a C-like language called SECREC [12].

SHAREMIND is secure in the *semi-honest* model with an *adaptive* adversary and at most one corrupted party. For the security guarantees to hold for real-life implementations, it is assumed that each miner node is controlled by a different organization and that there is sufficient motivation for the organizations to keep their data secret and thus not to cooperate in a malicious way. This can be achieved with legal contracts between the organizations.

Each of the computing nodes of the distributed server has a private and authenticated channel for communication with the other two nodes. The same applies for the communication between the controller applications and the miners.

SHAREMIND is programmed in C++ and runs on all of the major platforms of Windows, Mac OS X and Linux. The RakNet[1] network engine is used as the network layer of SHAREMIND. It uses UDP Data Transfer protocol for data communications. In this work, we are using the major version 2 of the framework.

Label	Operation	Description
Add	Addition	Add two private values
BitAdd	Bitwise addition	Bitwise add two private values
BitExtr	Bit share extraction	Extract all $\mathbb{Z}_2$ private bits from a $\mathbb{Z}_{2^{32}}$ private value
Div	Division	Divide a private value with another one
Equal	Equal comparison	Compare if two private values are equal
Mult	Multiplication	Multiply two private values
PubMult	Multiplication by public scalar	Multiply a private value with a public one
PubDiv	Division by public scalar	Divide a private value with a public one
ShareConv	Share conversion	Convert a $\mathbb{Z}_2$ private value to a $\mathbb{Z}_{2^{32}}$ private value
ShiftR[2]	Greater-than comparison	Compare if a private value is greater than another

Table 2.1: Existing SHAREMIND protocols for basic operations

[1]RakNet, `http://www.jenkinssoftware.com`. Last accessed: March 23rd 2012

[2]The ShiftR is a bit shift right protocol, which is used to implement the greater-than comparison.

2.3 Secure computation protocols

The protocols in SHAREMIND perform basic arithmetic operations on the secret shared values such that at the end of the execution, the result of the operation will be secret shared among the miner nodes. The shared inputs and the outputs of the operation will not be revealed in the process. The execution of the protocols is done in a synchronous manner, such that receiving messages for the computing nodes is blocking. The order of the messages in the communication channel between the nodes is preserved. This synchronized communication and also the protocol structure itself enforces the protocols to proceed in rounds. A node might need some input from the other nodes and cannot continue in executing a protocol before that information has been received.

For better performance, the protocols are vectorized, meaning that we perform the operations on the elements of large vectors in parallel, resulting in a faster per-element execution speed. The maximum size of the vectors processed at once is configurable by the batch size parameter. If the input vectors are too large, they will be processed in smaller chunks, defined by the parameter. The batch size has to be chosen depending on the network characteristics. If the batch size is chosen too large, excessive idle time is introduced because the nodes are waiting behind the network. However, if it is too small, the overhead in the network increases and sending the data takes more time.

The performance of the vectorized protocols grows with the input vector sizes. However, from a certain input size onwards, at a *saturation point*, the per-element computation time will stay near constant. At this point the network becomes saturated and increasing the input size will grow the running time of the protocol linearly. Since the communication complexity between protocols differs, the *saturation point* can also be found at different vector sizes. Note that, in theory, SHAREMIND would benefit from dynamically calibrating its network related parameters, based on the network parameters and the scheduled computations, to find the optimal settings for the protocols.

Resharing is performed on some of the inputs and outputs of the SHAREMIND protocols to guarantee the independence of the output from its inputs. The shares $[\![u]\!]$ are reshared as $[\![v]\!]$, such that the following conditions hold: $u = v$, all shares v_i are uniformly distributed and u_i and v_j are independent for all pairs of i, j.

Detailed examples of some of the protocols used in SHAREMIND can be found in Section 4. For additional information about the protocols, refer to [1, 5, 6].

2.4 Related Work

According to our knowledge, there are only a few research publications that assess the practical feasibility of SMC. A paper by Bogetoft et al. [7] discusses a prototype implementation of SMC in a secure auction setting. In this setting the auctioneer is a *trusted third party* consisting of n distributed machines processing the private input

of the clients. In the measurements the machines were connected in a LAN setting, except for one, which was connected through an ADSL Internet connection over a VPN connection. The performance of this system is evaluated for a differing number of *trusted third party* machines. The paper concludes that the comparison protocols and the implementation are feasible for real-world double auction applications.

Another paper by Kerchbaum et al. [13] compares the performance of two implementations of the greater-than-or-equal comparison protocol. One implementation is based on homomorphic encryption, while the other one uses secret sharing. They run performance benchmarks for both of the protocols with differing number of parties and conclude that the secret shares based protocol outperforms the homomorphic one. The experiments are run locally, simulating the parties on a single machine. The impact of the network communication on the performance of the protocols is not studied in the paper.

Chapter 3

Experimental methodology

3.1 Experimental setting

3.1.1 Hardware and software

The experiments are run on a computing cluster system consisting of three machines, each with the configuration described in Table 3.1. All of the machines are interconnected with separate links between each pair of machines. The Debian GNU/Linux 6.0.4 operating system and the SHAREMIND 2 development version is installed on each of the servers.

Component	Manufacturer and model
CPU	2 × Intel Xeon X5670 2,93GHz/6.4GT/12M
RAM	12 x Kingston KVR1333D3D4R9S/4G 4GB/1333MHz/ECC/REG
Motherboard	TYAN S7012, i5520
Hard drives	4 x Seagate ST3146356SS HDD 146GB/15K/SAS
RAID controller	3Ware 9690SA-4I 4-port/SASI/HW-RAID-0/1/10/5/6/50
Network	4 x 1Gb LAN on motherboard

Table 3.1: Cluster hardware configuration

3.1.2 Network tuning

The performance of a network link is mainly dependent on two properties: bandwidth and latency. To model how the performance of the protocols behaves depending on

these properties, we needed to perform the experiments on network links with different settings. However, since configuring our computing cluster with different network interfaces would have been impractical, we instead simulated the different properties on the same physical link. The tools we used for this purpose, since we are running a Linux based operating system, were the *tc* traffic control tool of the *iproute2*[1] framework in combination with the *netem* network emulation module.

These tools allow us to specify a hierarchical structure of network packet queuing disciplines that shape the outgoing traffic on a link. Additionally, it is possible to filter the traffic so that traffic shaping is only applied to certain packets. In our configuration, we use the *tbf* (token bucket filter) queuing discipline to limit the bandwidth and *netem* to add additional latency to the traffic. The structure of the configuration is illustrated in Figure 3.1. In the figure, the queuing discipline directly attached to the interface is a simple priority queue that assigns packets to different sub-queues depending on the filters and priorities. The traffic matching the filter is shaped, whereas the rest of the traffic is passed through normally. This is the simplest configuration that matched our need of having a strict control over the maximum possible bandwidth, or capacity, and the latency of the connections.

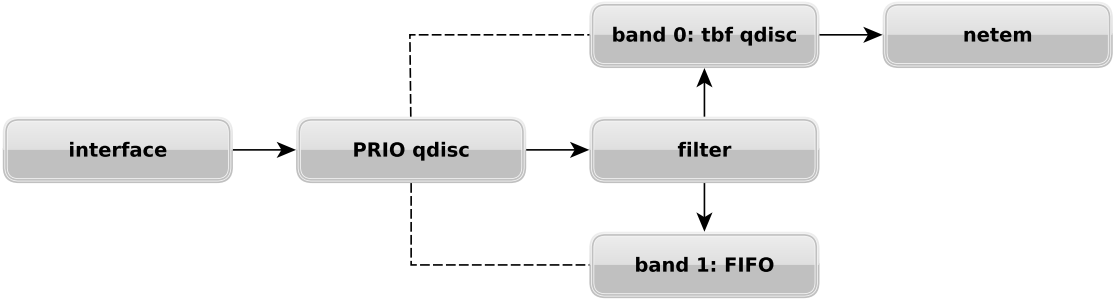


Figure 3.1: Structure hierarchy of traffic queuing

3.2 Experimental procedure

The basic procedure for running our experiments is similar across the experiments. In the next paragraphs, we make the assumption that SHAREMIND is correctly configured and running.

We use the *OperationBenchmark* tool, supplied by the SHAREMIND framework to run our experiments. The tool requires a number of parameters to be specified such as

[1]Linux Advanced Routing & Traffic Control, `http://lartc.org`. Last accessed: March 21st 2012

the SMC protocol executed, sizes of the input vectors, number of iterations to run, order of the test cases and so forth. Depending on the parameters, the tools then schedules a number of experiments. The procedure for each experiment is the following:

1. Request two random input vectors to be generated.
2. Request the secure operation to be run.
3. Wait for the operation to finish running.
4. Get the SHAREMIND server response that the computation is complete.
5. Measure the duration of the execution.

Each of the SHAREMIND computing nodes also logs the execution times of the operations to a profile. In our later analysis, we always use the profiling results from the computing nodes since it is more accurate for showing the real performance than the results shown by *OperationBenchmark*. This is because the latter also measures the round-trip time spent on communicating with the distributed server, which includes the computation requests and receiving the result back. The profiling results are closer to a real-life implementation where we only communicate with the client for input queries or data and output results.

The profiling results can be further processed by the *ProfileLogAnalyst* tool. For each experiment run by the *OperationBenchmark* tool, it aggregates the results by collapsing all the lower level operations performed during that experiment into one set of measured attributes. As a result, we get the time and communication cost for a protocol with a given input vector size.

3.3 Measurements

3.3.1 A breakdown of execution time

We get the running times of the protocols from the miner profiling results. This time includes all the sub-operations executed during the protocol. An extract of the timing results can be seen in Table 3.2. Each row in the table represents a benchmark result for a given vector size (element-wise operations on input vectors of the given size). The total running time is given and also time spent on different subtasks such as processing the incoming and outgoing queue or waiting for messages from the other nodes. Note that the total running time may be less than the summed time spent on the subtasks because the protocol can be executed in multiple threads.

We can plot these results as shown in Figure 3.2, where the horizontal axis shows the size of the input vector and the vertical axis shows the running time in milliseconds. The running time stays constant for smaller vector sizes because the network can handle these sizes with ease. After a certain size (*saturation point*), the running time starts increasing linearly. This happens for two reasons. First, because the communication

Nr.	Protocol	Input size	In queue	Out queue	Wait for receive	Total
1	Mult	10000	4 ms	2 ms	25 ms	26 ms
2	Mult	100000	29 ms	23 ms	139 ms	150 ms
3	Mult	1000000	301 ms	228 ms	1420 ms	1536 ms
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.2: Multiplication running time profile results

channels become saturated (we reach the maximum throughput for the protocol) and second, because the messages need to be broken down into smaller pieces (when vector sizes are greater-than the batch size), adding extra time cost. For each protocol, the saturation point is reached at different input vector sizes.

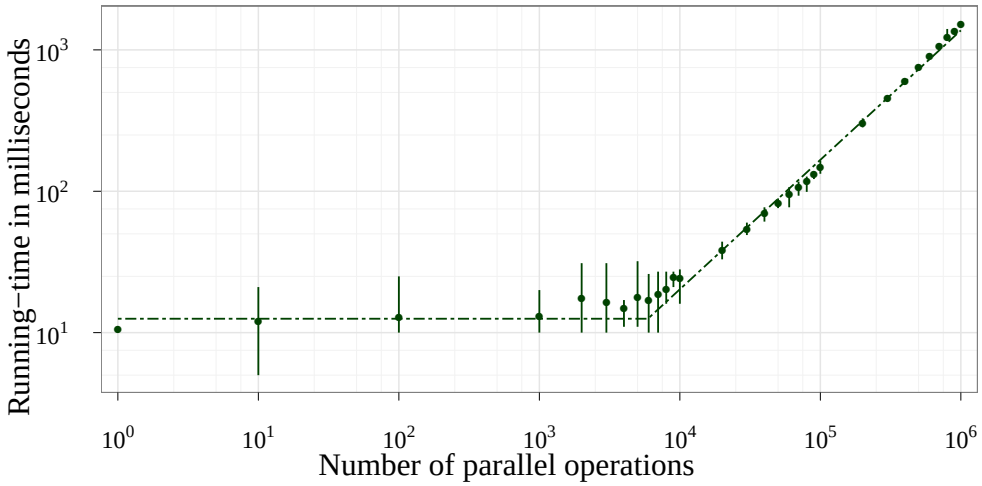


Figure 3.2: Multiplication running time based on input vector size

3.3.2 Measuring network traffic throughput

The network engine used in SHAREMIND, RakNet, measures the amount incoming and outgoing traffic for the entire duration of a connection and also for a shifted time window measured over one second. The latter value is continuously logged during the lifetime of a miner. These measurements are made at small fixed time intervals. This traffic information is illustrated in Figure 3.3, where we show the incoming traffic rate during the execution of a multiplication protocol benchmark. The traffic is shown for a connection

between two miners, measured at one end of the connection. The average traffic rate for the duration of that connection is measured at 33.3 Mbit/s, with the median at 41.8 Mbit/s. A similar plot can be drawn for the outgoing traffic.

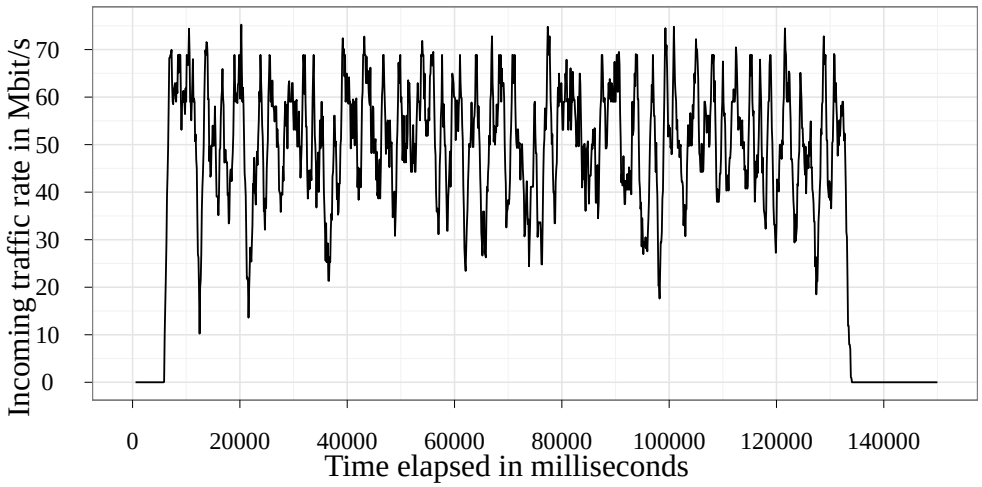


Figure 3.3: Incoming traffic rate during the multiplication benchmark

We also measure the total amount of bytes sent during the execution of a protocol. We do not use this information directly. However, it can be useful for validating the communication complexity of a protocol.

3.3.3 Measuring network latency

The round-trip time (ping) on the network connections to the other nodes is also continuously measured, similarly to the network traffic rate. The measurements are again made at fixed intervals and reported by the network layer. Figure 3.4 shows the round-trip time on a connection between two miners during the execution of a multiplication protocol benchmark (in ideal network settings). The average ping for the duration shown in the figure is 7.9 ms, with a median of 1 ms.

To look at the distribution of the latency we have a custom protocol that sends a vector of fixed size over the network to another node. The receiving node immediately returns it to the sender, which logs the round-trip time. This process is repeated over many iterations. Note that when measuring the round-trip time this way, it also includes the time that the SHAREMIND network layer takes to process the incoming and outgoing messages.

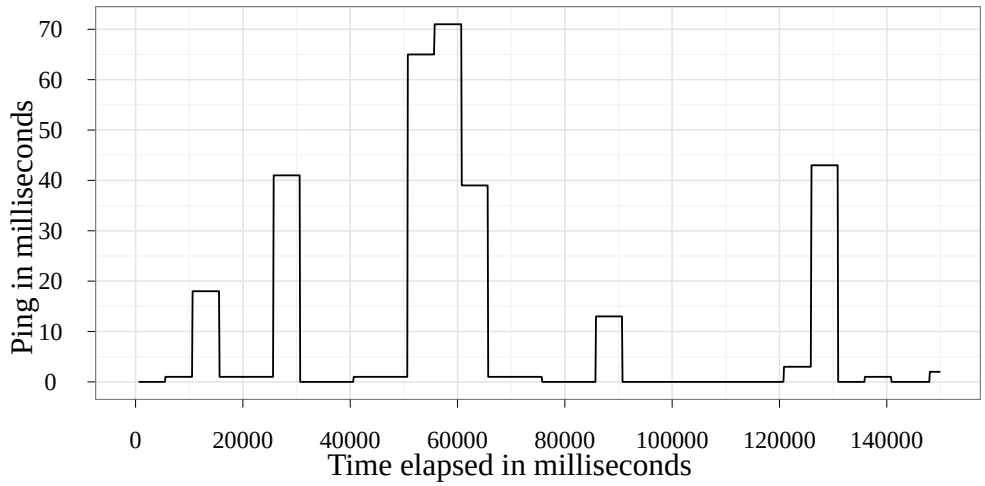


Figure 3.4: Round-trip time during the multiplication benchmark

Chapter 4

Structural analysis of Sharemind protocols

4.1 Theoretical complexity of Sharemind protocols

The theoretical complexities of the SHAREMIND protocols has been analyzed by Bogdanov et al. in [6] and are shown in Table 4.1. Both the round and communication complexity depends on the size of the values the protocol operates on. In our case, since we are working on 32-bit values, $n = 32$ and $\ell = \log_2 n = 5$. For the division protocol the additional parameters are $n' = 37$ and $m = 254$. For detailed discussion what the parameters are and how they were derived, refer to the cited paper.

Protocol	Rounds	Communication
Mult	1	$15n$
ShareConv	2	$5n + 4$
Equal	$\ell + 2$	$22n + 6$
ShiftR	$\ell + 3$	$12(\ell + 4)n + 16$
BitExtr	$\ell + 3$	$5n^2 + 12(\ell + 1)n$
PubDiv	$\ell + 4$	$(108 + 30\ell)n + 18$
Div	$4\ell + 9$	$2mn + 6m\ell + 39\ell n + 35\ell n' + 126n + 32n' + 24$

Table 4.1: Theoretical complexity of the protocols

In the following sections we study the structure of two SHAREMIND protocols. We later utilize some of the observations in building the performance model for the protocols.

4.2 Multiplication

The structure of the multiplication protocol is illustrated in Figure 4.1. It is a symmetrical protocol, consisting of one round where each miner transmits $5n$ message bits for each pair of input elements the operation is executed on. The total amount of traffic sent during the protocol by the three miners is $3 \times 5n = 15n$, which is the theoretical communication complexity seen in Table 4.1.

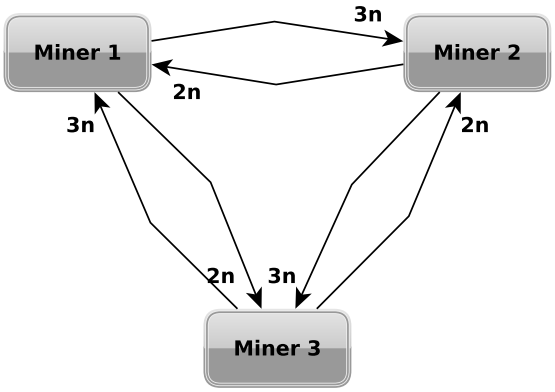


Figure 4.1: Multiplication protocol rounds

A more detailed structure of the protocol is shown in Table 4.2. The multiplication protocol computes the shared value $[\![w]\!] = [\![uv]\!]$, given the shared input values $[\![u]\!]$ and $[\![v]\!]$. The randomly generated values r_i, t_i, s_i are used to ensure the independence of the inputs $[\![u]\!], [\![v]\!]$ and output $[\![w]\!]$ by resharing those values. We denote the reshared inputs u', v' as $u' = u, v' = v$ and the reshared output w as $w = w'$. In the table, the reshared output w is computed directly, without the intermediate output w'. The correctness of the protocol follows from

$$\begin{aligned} w &= w_1 + w_2 + w_3 \\ &= w'_1 + w'_2 + w'_3 \\ &= u'_1v'_1 + u'_1v'_3 + u'_3v'_1 + u'_2v'_2 + u'_2v'_1 + u'_1v'_2 + u'_3v'_3 + u'_3v'_2 + u'_2v'_3 \\ &= (u'_1 + u'_2 + u'_3)(v'_1 + v'_2 + v'_3) \\ &= (u_1 + u_2 + u_3)(v_1 + v_2 + v_3) \\ &= uv. \end{aligned}$$

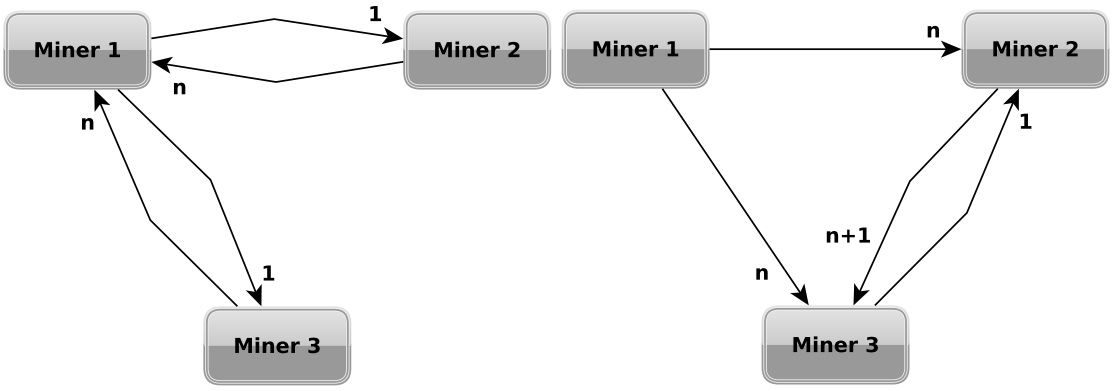
Note that there is only one round of communication in the multiplication protocol since no messages are sent in the second round, shown in the table. From the table we can see that each node sends a total of 5 messages per input pair to the other nodes, resulting in the $5n$ message bits per node and a total of $15n$ bits, where n is the bit length of one element.

	Node 1	Node 2	Node 3
Input	u_1, v_1	u_2, v_2	u_3, v_3
Round 1	$r_1, t_1, s_1 \leftarrow \mathbb{Z}_{2^n}$ $u'_1 := u_1 - r_1$ $v'_1 := v_1 - t_1$ Send r_1 to Node 3 Send t_1 to Node 3 Send u'_1 to Node 2 Send v'_1 to Node 2 Send s_1 to Node 2	$r_2, t_2, s_2 \leftarrow \mathbb{Z}_{2^n}$ $u'_2 := u_2 - r_2$ $v'_2 := v_2 - t_2$ Send r_2 to Node 1 Send t_2 to Node 1 Send u'_2 to Node 3 Send v'_2 to Node 3 Send s_2 to Node 3	$r_3, t_3, s_3 \leftarrow \mathbb{Z}_{2^n}$ $u'_3 := u_3 - r_3$ $v'_3 := v_3 - t_3$ Send r_3 to Node 2 Send t_3 to Node 2 Send u'_3 to Node 1 Send v'_3 to Node 1 Send s_3 to Node 1
Round 2	$u'_1 := u'_1 + r_2$ $v'_1 := v'_1 + t_2$ $u'_3 := u'_3 + r_1$ $v'_3 := v'_3 + t_1$ $s_1 := s_1 - s_3$ $w_1 := u'_1v'_1 + u'_1v'_3$ $+u'_3v'_1 + s_1$	$u'_2 := u'_2 + r_3$ $v'_2 := v'_2 + t_3$ $u'_1 := u'_1 + r_2$ $v'_1 := v'_1 + t_2$ $s_2 := s_2 - s_1$ $w_2 := u'_2v'_2 + u'_2v'_1$ $+u'_1v'_2 + s_2$	$u'_3 := u'_3 + r_1$ $v'_3 := v'_3 + t_1$ $u'_2 := u'_2 + r_3$ $v'_2 := v'_2 + t_3$ $s_3 := s_3 - s_2$ $w_3 := u'_3v'_3 + u'_3v'_2$ $+u'_2v'_3 + s_3$
Output	w_1	w_2	w_3

Table 4.2: Multiplication protocol structure

4.3 Share conversion

The share conversion protocol structure is shown in Figure 4.2. The protocol consists of two rounds and is asymmetrical, meaning that the nodes perform different tasks. The traffic transfered by the three miners amounts to $5n + 4$ bits per input, also seen in Table 4.1.



(a) Round 1 (b) Round 2

Figure 4.2: Share conversion protocol rounds

The share conversion protocol converts a shared value $[\![u]\!]$ in $\mathbb{Z}_2$ to a shared value $[\![w]\!]$ in $\mathbb{Z}_{2^n}$. This conversion is useful for algorithms working over $\mathbb{Z}_{2^n}$ that perform bit-level operations. From $[\![w']\!]$, we obtain reshared output $[\![w]\!]$, using the random values t_{ij}. To show the correctness of the protocol we note that

$$\begin{aligned} u &= u_1 \oplus u_2 \oplus u_3 \\ &= b \oplus m \oplus b_2 \oplus s_2 \oplus b_3 \oplus s_3 \\ &= m \oplus s. \end{aligned}$$

If $s = 1$, then the output is $w = w_1 + w_2 + w_3 = w'_1 + w'_2 + w'_3 = 1 - m_2 - m_3 = 1 - m$. The latter operation is equal to the binary operation $m \oplus 1$ when computed in $\mathbb{Z}_{2^n}$. Alternatively, if $s = 0$, then the result is $w = w_1 + w_2 + w_3 = w'_1 + w'_2 + w'_3 = m_2 + m_3 = m$, equal to $m \oplus 0$ in $\mathbb{Z}_{2^n}$.

4.4 Practical communication complexity

Based on the structure of each protocol, we can compute the *critical path* of the protocol communication. The *critical path* is the minimum amount of communication that has to be finished to complete the protocol. We have to take into account that some data can be

	Node 1	Node 2	Node 3
Input	u_1	u_2	u_3
Round 1	$b, b_2 \leftarrow \mathbb{Z}_2$ $m_2 \leftarrow \mathbb{Z}_{2^n}$ $m := \mathbb{Z}_{2^n}(b \oplus u_1)$ $m_3 := m - m_2$ $b_3 := b \oplus b_2$ Send b_2 to Node 2 Send m_2 to Node 2 Send b_3 to Node 3 Send m_3 to Node 3	$t_{21} \leftarrow \mathbb{Z}_{2^n}$ Send t_{21} to Node 1	$t_{31} \leftarrow \mathbb{Z}_{2^n}$ Send t_{31} to Node 1
Round 2		$s_3 := b_2 \oplus u_2$ $t_{23} \leftarrow \mathbb{Z}_{2^n}$ Send s_3 to Node 3 Send t_{23} to Node 3	$s_2 := b_3 \oplus u_3$ Send s_2 to Node 2

Round 3	$w_1 := t_{21} + t_{31}$	$s := s_2 \oplus s_3$
		if $s = 1$ **then**
		$w_2 := 1 - m_2$
		else
		$w_2 := m_2$
		$w_2 := w_2 - t_{21} - t_{23}$
			$s := s_2 \oplus s_3$
			if $s = 1$ **then**
			$w_3 := -m_3$
			else
			$w_3 := m_3$
			$w_3 := w_3 - t_{31} + t_{23}$

Output	w_1	w_2	w_3

Table 4.3: Share conversion protocol structure

transferred in parallel—two nodes can send data to a third node at the same time. The round barriers play an important role in calculating the path, as new messages can only be sent after finishing the previous round.

Based on how we model the connections in the network, we get different results for the *critical path*. We consider two types of connections: *full-duplex* and *half-duplex*. With the *full-duplex* network connections, data can be transferred in both directions between two nodes independently, meaning that both directions have their own bandwidth limit. The *half-duplex* connections have a shared bandwidth limit for both directions of the connection.

The computed *critical path* coefficients are shown in Table 4.4. As an example, we show how the coefficient for the multiplication protocol is computed. It is easy to see the amount of traffic to be transferred to complete the protocol in Figure 4.1, as multiplication is a symmetrical and single round protocol. For every connection between the nodes, in the *full-duplex* case we take the maximum of the data sent either way and in the *half-duplex* case, the sum of the data sent both ways. The coefficients for *full-duplex* and *half-duplex* are 3.0 and 5.0 accordingly. For multi-round protocols we also have to take into account the round barriers.

Protocol	Full-duplex	Half-duplex
Mult	3.0000	5.0000
ShareConv	1.0625	2.0937
Equal	5.9375	6.9687
BitAdd	15.9993	15.9993
ShiftR	32.0625	34.1250
BitExtr	69.0000	88.0000
PubDiv	92.0937	93.1875
Div	435.0937	548.2187

Table 4.4: Protocol communication coefficients

The more complex protocols are analyzed by first constructing a communication graph. The round barriers for each miner are represented as the vertices in the graph. The vertices are connected with directed edges denoting the communication from one miner to another with weights according to the volume of data transferred. The resulting graph is directed and acyclic. On this graph we find the longest path from any source node to any sink node, giving us the critical path of the communication.

Chapter 5

Modeling the performance of secure computation protocols

5.1 Goal

The theoretical complexity of SMC protocols is well studied. However, it is not enough to accurately estimate the performance of a real-life SMC deployment. Our goal is to find a model for each of the SHAREMIND protocols that would estimate the computation time depending on the bandwidth and the latency of the network links between the computing nodes.

We first construct a base model that estimates the running time of a protocol, depending on a given input size. In our base model, the network parameters are known and controlled with our network simulation tools, although some deviation from these parameters exist. The controlled parameters are bandwidth and latency of the connections. Other network parameters (such as packet loss) are not taken into account as be believe that a model based on bandwidth and latency sufficiently accurate. For each protocol we get the coefficient estimates for all of the different parameter sets, which we can then use to predict the performance of that protocol. The data can also be used to construct more general model over multiple protocols, at the cost of some loss in accuracy.

The results are only shown for 5 of the SHAREMIND protocols because of the time constraints set on this work. However, we believe that useful observations can be made based on the data, leading to generalizations that extend to the other protocols.

5.2 Experimental plan

Since we want to predict the computation time depending on bandwidth and latency, we had to perform experiments with different combinations of the two parameters. We

simulated the network parameters as described in Section 3.1.2. The set of parameters we used for the bandwidth were: 1 Gbit/s (physical link limit), 100 Mbit/s, 10 Mbit/s and 1 Mbit/s. Note that we removed the 1 Gbit/s bandwidth results because they were identical to the 100 Mbit/s setting, since the average throughput recorded was always below 100 Mbit/s. This result hints that the performance of the protocols could be improved by increasing the throughput of the protocol layers in SHAREMIND. With lower bandwidth limits we had to decrease the batch size parameter in the miner configurations. Lowering the batch size decreases the bandwidth used by the nodes and is needed for the network layer of SHAREMIND to function properly. The additional latency added to the traffic was: 0 ms, 25 ms, 50 ms and for some protocols 100 ms, 250 ms and 500 ms. It is important to note that the actual latency of the packets is the added latency plus the latency generated by the physical network. For each protocol the experiments are run with every combination of the two parameters.

In each experiment, we use the *OperationBenchmark* tool to run a set of benchmarks. The input vector sizes that are used range from 1 element to 1 million elements, with increasing step sizes. For the vector sizes in range 1 to 1000 the vector size V_i in experiment i is computed with the following equation $V_{i+1} = 10 \times V_i$. For greater sizes than 1000 the vector size is computed as $V_{i+1} = V_i + 10^{\lfloor \log_{10} V_i \rfloor}$. For some of the longer benchmarks (with low bandwidth and high latency values) we run the experiments up to 100000 elements. Before each benchmark, we also run a warmup phase of three sets of multiplication with a vector size of 1 million elements. This warmup phase ensures that the network channels are saturated and we get more consistent results. Without the warmup, the first computations in the benchmark would otherwise benefit from the absence of traffic and give biased performance results.

The size ranges of the protocol operations in the benchmarks also determines the range of the constructed models.

5.3 Constructing the model

Our base model is a regression model in the following form

$$T = f(p) + \frac{S}{g(b)} + \varepsilon$$

where T is the running time of the protocol, S is the number of bits sent in the protocol (communication complexity), f is a function dependent on the ping p, g is a function dependent on the bandwidth b and ε is the random error. The model estimates the dependent variable T based on the independent variable S. The coefficients we try to find are $f(p)$ (the intercept) and $1/g(b)$ (the slope). This model is based on a hypothesis that bandwidth is the determinative attribute for predicting the execution time of the protocols. This hypothesis is supported by the observation that the running-time of a

protocol starts increasing after we raise the size of the input vectors past the *saturation point* for that protocol. If the network is saturated, the bandwidth is maximally used and computing with larger vectors takes linearly more time.

The variable S is dependent on the size of the input vector, each input element increases the amount of communication linearly. However, the slope of this increment step differs from one protocol to another. The slope in the model we are constructing is dependent on the *critical path* of the protocol communication, discussed in Section 4.4. For any protocol the values of S are computed as $S = V \times C \times n$, where V is the size of the input vector, C is the *critical path* coefficient and n is the size of the elements in the protocol.

We used the linear least squares approach to fit the model to the data and estimate the slope and the intercept for the model. The coefficients give us $f(p)$ and $g(b)$ for each experiment, where ping p and bandwidth b are fixed by us. Weighted least squares was used to fit the model over multiple protocols, with the weights found by manual inspection.

The main tool used in our analysis to manipulate the data is the R statistical computing environment[1].

5.4 Analysis

5.4.1 Model evaluation methods

To measure how well our model fits the data, we use the coefficient of determination R^2. The coefficient shows how much variation in the data has been explained by the model and is computed as

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2},$$

where y_i is an observed value of the dependent variable, f_i is a predicted value of the dependent variable and $\bar{y}$ is the arithmetic mean of the dependent variables. If the coefficient is estimated to be above 0.8, then we say that the model fits the data with good accuracy.

To examine the statistical significance of the estimated coefficients, we form a pair of hypotheses. We check a null hypothesis against an alternative hypothesis, where the null hypothesis states that the slope of the model is 0, meaning that there is no linear relationship between the dependent and the independent variables. If we express our model in the form $Y = \alpha + \beta X + \varepsilon$, where $\beta = \frac{1}{g(b)}$, the hypothesis pairs will be

[1]The R Project for Statistical Computing, `http://www.r-project.org`. Last accessed: March 23rd 2012

$$H_0\text{: } \beta = 0$$
$$H_1\text{: } \beta \neq 0.$$

With the t-test we obtain the p-values (the probability that the null hypothesis is true) for the coefficients. We reject the null hypothesis when $p < 0.05$, showing that a significant relation exists between X and Y.

In the residual analysis of the model, the following properties have to be checked: independence of the residuals, homoscedasticity of the residuals and if the distribution of the residuals is normal. We note that although one or more of the properties hold in some cases, for most of the cases, the residuals cannot be shown to have these properties. However, we believe that estimations can still be made with the constructed models.

In the following analysis we assume *full-duplex* connections, as a better fit for the model was achieved in the case. However, note that the difference in assuming *half-duplex* or *full-duplex* connection is only apparent when comparing the protocols to one another or making generalizations across multiple protocols.

5.4.2 Models of individual protocols

The results of fitting our base model with the least squares method for different network parameters for the multiplication protocol is shown in Table 5.1. For nearly all of our data sets listed in the table, the R^2 coefficient shows that the model fits the data well and we can estimate the dependent variable with high accuracy. We also see a significant relationship between the predicted variable and the coefficient $g(b)$. The significance of $f(p)$ can not be shown in some cases. The residuals analysis shows that while some of the results indicate residuals with normal distribution, it is not true in all cases. Also, in many cases the residuals have non-constant variance, for example in the case when the residuals are increasing as the fitted values grow. Similar tables for the other protocols can be found in Appendix A.

The measured coefficients for the multiplication protocol are visualized in Figure 5.1. The figure shows two diagrams of how the coefficients behave as the latency changes. The $f(p)$ coefficient is shown in the upper plot and the $g(b)$ coefficient in the lower one. Note that latency is shown in the logarithmic scale. The $f(p)$ coefficient (intercept) of the model shows growing tendencies as the latency increases. The tendency is not strictly monotone in some cases, however, this might be attributed to the less reliable estimates for the coefficient, observed with high latency. The increasing coefficient can be explained by the fact that as the latency grows, more time has to be spent on a single operation, leading to an upwards shift of the intercept in the linear regression model.

The $g(b)$ coefficient (inverse of the slope) of the model shows a monotonically decreasing tendency, as the latency grows. We assume that with increasing latency, the cap of the maximum bandwidth achieved is lowered, leading to increased time spent

Added ping	Bw. lim.	$f(p)$			$g(b)$			R^2
		Coef.	Std. Err.	p-val.	Coef.	Std. Err.	p-val.	
0	100	5.88	0.85	0	63492.06	120.94	0	0.999
50	100	21.56	2.59	0	51493.31	212.12	0	0.995
100	100	40.78	1.56	0	40816.33	83.30	0	0.999
200	100	10.58	23.66	0.655	17540.78	215.38	0	0.956
500	100	129.16	13.68	0	8447.37	28.54	0	0.996
1000	100	35.59	30.98	0.252	33.60	0.01	0	1.000
0	10	−43.44	2.24	0	9443.76	6.24	0	1.000
50	10	−15.68	1.80	0	9453.58	4.47	0	1.000
100	10	9.77	1.78	0	9455.37	4.47	0	1.000
200	10	35.88	2.84	0	6982.75	3.90	0	1.000
500	10	87.64	5.99	0	3348.96	2.02	0	1.000
1000	10	210.28	40.26	0	34.01	0.09	0	0.999
0	1	−60.31	1.61	0	903.89	0.04	0	1.000
50	1	−33.10	1.59	0	903.91	0.04	0	1.000
100	1	−6.99	13.46	0.604	733.51	0.22	0	1.000
200	1	−10.01	8.82	0.257	416.03	0.05	0	1.000
500	1	28.76	13.36	0.032	180.77	0.01	0	1.000
1000	1	−168.88	33.51	0	31.57	0.00	0	1.000

Table 5.1: Model fitting results for the multiplication protocol

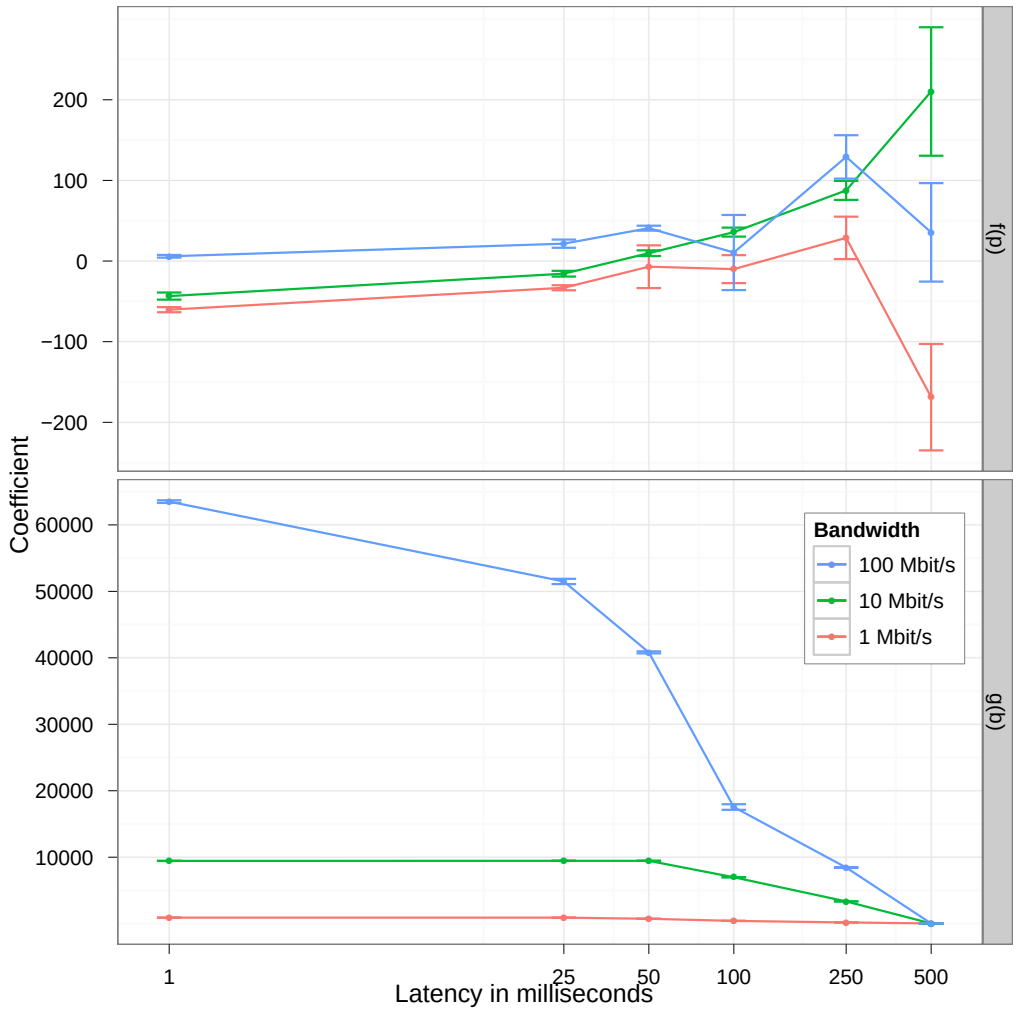
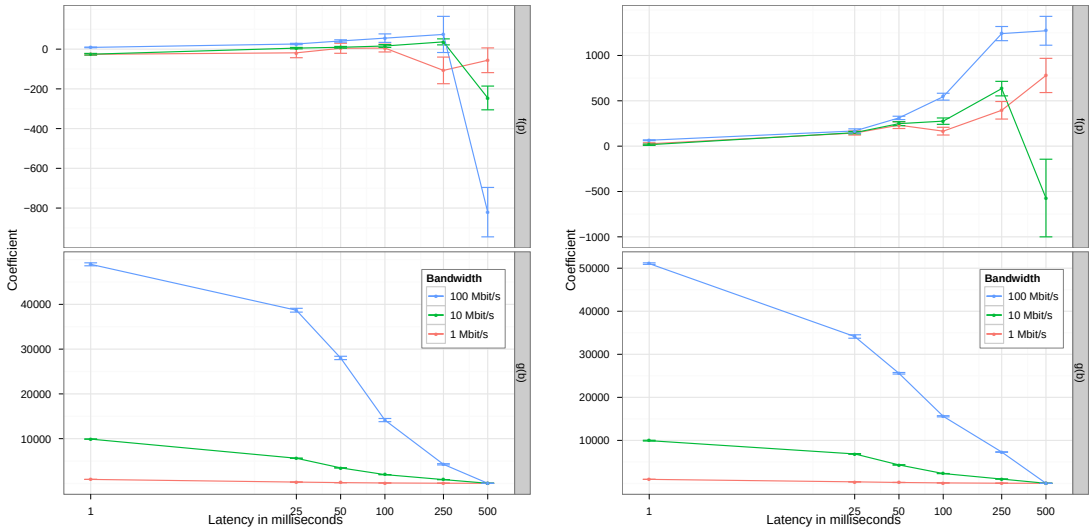


Figure 5.1: Estimated model coefficients for the multiplication protocol

on a single operation and a steeper slope in the model. The results for the slope of the model suggests that the relationship between $g(b)$ and the bandwidth holds.

Similar plots for the other protocols can be seen in Figure 5.2. The observations made about the coefficients of the model in the case of the multiplication protocol also apply to the other protocols.



(a) Share conversion

(b) Equality comparison

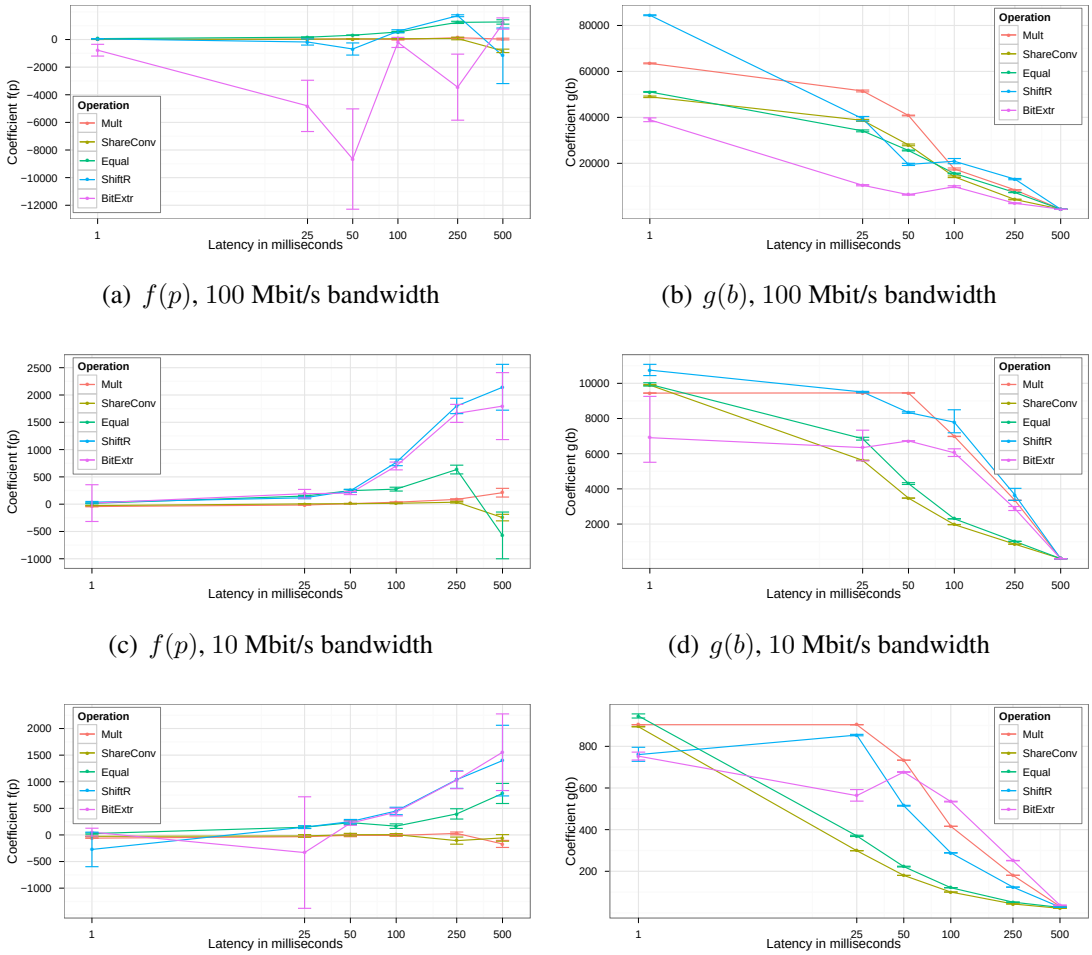
(c) Greater-than comparison

(d) Bit extraction

Figure 5.2: Estimated model coefficients for individual protocols

5.4.3 Joint modeling of secure protocol running time

Figure 5.3 shows how well the coefficient estimates align across the protocols. Both of the coefficients are plotted for all of the bandwidth levels in our experiments. The figure shows that some of the protocols behave similarly. The share conversion and the equality protocols give coefficient estimates with minimal difference. The difference in the values of the coefficients is greater between the other three protocols, however, a similar tendency can also be seen with these protocols.



(a) $f(p)$, 100 Mbit/s bandwidth

(b) $g(b)$, 100 Mbit/s bandwidth

(c) $f(p)$, 10 Mbit/s bandwidth

(d) $g(b)$, 10 Mbit/s bandwidth

(e) $f(p)$, 1 Mbit/s bandwidth

(f) $g(b)$, 1 Mbit/s bandwidth

Figure 5.3: Estimated model coefficients over all protocols

We construct two general models, one over the share conversion and the equality comparison protocol, as the two protocols show similar behavior, and the other over the

multiplication, greater-than comparison and bit extraction protocol. The results for the first general model are shown in Table 5.2. The R^2 coefficients in the table show that the two protocols in the first model behave very similarly and a single model can be used to predict the performance for both of the protocols.

Added ping	Bw. lim.	$f(p)$	$g(b)$	R^2		
				all	ShareConv	Equal
0	100	11.71	49627.79	0.997	0.996	0.997
50	100	15.40	35498.76	0.982	0.982	0.974
100	100	30.42	25947.07	0.982	0.979	0.977
200	100	81.29	14575.13	0.972	0.951	0.982
500	100	552.68	6268.41	0.913	0.829	0.967
1000	100	−2788.81	33.98	1.000	1.000	1.000
0	10	2.38	9938.38	0.997	0.999	0.995
50	10	78.58	6182.00	0.987	0.991	0.981
100	10	130.91	3853.56	0.985	0.990	0.979
200	10	285.42	2171.03	0.994	0.991	0.995
500	10	720.58	954.58	0.993	0.989	0.994
1000	10	853.28	33.54	0.999	1.000	0.999
0	1	−9.60	907.13	0.999	1.000	0.995
50	1	−204.05	317.79	0.990	0.994	0.971
100	1	−341.85	191.35	0.990	0.995	0.971
200	1	−1850.37	109.31	0.987	0.988	0.986
500	1	−4187.32	46.92	0.988	0.989	0.987
1000	1	−8125.15	24.03	0.988	0.989	0.987

Table 5.2: Model fitting results over the ShareConv and Equal protocols

The estimates for the second general model is shown in Table 5.3. The results for this model show that even though the model is accurate for some network parameters, it does not give good predictions in all cases. The accuracy is the lowest for the bit share extraction protocol, which may be due the higher complexity of the protocol, compared to the others.

Added ping	Bw. lim.	$f(p)$	$g(b)$	R^2			
				all	Mult	ShiftR	BitExtr
0	100	22.57	68634.18	0.905	0.993	0.926	0.731
50	100	−62.56	41511.00	0.765	0.938	0.953	0.258
100	100	−117.50	29913.25	0.683	0.866	0.807	0.180
200	100	124.39	16694.49	0.848	0.943	0.800	0.684
500	100	827.69	9912.77	0.743	0.962	0.868	0.196
1000	100	−3895.14	31.39	0.988	0.995	0.995	0.956
0	10	2.87	9578.54	0.873	1.000	0.958	0.296
50	10	282.43	9654.37	0.970	0.992	0.998	0.686
100	10	−63.67	8233.16	0.972	0.973	0.994	0.934
200	10	723.42	7131.65	0.933	0.975	0.807	0.910
500	10	1730.94	3421.96	0.920	0.967	0.779	0.900
1000	10	−861.02	31.25	0.983	0.991	0.978	0.931
0	1	323.25	867.90	0.981	0.997	0.925	0.975
50	1	322.95	830.60	0.941	0.987	0.993	0.717
100	1	1939.90	708.48	0.977	0.993	0.917	0.988
200	1	2505.30	459.15	0.917	0.991	0.776	0.849
500	1	5273.41	208.15	0.888	0.980	0.712	0.798
1000	1	12804.00	33.55	0.978	0.996	0.960	0.954

Table 5.3: Model fitting results over the Mult, ShiftR and BitExtr protocols

Chapter 6

Validating the model in a cloud environment

6.1 Motivation

The cloud environment enables quick deployment of web services by offering computing as a service. The cloud service providers run data centers with a large number of redundant machines. On top of these machines, they operate a virtualization layer, which in turn emulates the virtualized hardware provided to the customers. This additional virtualization layer brings benefits to the service providers and also the customers of the service. For the service provider, it allows for better utilization of the underlying hardware. For the customer it lowers the barrier of entry by exchanging capital expenditure for operational expenditure. It also offers *pay-as-you-go* payment schemes, better reliability and quick scalability, because the underlying machines emulating the virtual server can easily be modified by replacing some or adding new ones.

However, for many types of applications, a major issue with outsourcing computation and storage resources is that the cloud service providers have to be trusted to keep the information secure from outside attackers and also to not misuse the data themselves. For a lot of potential services on the cloud, the risk is not acceptable. When asked about the concerns with cloud computing, a survey [14] conducted by the European Network and Information Security Agency found that 67% of the small and medium-sized enterprises considered confidentiality of corporate data a "show-stopper" problem. However, many of those companies still show high interest in engaging with cloud computing to avoid in-house capital expenditure by outsourcing infrastructure, platforms or services.

A cryptographic solution to this privacy problem would be to use secure multiparty computations. With SMC we can still outsource the computation and storage, but we no longer have to trust the cloud provider since the data is secret shared or encrypted and the computations on the data are secure. In this work we would like to find out how

feasible it is to deploy SMC on the cloud in terms of performance and cost. There are also a few additional security concerns that have to be taken into account in the cloud environment when compared to the non-cloud setting.

To assess the feasibility of secure multiparty computation on the cloud, we deployed the SHAREMIND distributed server nodes with different cloud service providers.

6.2 Security concerns in cloud deployments

For multiparty computation to be secure, we make the assumption that we have a certain number of honest computing parties that do not collaborate with each other. In the case of SHAREMIND, we have three parties and we assume that none of them collaborate. It is difficult to make this assumption when all the three miner node machines are run by the same provider. Therefore we need to deploy the three miners with different cloud service providers. Each of the miners should be administrated by a different organization, such that each organization gains access to only one of the nodes. Additionally the cryptographic keys for each miner has to be generated and distributed securely. Also, the executed SECREC algorithms have to be distributed securely and checked for any undesired information leaks. An alternative approach to this distributed cloud setting would be co-located secure hosting, where the servers are located at the same geographic location but the access to the servers is restricted with some physical means.

With the security assumptions made in SMC and SHAREMIND in particular, most of the data privacy related issues are dealt with.

6.3 Choosing a cloud service provider

6.3.1 Pricing

In this work, we only consider cloud providers offering infrastructure as a service (IaaS) type of services, as we need a highly customizable server where we can compile and run our own software. Two types of payment schemes are used by the major cloud service providers today to charge for the running cost of the servers. First, the *pay-as-you-go*, also known as *on-demand*, scheme where the customers of the service pay for usage measured by a small time unit, usually an hour. Second, payments for longer time periods, usually a month, often equal to 730 or 750 hours. For some providers, the periodic payment schemes result in a lower per-hour cost and are preferred for longer periods of deployments with known usage estimates for the service. The running cost for both schemes is dependent on the resources allocated to the running virtual machine.

Another main source of cost, for some types of applications, is the bandwidth usage

charges for connections over the Internet. Bandwidth is usually charged per GB of traffic and is often only charged for outgoing traffic. No charges apply to the local traffic inside the data center for most cloud providers.

Finally, other minor, however noticeable sources of cost are caused by additional services, for example load balancers, or for allocation of some additional resources, such as fixed public IP addresses.

6.3.2 Location

One of the criteria for choosing the service provider is location. To minimize the delays for the clients of a service in the cloud, it is usually deployed in a data center not too far from the users. However, for our deployments, since we tested different configurations, we mostly valued the flexibility of the data center choices that we can run our nodes on. A large number of providers only offer their services in North America and Europe regions.

6.3.3 Chosen providers

In this work we chose to use Amazon EC2, Rackspace and Linode as our cloud service providers. Amazon EC2 was chosen mainly its market leader position. In choosing the other two, Rackspace and Linode, we took into account the pricing and the data center locations. The providers we considered, but did not select were AT&T, GoGrid, JoyentCloud, LayeredTech, OpSource, Softlayer, Terremark and VPS.NET.

6.4 Setting up a Sharemind installation on the cloud

We tested SHAREMIND in two different settings. First, a global setting where each node is located at a great distance to the other nodes. The locations chosen for the nodes were Tokyo (Japan), Fremont (United States of America) and London (United Kingdom). We also considered Ashburn, Virginia as a location instead of Fremont, California in the United States. However, we achieved more balanced connections using the Fremont data center, because of slightly better connectivity with Asia.

In the second, local setting, we chose three nodes in fairly close geographic locations. The locations for this setting were chosen in Europe, more specifically, two nodes in London, United Kingdom and one node in Dublin, Ireland.

A summary of all the different configurations is shown in Table 6.1.

The cloud servers of out choice had to meet a rough minimum criteria of at least 2 CPUs and at least 2 GB of RAM. The chosen server configurations are listed in Table 6.2. The same configuration was used for the global cloud and the European cloud.

Global cloud Sharemind deployment		
Node	Provider	Location
Node 1	Linode	Fremont, CA, US
Node 2	Rackspace	London, GB
Node 3	Amazon EC2	Tokyo, JP
European cloud Sharemind deployment		
Node	Provider	Location
Node 1	Amazon EC2	Dublin, Ireland
Node 2	Rackspace	London, GB
Node 3	Linode	London, GB

Table 6.1: Cloud service providers and data center locations

Provider	CPU	RAM	HDD	OS
Amazon EC2	2 vCPU	7.5 GB	850 GB	Ubuntu 11.10
Rackspace	4 vCPU	2 GB	80 GB	Debian 6
Linode	4 vCPU	2 GB	80 GB	Debian 6

Table 6.2: Server configurations

Since all of the three machines on the cloud are running a Linux distribution, the process of setting up SHAREMIND was similar for the servers. After installing the required packages either manually or through the given package management system, SHAREMIND was compiled from the source code. Once the software for each node was ready, we set up the keys required for the communication and configured the network addresses.

6.5 Measuring the parameters for the model

We measured the average round-trip time over 100 ping requests. This was done using the common *ping* tool.

To measure the bandwidth of a connection we saturated the channel, by transmitting as much data as the channel allowed over a duration of 10 seconds. As a UDP based data transfer protocol is used by the network layer of SHAREMIND, we measure the maximum throughput of UDP. For these measurements we used the *iperf*[1] network throughput testing tool. A channel with UDP bandwidth capacity over 100 Mbit/s is

[1]Iperf, `http://iperf.sourceforge.net/`. Last accessed: April 12th 2012

reported up to 100 Mbit/s.

We also measured the bandwidth in a similar way over TCP, as we noticed that the UDP measurements alone do not give a good estimation of the actual bandwidth usage by SHAREMIND.

The round-trip time and the bandwidth of the connections for the cloud settings are shown in Table 6.3. The bandwidth values for both, the TCP and UDP traffic, is shown in Mbit/s and the round-trip time in milliseconds. The row denotes the sender and the column the receiver in the tests. In some cases, the connections are not symmetrical. The measured values vary over time, with the TCP values showing most variation.

Global cloud									
	Node 1			Node 2			Node 3		
	TCP	UDP	Ping	TCP	UDP	Ping	TCP	UDP	Ping
Node 1				30.8	48.6	149.2	40.7	48.6	109.8
Node 2	12.5	59.7	149.1				4.6	59.7	264.7
Node 3	18.9	100	112.6	30.2	100	264.7			
European cloud									
	Node 1			Node 2			Node 3		
	TCP	UDP	Ping	TCP	UDP	Ping	TCP	UDP	Ping
Node 1				32.0	48.5	2.3	45.0	48.5	21.8
Node 2	56.1	59.6	2.5				52.9	59.6	20.3
Node 3	167	99.6	23.4	189	99.8	18.6			

Table 6.3: Measured bandwidth and round-trip time parameters

6.6 Estimating the running time of algorithms

Usually, to compute something useful, we need to run the protocols inside of larger algorithms. Based on the known computations of an algorithm, we can estimate the running time of its operations. For more information on frequent itemset mining in SHAREMIND refer to [4]. Table 6.4 shows the operations processed during the computation of the Apriori algorithm, calculating frequent item sets in data. We compare the estimated running time to the actual running time. The algorithm is executed on our experimental cluster. The total prediction time and the actual time for the multiplication operation differ by 8.18%, which is fairly accurate. However, the difference for the greater-than comparison operation is 56.22%, which is likely due to the very small vector sizes used in the operations. Note that the total running time of the algorithm is 137415 ms, which also includes other operations, such as database queries. For this

prediction it was unnecessary to measure the network, as the cluster network parameters were already known.

Operation	Operation count	Vector size	Predicted running time	Actual running time
Mult	88	812400	108613 ms	118256 ms
Mult	1	203100	313 ms	373 ms
Mult	1	8124	18 ms	26 ms
ShiftR	91	100	4674 ms	10684 ms
ShiftR	1	25	50 ms	116 ms
ShiftR	1	1	50 ms	108 ms

Table 6.4: Apriori algorithm running time prediction in the cluster

6.7 Estimating protocol performance on a new deployment

We also attempted to predict the performance of secure computations in any network setting, based on the models constructed in Section 5. To make a prediction based on the model, knowing the measured network parameters, we first have to compute the model coefficients $f(p)$ and $g(b)$. This is done by constructing a 2-dimensional grid for a given model, where one dimension holds the bandwidth values and the other dimension the latency values. The coefficient values are known at certain points, namely at the parameter values, which we used in our experiments.

Given the measured latency and bandwidth as p' and b', we can set a point in the grid and find the cell the point maps to. For each of the four corners of the cell we have the known values f_i and g_i for the coefficients to which we assign a weight w_i. To calculate the weights we compute the Euclidean distance from the measured point to each of the corners. For this, we first have to scale the the axes of the cell, such that it forms a square. A corner closer to the measured point is given a higher weight. The weights are additionally normalized, such that $\sum_{i=1}^{4} w_i = 1$. Finally, we calculate the coefficients as $f(p') = \sum_{i=1}^{4} f_i \times w_i$ and $g(b') = \sum_{i=1}^{4} g_i \times w_i$. We can only estimate the coefficients this way, when the measured bandwidth and latency falls within the limits of the network parameters we used in our experiments.

We performed similar protocol benchmarks in the cloud, as in our experiment cluster. The prediction results we got with the measured network parameters in the cloud were inaccurate and inconsistent. The exact results are omitted from this work, as no useful observations besides the invalidity of the predictions was drawn from the results.

In the cloud environment we were unable to accurately predict the running time of the protocols. However, we were still able to perform some analysis on the model itself. In each of the benchmarks we used to estimate the coefficients for our models, we also gathered some network statistics. For each protocol we measured the average ping and the average bandwidth used during the protocol execution, also known as average throughput. For the exact measured ping and bandwidth values refer to Appendix B. This gives us a set of n known reference pairs $mean(P_1), \ldots, mean(P_n)$ and $mean(B_1), \ldots, mean(B_n)$ for each protocol, where we also know the values for the model coefficients $f(p_i)$ and $g(b_i)$ for the model i. After measuring the average ping and throughput for a benchmark in the cloud, we assign a weight w_i to each of the known points based on the distance from the measured pair $mean(P')$ and $mean(B')$. The model with values closer to the measured value in the cloud is given a higher weight. We normalize the weights, such that $\sum_i^n w_i = 1$, and compute the coefficients as $f(p) = w_i \times f(p)_i$ and $g(b) = w_i \times g(b)_i$.

The results for estimating the coefficients this way and the accuracy of the predictions is given in Table 6.5. The table shows the measured ping and bandwidth values on the cloud and also the estimated model coefficient values based on the network parameters. The R^2 coefficient indicates that we can predict the results this way with some accuracy, although for most cases the model provided a lower accuracy than was observed in the experimental setting. This may indicate that the method we used to estimate the model coefficients is not very accurate, however, the underlying model works in the right way.

6.8 Discussion

The model analysis shows that we can use the models to predict the computation time of the protocols in the cluster SHAREMIND deployment. This is mainly useful to estimate the computation time of larger algorithms utilizing the protocols.

We were unable to reliably predict the running time of the protocols in the cloud, based on the models. However, we were able to give some estimations by comparing the average used bandwidth of the known models and the measured benchmark in the cloud. Based on this result we can say that the model is not invalid, but since we are unable to correctly evaluate the input parameters for the model, primarily the effective bandwidth, we are unable to make predictions with the model.

Global cloud					
Operation	Average ping	Average throughput	Estimated $f(p)$	Estimated $g(b)$	R^2
Mult	235.90 ms	0.57 Mbit/s	-4.01	1933.82	0.726
ShareConv	207.01 ms	0.73 Mbit/s	-28.33	2572.97	0.877
Equal	189.99 ms	0.42 Mbit/s	263.10	1695.60	0.671
ShiftR	219.45 ms	0.84 Mbit/s	567.16	4086.95	0.732
BitExtr	281.70 ms	0.78 Mbit/s	137.00	2413.72	0.631
European cloud					
Operation	Average ping	Average throughput	Estimated $f(p)$	Estimated $g(b)$	R^2
Mult	29.14 ms	1.02 Mbit/s	-26.38	4263.76	0.895
ShareConv	27.83 ms	1.17 Mbit/s	-12.64	5739.77	0.604
Equal	30.38 ms	1.54 Mbit/s	182.85	6129.75	0.708
ShiftR	37.68 ms	2.03 Mbit/s	198.84	7071.33	0.625
BitExtr	66.46 ms	1.28 Mbit/s	-473.54	3411.62	0.669

Table 6.5: Estimated models based on average utilized bandwidth

Chapter 7

Estimating the economic feasibility of secure computation in the cloud

7.1 Performance of Sharemind deployments

To performance of the SHAREMIND protocols determines the applications that the framework can be used for. The applications range from secure surveys with few computations to data mining algorithms with complex computations. Table 7.1 shows the performance of SHAREMIND protocols, depending on the deployment environment. The cloud deployments are roughly $10 - 30$ times slower than the ideal cluster setting. This indicates that if we can perform computations for an application in the cluster environment withing a reasonable time constraint, we can also perform these computations in the cloud environment with a predictable factor of increased running time.

Operation	Deployment setting		
	Cluster	Global cloud	European cloud
Mult	690000 ops/s	27850 ops/s	48300 ops/s
ShareConv	1360000 ops/s	58500 ops/s	120000 ops/s
Equal	254000 ops/s	12750 ops/s	21350 ops/s
ShiftR	95000 ops/s	2630 ops/s	3072 ops/s
BitExtr	28000 ops/s	633 ops/s	977 ops/s

Table 7.1: Protocol performance in different configurations

7.2 Cost of Sharemind deployments

In an example scenario we have three organizations that want to conduct a survey. However, the nature of the data gathered in the survey is too sensitive to be handed over to a single party. To solve this problem, the parties decide to use SMC to privately gather the data and securely generate the survey results. They decide to set up a SMC platform, in this case SHAREMIND, such that each of the three organizations controls one of the nodes, performing the data gathering and result generation. Each of the parties chooses cloud service provider and rents a server, where they set up their node for the duration of the survey.

After some planning, the organizations agree on a time schedule. They decide to set up the system in a time frame of one week, after which they will start collecting the survey data. The data gathering period ends after one month, when finally, the summary results of the survey are computed and released to the involved parties.

The chosen survey form contains 20 multiple choice questions, each with 5 choices for the answer. After the data gathering period we find out that the survey has been answered 5000 times. For each of the questions in the survey, the parties wish to generate a histogram. Based on this data, we estimate the computation time and cost for generating the survey results. To generate the histograms, we need to utilize the comparison protocols in the SHAREMIND framework. In the case of fixed set on answers, one could use the equality comparison protocol to count the answers. In the case of a non-fixed numeric answer, one could use the greater-than comparison protocol to count the answers within a certain range. We also estimate the cost for the setup phase and the data gathering phase based on the duration.

The estimations can be seen in Table 7.2. The amount of data transferred during the setup phase is negligible. In the data gathering phase, the data amount can be visualized as a table of values, with 5000 rows and 20 columns. Each of the columns contains the answer to a question. The values of the table are shared between all of the parties. Based on this information we can estimate the transferred data amount as $5000 \times 20 \times 3 \times 32 = 1.2$ MB. In the computation phase, if we have k possible answer choices, which in our case equals to 5, then we have to execute the run the equality protocol on the each of the questions $k - 1$ times. Knowing the communication complexity of the protocol, discussed in Section 4.1, we can estimate the transferred data amounting to $5000 \times 20 \times 4 \times 710 = 35.5$ MB. With the greater-than protocol (labeled ShiftR), when counting the answers in a certain range, we have to execute the protocol $k - 1$ times on each question, where k is the number of ranges we divide the histogram into. If we assume that $k = 5$, then the data amounts to $5000 \times 20 \times 4 \times 3472 = 173.6$ MB. In calculating the upkeep cost we assumed 750 hours charge for a month with a minimum of 0.12 and a maximum of 0.36 USD per hour. As for the data cost, we assumed the bandwidth charges for only the outgoing traffic with a minimum charge of 0.1 and a

maximum charge of 0.2 USD per GB of data.

Based on the constructed protocol performance models and the measurements made on the cloud we can also estimate a rough duration for the computations made for generating the results of the survey.

Phase	Duration	Upkeep cost per node	Data transferred	Data cost
Setting up	1 week	negligible	negligible	negligible
Data gathering	1 month	90 – 270 USD	1.2 MB	< 0.001 USD
Computations: Equal prot.	< 60 s	negligible	35.5 MB	0.0036 – 0.0071 USD
Computations: ShiftR prot.	< 240 s	negligible	173.6 MB	0.0174 – 0.0347 USD

Table 7.2: Estimated costs of the secure survey

The cost estimations indicate that the computation costs are negligible compared to the cost of running a cloud server for the duration of the data gathering phase. The cost of the computations will only start to be significant in scenarios where we perform very computation heavy tasks, where the data communication cost between the nodes starts to overweight the server upkeep cost. However, this scenario will only be reached with data transfers more than 600 GB to 1 TB a month, depending on the transfer cost. It is highly unlikely for any survey to generate this much traffic unless we are performing some complex analysis on the data. Alas, with heavy data traffic, one might adapt to a more suitable cloud service pricing model. For example, some providers do not charge for a certain amount of data transfer each month, others apply a smaller per GB charge for larger data transfers.

Chapter 8

Conclusion

The theoretical complexity of secure multiparty computation protocols is well studied. However, the theoretical results are not enough to accurately predict the performance of an SMC protocol in a real-world deployment setting. In this work we carried out the task of constructing a performance model for SMC protocols for the SHAREMIND framework.

In the process of constructing the general performance model, we first proposed a base model, in the form of a linear regression, predicting the protocol running time based on its input size. Experiments were run on our testing cluster to gather data for the model analysis. The analysis indicated an accurate fit of the model on the data. A generalization of the model was made based on the similar behavior of some of the protocols.

To validate our general model, we set up a set of servers in the cloud environment. In this setting we measured the parameters of the network connections between the machines. The predictions were compared to the actual computation results. We were unable to accurately predict the running time of the protocols in the cloud. However, we concluded that this result was probably due to the inability to accurately estimate the effective network parameters to compute the model coefficients.

The model validation in the experiment cluster environment showed that the models can be used to accurately predict the running time of the secure operations inside more complex algorithms.

In the last part of the work, we utilized the general model to assess the feasibility of SMC in the cloud environment. With the model, we computed time estimations for executing certain operations in a sample scenario. The cost estimation showed that for a secure survey scenario, the cost of the secure computations is low compared to the cost of keeping the server running during the data gathering phase for the survey. The cost of the performed operations only starts to play a role with large data sets and computation heavy algorithms.

The results of this work indicate that it is indeed feasible to do secure multiparty

computation in the cloud environment for a whole range of real-world scenarios. This mainly benefits the potential cloud service scenarios where privacy of the stored data is one of the primary concerns.

In this work we also found some indications of possible improvements to the SHAREMIND framework. We noticed that even though the protocols have a lot of available bandwidth, they are not using it. For high throughput connections, the performance of the protocols may be significantly increased if the bandwidth utilization rate can be improved.

As a future work, we could try other approaches to construct the models or, alternatively, build a specialized tool to measure the bandwidth parameter for the models.

Bibliography

[1] The Sharemind secure computation framework. http://sharemind.cyber.ee. [Online; accessed 06-May-2012].

[2] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), STOC '88, ACM, pp. 1–10.

[3] BLAKLEY, G. R. Safeguarding cryptographic keys. *International Workshop on Managing Requirements Knowledge 0* (1979), 313.

[4] BOGDANOV, D., JAGOMÄGIS, R., AND LAUR, S. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Proceedings of the Pacific Asia Workshop on Intelligence and Security Informatics* (2012), Heidelberg: Springer, pp. 112–126. To appear.

[5] BOGDANOV, D., LAUR, S., AND WILLEMSON, J. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings* (2008), vol. 5283 of *LNCS*, Springer, pp. 192–206.

[6] BOGDANOV, D., NIITSOO, M., TOFT, T., AND WILLEMSON, J. High-performance secure multi-party computation for data mining applications, 2012. To appear.

[7] BOGETOFT, P., DAMGÅRD, I., JAKOBSEN, T., NIELSEN, K., PAGTER, J., AND TOFT, T. A Practical Implementation of Secure Auctions based on Multiparty Integer Computation. In *Financial Cryptography and Data Security*, G. Di Crescenzo and A. Rubin, Eds., vol. 4107 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 142–147.

[8] CHAUM, D., CRÉPEAU, C., AND DAMGÅRD, I. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1988), STOC '88, ACM, pp. 11–19.

[9] CHOR, B., GOLDWASSER, S., MICALI, S., AND AWERBUCH, B. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1985), IEEE Computer Society, pp. 383–395.

[10] CRAMER, R. Introduction to Secure Computation. In *Lectures on Data Security*, I. Damgård, Ed., vol. 1561 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1999, pp. 16–62.

[11] CRAMER, R., DAMGÅRD, I., AND NIELSEN, J. B. Multiparty Computation, an Introduction, 2009.

[12] JAGOMÄGIS, R. SecreC: a Privacy-Aware Programming Language with Applications in Data Mining. Master's thesis, Institute of Computer Science, University of Tartu, 2010.

[13] KERSCHBAUM, F., BISWAS, D., AND DE HOOGH, S. Performance Comparison of Secure Comparison Protocols. In *20th International Workshop on Database and Expert Systems Application, 2009. DEXA '09.* (31 2009-sept. 4 2009), pp. 133–136.

[14] NETWORK, E., AND (ENISA), I. S. A. An SME perspective on Cloud Computing, November 2009. Survey.

[15] SHAMIR, A. How to share a secret. *Communications of the ACM 22* (November 1979), 612–613.

[16] YAO, A. C. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1982), SFCS '82, IEEE Computer Society, pp. 160–164.

Appendix A

Model coefficient estimation results

In the following tables, the model coefficients are obtained assuming *full-duplex* network connections. For individual protocols, the accuracy while assuming *half-duplex* connections is the same and the coefficients differ by a protocol dependent constant factor.

Added ping	Bw. lim.	$f(p)$			$g(b)$			R^2
		Coef.	Std. Err.	p-val.	Coef.	Std. Err.	p-val.	
0	100	8.73	0.85	0	48947.63	167.71	0	0.996
50	100	25.83	1.73	0	38684.72	209.51	0	0.991
100	100	41.14	2.92	0	28026.91	188.52	0	0.986
200	100	55.32	10.86	0	14142.27	180.00	0	0.952
500	100	73.99	46.23	0.111	4265.48	69.87	0	0.924
1000	100	−820.53	63.19	0	34.50	0.01	0	1.000
0	10	−25.62	2.01	0	9900.99	16.67	0	0.999
50	10	5.06	1.79	0.005	5622.72	4.74	0	1.000
100	10	9.26	2.19	0	3476.21	2.18	0	1.000
200	10	15.81	3.54	0	1968.78	1.12	0	1.000
500	10	36.51	7.68	0	856.15	0.47	0	1.000
1000	10	−245.72	30.39	0	33.99	0.00	0	1.000
0	1	−26.05	2.03	0	894.20	0.14	0	1.000
50	1	−18.61	12.51	0.138	298.88	0.09	0	1.000
100	1	4.25	12.82	0.741	180.18	0.03	0	1.000
200	1	5.38	9.97	0.590	100.31	0.01	0	1.000
500	1	−107.06	34.06	0.002	43.21	0.01	0	1.000
1000	1	−56.06	31.63	0.077	22.13	0.00	0	1.000

Table A.1: Share conversion protocol model coefficients

Added ping	Bw. lim.	$f(p)$			$g(b)$			R^2
		Coef.	Std. Err.	p-val.	Coef.	Std. Err.	p-val.	
0	100	64.98	2.35	0	51072.52	78.25	0	0.999
50	100	167.54	11.73	0	34129.69	198.02	0	0.989
100	100	311.36	9.44	0	25588.54	91.67	0	0.996
200	100	545.36	19.30	0	15598.19	70.56	0	0.994
500	100	1241.22	39.66	0	7296.07	31.41	0	0.994
1000	100	1272.12	80.80	0	33.54	0.00	0	1.000
0	10	15.67	3.79	0.000	9940.36	46.44	0	0.995
50	10	148.61	6.85	0	6852.60	40.38	0	0.993
100	10	248.00	11.09	0	4301.08	25.71	0	0.992
200	10	275.68	17.84	0	2299.01	1.43	0	1.000
500	10	634.45	40.72	0	1015.47	0.63	0	1.000
1000	10	−572.11	217.18	0.009	32.80	0.03	0	1.000
0	1	25.90	5.78	0	945.78	5.00	0	0.996
50	1	144.42	11.18	0	370.08	1.48	0	0.998
100	1	231.45	18.62	0	222.51	0.89	0	0.998
200	1	165.30	21.58	0	121.14	0.04	0	1.000
500	1	395.42	49.08	0	51.84	0.02	0	1.000
1000	1	779.49	95.68	0	26.54	0.01	0	1.000

Table A.2: Equality comparison protocol model coefficients

Added ping	Bw. lim.	$f(p)$			$g(b)$			R^2
		Coef.	Std. Err.	p-val.	Coef.	Std. Err.	p-val.	
0	100	50.15	4.99	0	84459.46	71.33	0	1.000
50	100	−178.63	113.39	0.116	39416.63	481.64	0	0.955
100	100	−686.07	223.35	0.002	19516.00	236.14	0	0.957
200	100	596.76	56.04	0	20903.01	568.02	0	0.861
500	100	1723.97	34.14	0	13123.36	136.06	0	0.977
1000	100	−1177.33	1019.50	0.250	33.15	0.05	0	1.000
0	10	35.98	7.06	0	10750.38	160.64	0	0.974
50	10	116.82	8.92	0	9504.80	18.97	0	0.999
100	10	245.18	13.17	0	8340.98	21.57	0	0.999
200	10	765.16	30.22	0	7788.77	328.20	0	0.815
500	10	1799.34	71.15	0	3657.78	170.45	0	0.782
1000	10	2141.79	212.11	0	34.58	0.05	0	1.000
0	1	−272.24	164.06	0.100	760.33	16.98	0	0.940
50	1	146.53	13.12	0	854.11	1.71	0	0.999
100	1	250.11	20.56	0	514.78	0.98	0	1.000
200	1	450.31	35.26	0	288.44	0.52	0	1.000
500	1	1039.24	82.75	0	123.98	0.23	0	1.000
1000	1	1397.16	335.47	0.000	28.01	0.05	0	1.000

Table A.3: Greater-than comparison protocol model coefficients

Added ping	Bw. lim.	$f(p)$			$g(b)$			R^2
		Coef.	Std. Err.	p-val.	Coef.	Std. Err.	p-val.	
0	100	−772.63	215.90	0.000	38940.81	424.59	0	0.966
50	100	−4805.26	943.27	0	10459.16	132.37	0	0.953
100	100	−8654.31	1846.24	0	6339.14	94.84	0	0.935
200	100	−211.16	185.19	0.255	9843.49	193.79	0	0.922
500	100	−3449.87	1213.22	0.005	2624.95	90.19	0	0.795
1000	100	1158.66	206.37	0	26.96	0.01	0	1.000
0	10	19.75	169.88	0.908	6909.42	883.67	0	0.368
50	10	192.10	39.07	0	6346.79	427.79	0	0.786
100	10	202.98	13.82	0	6724.50	6.78	0	1.000
200	10	699.78	36.08	0	6051.07	109.85	0	0.959
500	10	1662.92	83.37	0	2878.86	57.52	0	0.951
1000	10	1796.62	309.89	0	26.25	0.02	0	1.000
0	1	52.60	36.72	0.159	752.82	9.41	0	0.993
50	1	−331.17	529.80	0.533	563.24	13.99	0	0.927
100	1	221.43	20.14	0	676.79	0.77	0	1.000
200	1	429.79	35.85	0	534.72	0.85	0	1.000
500	1	1032.15	82.48	0	251.20	0.43	0	1.000
1000	1	1554.63	364.06	0	37.07	0.04	0	1.000

Table A.4: Bit extraction protocol model coefficients

Appendix B

Measured network parameters for the model experiments

Added ping	Bw. lim.	Round-trip time in milliseconds				
		Mult	ShareConv	Equal	ShiftR	BitExtr
0	100	9.23	1.93	3.22	15.51	9.32
50	100	68.71	55.39	55.17	68.60	76.08
100	100	125.88	104.87	104.22	127.29	140.59
200	100	248.12	204.68	205.11	220.67	241.39
500	100	619.09	541.17	514.10	544.21	622.82
1000	100	1070.69	1065.26	1072.25	1084.94	1074.60
0	10	30.32	11.86	0.84	5.53	1.58
50	10	66.75	52.21	52.87	61.14	51.59
100	10	107.47	101.28	101.21	103.44	119.69
200	10	202.85	200.96	200.84	201.23	202.94
500	10	501.71	500.76	500.75	500.72	501.27
1000	10	1067.06	1112.36	1054.85	1066.73	1057.48
0	1	11.67	2.60	5.79	5.44	11.98
50	1	54.89	50.84	52.58	50.89	65.22
100	1	100.79	100.65	100.69	100.77	115.78
200	1	200.72	200.63	200.55	200.75	204.12
500	1	500.70	500.62	500.55	500.70	501.68
1000	1	1084.21	1000.61	1000.57	1034.49	1030.34

Table B.1: Measured average round-trip times for model benchmarks

Added ping	Bw. lim.	Bandwidth in Mbit/s				
		Mult	ShareConv	Equal	ShiftR	BitExtr
0	100	27.79	13.65	18.84	39.96	24.85
50	100	21.35	9.93	12.70	20.83	7.19
100	100	16.89	7.58	9.37	11.09	4.43
200	100	8.98	4.56	5.90	5.83	5.57
500	100	4.53	1.90	2.82	3.02	1.81
1000	100	0.06	0.06	0.04	0.04	0.04
0	10	7.43	6.15	3.65	5.28	4.99
50	10	7.07	3.83	2.58	4.47	3.30
100	10	6.77	2.57	1.84	3.77	3.77
200	10	4.88	1.51	1.28	1.47	1.74
500	10	2.40	0.69	0.57	0.74	0.85
1000	10	0.06	0.06	0.03	0.04	0.03
0	1	0.82	0.78	0.62	0.54	0.64
50	1	0.82	0.29	0.45	0.47	0.37
100	1	0.66	0.18	0.32	0.29	0.38
200	1	0.38	0.10	0.07	0.17	0.27
500	1	0.17	0.04	0.03	0.07	0.12
1000	1	0.05	0.03	0.03	0.04	0.05

Table B.2: Measured average utilized bandwidth for model benchmarks