Performance of Secure Multiparty Computation

Ludwig Dickmanns, Marcel von Maltitz[*]
[*]*Chair of Network Architectures and Services, Department of Informatics*
Technical University of Munich, Germany
Email: ludwig.dickmanns@outlook.de, vonmaltitz@net.in.tum.de

***Abstract*—With the recent advancements in modern computer technology secure multiparty computation (SMC) evolved from a mere theoretical approach to a number of actively developed software projects. A major advantage of SMC is that it allows a set of parties to jointly calculate a function without any party revealing it's input.**
In our study we evaluated the influence of network parameters on the performance of a SMC framework, in order to derive an outlook for the feasibility of SMC applications in the future. For the evaluation the following parameters and the corresponding measurements were chosen: The impact of increasing the number of peers on execution time and protocol invocations and the effect of added network latency and decreased bandwidth on the execution time.
Our results indicate that SMC is a feasible option, especially in setups with a high bandwidth, low network latency and a limited number of peers. Linear increase in peers led to a linear increase in execution time and protocol invocations. The execution time increased drastically for a transmission rate of 10 MBit/s or lower. However, added network latency had the most significant negative impact.

***Index Terms*—secure multiparty computation, performance, measurement**

1. Introduction

The main goal of Secure Multiparty Computation is to allow several parties calculating a joint function. The corresponding inputs of each party are kept private and no Trusted Third Party should be required for the calculation.

The most prevalent example for SMC is Yao's Millionaires' Problem: Two millionaires wish to determine whom of both is wealthier – without either one of them revealing their credit balance. As mentioned in the beginning, no Trusted Third Party should be required. In his paper "Protocols for Secure Computation" from 1982, aforementioned A. C. Yao proposes a solution to this problem, which satisfies the above mentioned criteria. Furthermore, the researcher describes a generalized approach for similar problems with more than two parties calculating a collective function without revealing their inputs, e.g. "Mental Poker" [1]. However, at this point in time a practical implementation was not feasible due to lack in computational power. Fortunately – with the advancement of technology over the recent years – computers are now capable of performing such tasks in an appropriate amount of time and thus SMC-Frameworks are emerging [2] [3].

Besides computational power there is another important point to consider when answering the feasibility question: Network performance. The purpose of this paper is to investigate in and help answering the following question: How do network parameters influence the performance of SMC? Hence, measurements were carried out examining the influence of the following parameter:

- Number of peers
- Network latency
- Transmission rate

Only for the first one, number of peers, we identified the impact on the amount of evaluated protocols, because the other two parameters are not influencing it. For all three of them execution time measurements were carried out. This is the most important factor in the context of usability because the application should be able to operate in an user-acceptable amount of time.

The remainder of this paper is structured as follows: After identifying related work in Section 2, there will be information about our testing setup in Section 3 divided in two parts: Testbed (3.1) and the SMC framework of choice and our adjustments to it (3.2). Then, the results of the measurements are shown in Section 4 and in Section 5 those results and the corresponding consequences are discussed. In the last Section (6) we draw a conclusion regarding our results and provide an outlook for future research.

2. Related Work

Firstly – as discussed in the introduction – the theoretical foundation for SMC was layed out in 1982 by A. C. Yao [1]. Secondly several SMC frameworks were implemented. Hence, the third step is to evaluate the performance of such applications in order to discuss whether the technology is applicable.

One publication investigating on this topic is "A performance and resource consumption assessment of secure multiparty computation" by Marcel von Maltitz and Georg Carle [4]. Here, the researchers analyzed the following parameters:

- Number of peers
- Network latency
- Transmission rate
- Packet loss

- Input data parallelization

With the purpose of examining data about their impact on the following performance indicators:

- Execution time
- CPU cycles
- Heap memory consumption
- Transmitted bytes

Their research work resulted in a promising outlook for SMC in the future. Intranet applications can already be considered feasible, however for Internet and mobile Internet applications the main bottleneck is network latency. A difference to this paper is the SMC protocol (BGW) used by the software under test. BGW will not be explained in this paper, however, in Section 3.2.1 there is a short introduction to SPDZ, which is the SMC protocol used by the software we tested.

In this paper there will be an analysis of three of the above five parameters. We used the same network setup in order to either validate the thesis by delivering a similar outcome or providing room for discussion by showing different results. However, we did not analyze packet loss and input data parallelization as this would exceed the frame of this study. Furthermore, we did not investigate CPU cycles, heap memory consumption and transmitted bytes, for the same reason.

Further measurements were taken in [5] [6].

3. Test Setup

The setup, in which the time measurements were carried out, consists of two parts. The first of which is the testbed, i.e. the hardware the test and measurement software was ran on. The second part is the SMC framework software – namely FRESCO – and our adjustments to it. Firstly, general information on FRESCO is given. Then, the tested application is introduced. Afterwards, we explain how the measurements were taken.

3.1. Testbed

For our tests we used a range of three to 17 physical peers in order to derive the influence factor of adding additional peers to the network. The hardware for each host was equal: A four core Intel Xeon CPU running at 2.50GHz with 8192KB of cache and 16GB RAM. All hosts were connected to each other with an 1 GBit networking interface and the default link latency is around 0.18ms. The network is organized in a star topology: A central switch in the middle is connected to three other switches. The ladder are connected to five to six hosts. As the operating system of choice on each machine Debian Stretch (9.4) was used with a 4.9 Linux kernel.

3.2. FRESCO

In this subsection we introduce the secure multiparty computation framework, which we used for our tests. Then software under test is introduced. Here, we adjusted an demo application, which is part of the framework, and those adjustments are explained. Afterwards, the methodology for the measurements is introduced.

3.2.1. General. The FRamework for Efficient Secure COmputation (FRESCO) is developed by the Alexandra Institute in Denmark. It is licensed under the open source MIT license. According to their documentation [7], the framework is already prototypically in usage, e.g. to evaluate surveys without revealing the answers of the participants. The code is written in Java, which helps with platform independence. Summarizing, the main goals of FRESCO are providing an infrastructure for uncomplicated SMC application development, with an easily adjustable design. Earlier mentioned protocols are defined in a protocol suite and represent the base functions of it. Hence, applications are built using protocols. Batches contain a certain number of protocols, which are evaluated in parallel. In addition, the effort for developing an individual protocol suite is reduced by contributing a framework of reusable patters. A central feature of FRESCO is an extra abstraction layer, in order to separate algorithm/application development from the mathematical realization of the underlying SMC protocol. Finally, FRESCO tackles scalability issues by supporting preprocessing and parallel execution. FRESCO is actively developed and maintained. The SMC protocol used by FRESCO is SPDZ, which allows secure arithmetic calculations for multiple parties via secret sharing [8]. A great example to explain and illustrate arithmetic operations with secret sharing is the addition with multiple parties. In this case, n parties wish to collaboratively calculate the sum of their inputs, here integer values. Party i contributes the input x^i. Each party splits it's inputs into n randomly sized parts $(x_1^i, ..., x_n^i)$ – so called shares – in a way that adding up all of the shares results in the input (x^i). E.g. for the input x^i this means $\sum_j x_j^i = x^i$. In the next step, the shares are exchanged between the parties: Party j receives share j of each party $(x_j^1, ..., x_j^n)$. Then, Party j calculates partial sum r_j of the received shares. In the last step all partial results are exchanged allowing each party to calculate the total sum [9]. The previously cited article and its follow-ups provide further insights to arithmetic operations with secret sharing and SPDZ.

3.2.2. Tested Application. For our studies we used version 1.1.2 of FRESCO [2], which contains demo applications. In one of those three parties are collaboratively calculating the sum of an integer array, which is the input of party one. Parties two and three initially had no inputs. In order to make this example more realistic, however, we adjusted it in order to allow more than three parties, with each of them contributing their own input array. Instead of the sum, the application calculates the mean and the variance of all of the inputs. Previously the input was hard coded inside the FRESCO code, but in our example the inputs are now read in from a CSV file. With those adjustments the example was made more versatile for testing as this allows to draw a conclusion about the impact of adding extra peers.

3.2.3. Measurements. As mentioned in the beginning of this paper, measurements were taken on the evaluated protocols and the corresponding batches and execution time. When running the FRESCO application, each peer produces an individual log file, where data about the last run are stored. Both evaluated performance indicators were extracted from those files and the collected data is illustrated in the figures of this paper in Section 4. Each measurement was carried out ten times and the median of those measurements was used for the illustrations.

4. Results

Separated by the input parameters – namely number of peers (4.1), network latency (4.2) and transmission rate (4.3) – this section addresses the results of this study. Hence, interpreting the impact of the above parameters on the execution time and the amount of transferred data is the content of this chapter.

4.1. Number of Peers

For our tests we used a range of three to 17 peers in order to investigate in the change in execution time (4.1.1) and protocol invocations (4.1.2) with an increasing amount of peers.

4.1.1. Execution Time. The effect of an increasing number of peers on the execution time is demonstrated in Figure 1. The x axis shows the number of peers, whereas on the y axis the execution time can be seen. A higher amount of peers leads to more input for computation as well as an increase in communication between the peers. Both of these factors result in an extended execution time. In addition, the figure illustrates a linear behavior, which can be considered positive as e.g. quadratic behavior would be much worse.

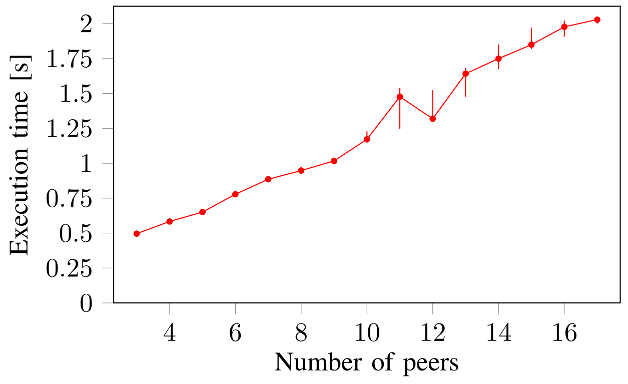

Figure 1. The execution time depending on the number of participating peers.

4.1.2. Protocols and Batches. Besides the execution time, the amount of protocol invocations and the number of corresponding batches was measured. As shown in Figure 2 an increase in peers led to more protocol invocations in an higher amount of batches. Both measured indicators seem to follow linear behavior – as the execution time does – in relation to the number of peers. While the quantity of batches grows relatively slowly, the number of protocol invocations increased faster in comparison. However, as both numbers are increasing linearely, this example provides an argument in favor of the feasibility question. For very large applications this still remains a point to consider, but even here (and from the execution time standpoint) linear increase is by far more acceptable in contrast to e.g. quadratic or cubic behavior.

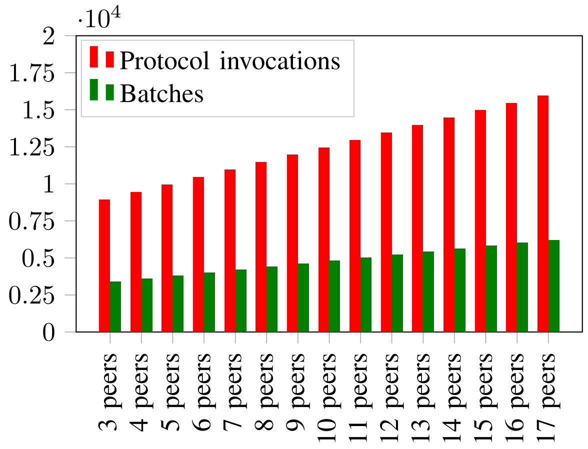

Figure 2. The amount of native protocol invocations and the number of batches in which they have been executed depending on the number of participating peers.

4.2. Network Latency

Another parameter evaluated in the study for this paper is network latency and how it affects the execution time of the application. In order to add latency `tc` was used. Figure 3 illustrates how additional network latency affects the execution time for the range of three to 17 peers connected to each other with a bandwidth of 1000 MBit/s. The increase in execution time in relation to network latency behaves similar to a root function. However, the growth in run time increases significantly. As an example, the execution time without additional latency is approximately under five seconds for the range of three to 17 peers, whereas a delay of 10ms results in an factor five to 17 growth in execution time for the corresponding number of peers. After this strong increase the slope flattens. Increasing added network latency from 10ms to 50ms (factor five) results in an approximately factor two increase in execution time.
Inferentially, it has to be stated that the network latency can result in a problem for SMC. Especially for mobile and Internet use cases this imposes a problem and possible limitations as in those cases a latency of 10ms is already quite optimistic. Hence, for such applications, latency can lead to an significant increase in execution time.

4.3. Transmission Rate

The last parameter we analyzed for this study was the transmission rate. In order to decrease the bandwidth from a maximum of 1000 MBit/s to a minimum of 1 MBit/s the same tool as in Section 4.2 was used (`tc`). Equal to the other measurements, the experiments were made with three to 17 hosts. Execution time was measured in the aforementioned range for 1 MBit/s, 10 MBit/s, 100 MBit/s and 1000 MBit/s. Figure 4 illustrates that there is no significant change in execution time between 1000

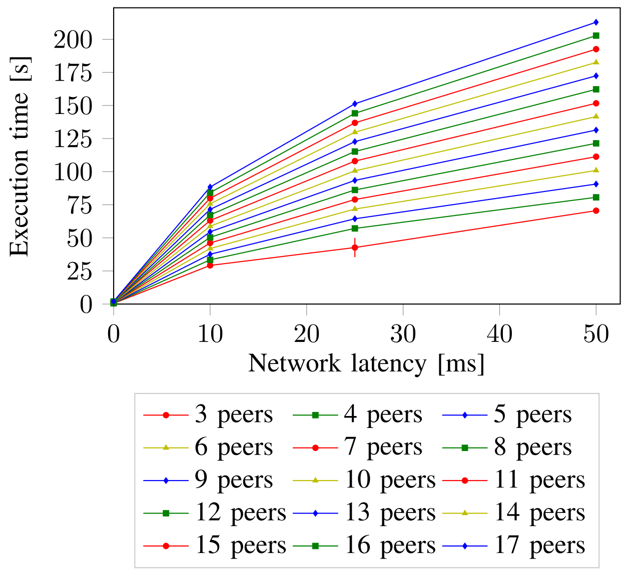

Figure 3. The execution time of the secure computation depending on the number of peers and on the network latency of the network. The latency highly influences computation time irrespective of the available transmission rate.

MBit/s and 100 MBit/s. However, from 100 MBit/s to 10 MBit/s there is a slight increase in time for execution. In the last interval, from 10 MBit/s to 1 MBit/s, the increase in execution time is much steeper, especially when the lower bandwidth is combined with a large number of participating peers. This behavior can be interpreted as follows: With 10 MBit/s and lower the transmission rate constitutes a bottleneck, but as soon as this threshold is exceeded there are only slight improvements in execution time.
Therefore, this may result in problems for large scale applications with a huge amount of hosts and some peers having lower bandwidths, e.g. mobile apps and Internet applications. However, for smaller networks with a high transmission rate between the peers, for example an intranet setup, the SMC approach remains more feasible.

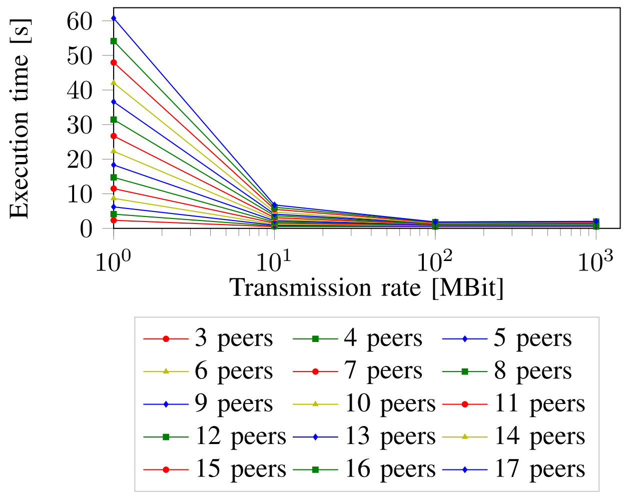

Figure 4. The execution time of the secure computation depending on the transmission rate. The shown case has no additional network latency.

5. Discussion

The purpose of this Section is to put the results from Section 4 in context and – as mentioned in Section 2 – compare them to the results of "A performance and resource consumption assessment of secure multiparty computation" [4].
As discussed in Section 4.1, the execution time increases linearly with an increasing number of peers in our case. The previously mentioned study came to the same results, which strengthens the thesis of having linear behavior.
For protocol invocations no comparable measurements were taken. In our case, we were able to also identify linear correlation between number of peers and protocol invocations.
Transmission rate can be considered a smaller problem. Both studies came to the result that a relatively low bandwidth of 1 MBit/s significantly increases the execution time, but even an increase to 10 MBit/s reduces time of execution drastically. Further increase in transmission rate only has a small effect.
In our study network latency constitutes the most important factor, as it had the largest absolute impact on the execution time. The related study supplies equal results, which strengthens this thesis.

Hence, for intranet applications, which are mostly connected with broad and fast connections, secure multiparty computation can be considered a viable option. Under the described circumstances, it is possible to keep execution time acceptably low. However, for applications with a less powerful network infrastructure, for example mobile apps, this can result in a high execution time, as the factors combine, i.e. a low transmission rate (EDGE: 384 KBit/s, 3G: 7.2 MBit/s (HSPA) [10]) and a higher network latency(EDGE: 200-450ms [11], 3G: 100-350ms [12]).

6. Conclusion

Summarizing, SMC provides a great software solution when several parties wish to collaboratively calculate a function, without either of them revealing their input. The theoretical concept already existed over 35 years ago, but recent advancements in computer and network technology allow practical implementations. However, there are still considerable limitations to secure multiparty computation. With state-of-the art technology and networks, intranet applications can already be considered feasible due to a limited amount of peers, low network latency and a high transmission rate. In such setups an acceptable execution time seems realistic. However, in use cases with less optimal circumstances there is a significant increase in execution time. While a decrease in transmission rate leads to an increase of up to approximately factor five, the influence of the amount of peers was at factor three from three to 17 participants. The greatest absolute increase in execution time came from the network latency. Even a small delay of 50ms increased the time of execution with up to a factor of over 100. With those limitations real-time or close to real-time applications are unfeasible at the moment. However, for use cases with softer time restrictions or faster infrastructure, secure multiparty computation can be feasible. With further advances in internet and network technology – increasing bandwidth in combination with lower network latency – SMC can exceed the limitations of today. Hence, it remains an interesting topic for further research and investigation.

References

[1] A. C. Yao. Protocols for Secure Computations. *Proceedings of the 23nd Annual Symposium of Foundations of Computer Science*, Washington, DC, USA: IEEE, pp. 1-5, 1982.

[2] A FRamework for Efficient Secure COmputation. https://www.github.com/aicis/fresco. 2018.

[3] Sharemind MPC. https://sharemind.cyber.ee/sharemind-mpc/.

[4] Marcel von Maltitz and Georg Carle. A performance and resource consumption assessment of secure multiparty computation. *CoRR*, abs/1804.03548, 2018.

[5] M. Burkhart, M. Strasser, D. Many and X. Dimitropoulos. SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics. *Proceedings of the 19th USENIX Conference on Security*, p. 15, 2010.

[6] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. *IACR Cryptology ePrint Archive*. Springer, 2008, no. October, p 289.

[7] FRESCO Documentation. https://fresco.readthedocs.io. 2018.

[8] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. *Lecture Notes in Computer Science*, vol. 7417, 2012, pp. 643–662.

[9] Bristol Cryptography Blog. https://bristolcrypto.blogspot.com/2016/10/what-is-spdz-part-1-mpc-circuit.html. 2016.

[10] Mobiles Internet. https://de.wikipedia.org/wiki/Mobiles_Internet. 2018.

[11] Alles zum Thema edge. https://www.onlinekosten.de/mobiles-internet/edge/.

[12] 3G/4G Ping Times/Latency. https://www.evdoinfo.com/content/view/4818/64/.