Making Secure Multi-Party Computation Scalable

Nicholas D. Gilbertson
gilb0578@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Secure multi-party computation, or SMPC for short, is a topic that has been around for decades, but has become more relevant with each passing year. Secure multi-party computation is the simple concept of computation involving multiple individuals without compromising the privacy of any individual's data. Due to this general definition, there are many real life uses for SMPC, as well as many different implementations of it. In this paper we'll take a look at many of the real world applications of SMPC, as well as take a deep dive into multiple of the modern implementations of SMPC and the current problems that they face. Finally, we'll look into possible solutions for the scalability issues that SMPC protocols suffer from. We'll look into a new protocol called nanoPI which aims to improve upon these scaling issues by using entirely different mechanics in multiple parts of the computation. The final solution to scaling issues that we'll cover is a query compiler called Conclave which achieves large improvements in efficiency for certain uses of SMPC.

Keywords: Secure multi-party computation, nanoPI, scaling, garbled circuit evaluation, secret sharing, Conclave, Query Compiler

1 Introduction

Secure multi-party computation (SMPC), also known as privacy preserving computation, and secure function evaluation, is a way for multiple participants to compute a shared result while keeping each individual participant's data completely private. SMPCs are decentralized net-based protocols where each participant in the SMPC completes a part of the computation. SMPC can receive input from any number of participants, and can be configured so that all or a subset of these participants receive the output. The privacy of the computation is preserved given that the number of corrupt parties involved in this computation doesn't exceed the SMPC protocol's given security threshold. [1]

Some examples of real world uses for secure multi-party computation include: privacy in processes such as auctions, secretive decision making in machine learning, use of third-party programs to compute something without the third-party having access to a participant's data, as well as the ability for data scientists to analyze a full set of data without compromising any individual's personal information. An auction is a good example of where every participant gives an input and every participant gets to see the output at each stage: the highest current bid. SMPC when used by data scientists or researchers for things such as financial or medical research is a great example of SMPC where only some of the participants get to see the output of the computation. In the case of medical research, the participants supplying the medical data don't need the outcome, so the only person who gets to see the result of the computation is the researcher.

We'll be looking at one popular implementation of SMPC to first understand the concept, and then will present the developing solutions to the scaling problems within SMPC.

The popular implementation of SMPC that we'll look into is Garbled Circuit Evaluation (GCE). GCE is a popular and relatively simple implementation of SMPC with a few different variations based on the needs of SMPC in different contexts. These differences stem from the needs of more or less security in different contexts.

The prototype scalable SMPC model we'll be looking in section 4.2 is called nanoPI. The currently popular implementations of SMPC scale very poorly. Because of this, they're unable to run computations on large scale data sets without losing significant levels of performance. NanoPI was developed to be efficient in time as well as space used, attempting to avoid both of the reasons that SMPC is inefficient with regards to large data sets.[7]

The alternate solution to SMPC scaling issues we'll be looking at in section 4.3 is a query compiler named Conclave which can be used to speed up pre-existing SMPC implementations. Conclave takes certain SMPC steps and runs them in the clear. This solution allows for parallel processing for parts of the SMPC computation, thus improving the scalability greatly.[3, 5]

Finally, we'll go over the results of the testing done with nanoPI as well as Conclave, and compare them with current SMPC statistics in order to come to a conclusion about whether or not there is a solution to the scaling issues within SMPC.

2 Background

2.1 Security classifications within SMPC

There are two main types of security that an SMPC protocol can achieve: Passive Security and Active Security. However before diving into a description of either, it's important to note that in some implementations of SMPC, the participants of the computation may be unknown. That means that it's

entirely possible to run into situations where multiple of the participants are both malicious, and communicating with one another. Because of this, when evaluating the security of an SMPC protocol, if there are multiple malicious parties involved in the computation, then we assume that they are cooperating and are able to share data. This is because multiple, cooperating malicious parties represents the worst case scenario, and a truly secure SMPC protocol will be secure even when faced with multiple malicious participants.

Passively secure multi-party computation defends against semi-honest adversaries, also known as honest-but-curious adversaries. A semi-honest adversary may be able to look at the internal states and other information about the SMPC process running on another participant's machine, however they do nothing to modify or affect the code running the protocol. Passive security means that even if a semi-honest adversary got a view into a part of the SMPC protocol that they aren't supposed to be able to see, they wouldn't be able to get any information about any of the private inputs involved in the computation.

While passive security offers an important layer of protection, it is by no means a fully secure protocol. Active security is the next step up, and protects against active adversaries, also known as Malicious Adversaries. An active adversary is someone who changes the protocol in some way to exploit a vulnerability, one which gets them information about the private inputs of the computation, or modifies the output so that it no longer reflects the actual result of the computation. Active security is the highest level of security for an SMPC protocol, and should in theory be completely secure.

Another type of security for SMPC worth mentioning is a concept called Covert Security. In some cases, especially when an active adversary is attempting to deviate from an SMPC protocol in order to change the output of the computation, knowing whether or not a protocol has been cheated is just as effective as being actively secure in the first place. In essence, if the participants of an SMPC know that the output has been tampered with, they clearly won't trust it. A protocol which doesn't necessarily offer full active security, but can detect with 99 percent plus probability when the protocol has deviated from, is covertly secure. In some cases, covertly secure SMPC protocols are more efficient than their actively secure counterparts, and can sometimes be a better fit for use cases while still offering the security that the output of the computation will be real. It's also important to note that both actively secure protocols and covertly secure protocols are also passively secure. [4]

3 Modern SMPC

3.1 Garbled Circuit Evaluation

Garbled circuit evaluation (GCE) is a two-party SMPC protocol used to privately evaluate a Boolean circuit. Given participants $participant_1$ and $participant_2$, they can evaluate a function $f(input_1, input_2)$ where $input_1$ and $input_2$ are the inputs of $participant_1$ and $participant_2$ respectively. In GCE one participant acts as the circuit generator and the other participant acts as the circuit evaluator. Assuming that $participant_1$ is the circuit generator, they will create a garbled representation of the circuit, which is achieved by associating each of the two possible values of the binary wires with randomized labels. The other participant is tasked with evaluating the GCE without having any understanding of the labels it uses to complete the evaluation. Once the evaluation is completed, the outputs can be decoded to show their results and can be given to one or both of the involved participants.[1]

Garbled circuit evaluation uses a form of the technique Oblivious Transfer called "1-out-of-2 Oblivious Transfer". This technique allows for the circuit evaluator to obtain the wire labels for their private inputs, and only for their private inputs. In the case of GCE, the sender in Oblivious Transfer is the circuit generator. The generator has two strings, $string_0$ and $string_1$, whereas the receiver (the circuit evaluator in this case) only has one bit b, which correlates with $string_0$ or $string_1$. By using Oblivious Transfer, the circuit evaluator is able to obtain string $string_b$ (either $string_0$ or $string_1$), while the circuit generator has no idea about which string was actually received. Now that the circuit evaluator has the string which correlates with the wire label for their inputs, they can evaluate their inputs over the circuit. [1]

In GCE a garbling algorithm $G()$ is a randomized algorithm which turns a function f into three separate functions; $G(f) \rightarrow (f_e, f_g, f_d)$. To be able to get the expected value, it's necessary that $f = (f_e \circ f_g \circ f_d)$ on any input. There is an encoding function f_e which allows us to evaluate a garbled input $f_e(input)=input_e$, where $input$ is the set of both participant's inputs. At this point using the garbled input $input_e$ and garbled function f_g, we are able to get a garbled output $output_g$. This results in the completion of the computation and Boolean circuit Using the decoding function f_d on the garbled output returns the final output $output_d$, which is equivalent to the result of $f(input)$ thus completing the GCE (see figure 1). [2]

It is required that $G()$ be randomized, so that there is no realistic deterministic way to predict the outputs (f_e, f_g, f_d) of $G(f)$. The garbled input $input_e$ is generated first so that neither participant is able to decipher the private data being used in the computation. This garbled input is also generated in such a way that when f_g is applied to $input_e$, it completes the intended Boolean circuit, but still returns an encoded result $output_g$ so that there is control over who is and who isn't able to receive the result. The decoding function f_d is only given to those who are the intended recipients for the result of the GCE. The oblivious transfer mentioned earlier is used when transferring data to the circuit evaluator in GCE,

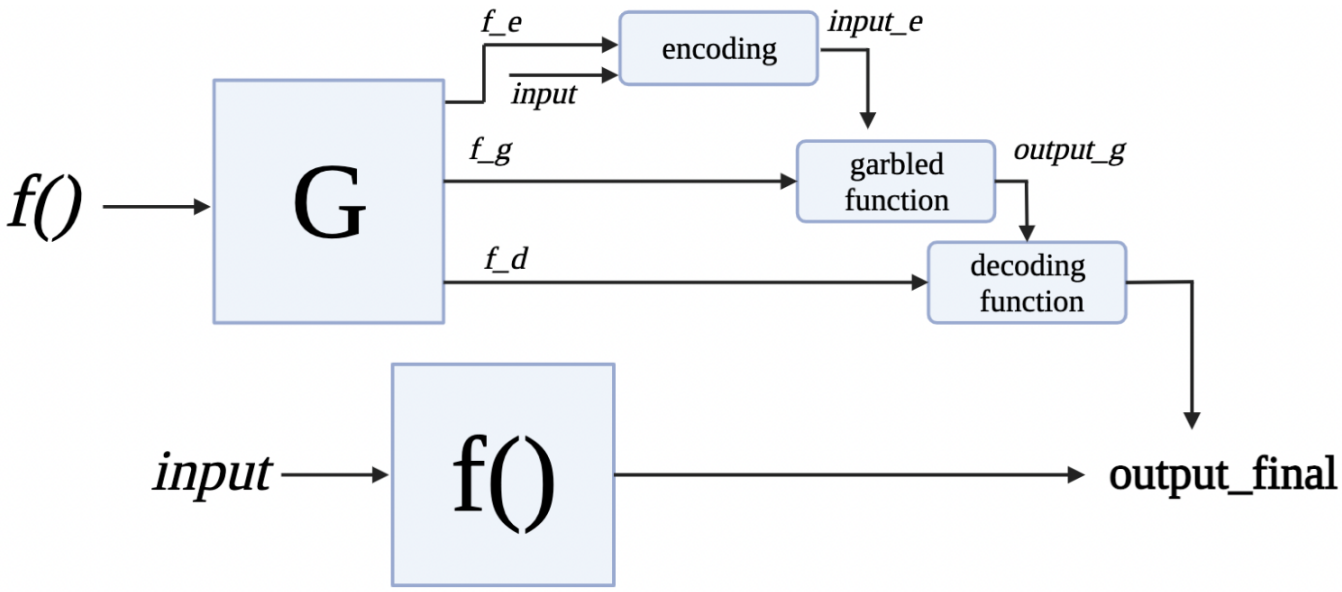

Figure 1. Garbling function G turns $f()$ into three components f_e for encoding the initial input to the garbled input, f_g for evaluating the garbled inputs over the garbled function, and f_d for decoding the garbled outputs to their final value, which is equivalent to running the original inputs over the original function.

so that the circuit generator isn't aware of which part of their data is being used in which part of the circuit evaluation.

Modern GCE is a powerful and flexible SMPC protocol, as newer implementations have been shown to work well with very detailed and complex functions, and also compute efficiently and effectively even for circuits including hundreds of millions of gates. [2]

4 Scaling Issues and Attempted Solutions

4.1 Scaling Issues

The main current issue with SMPC protocols is that they don't scale effectively. Current Active SMPC protocols need either linear space, linear rounds of computations, or both [7]. Linear in this case, meaning the size of the circuit representation. After much research and discussion, the best modern two-party SMPC protocols are able to evaluate over one hundred thousand AND gates per second, even in actively secure SMPC implementations that protect against active adversaries. A new garbling technique used in GCE developed by Wang et al. has made constant-round computations with n parties safe even when n-1 of the parties involved in the computation are malicious.[6] This is a powerful protocol, and can sufficiently meet the needs of many modern SMPC applications. However, while SMPC protocols have improved drastically in time efficiency, the statement that current active SMPC needs linear space or linear rounds is still true, meaning that current active SMPC protocols fail to scale on space efficiency.

This is troublesome for multiple reasons, the first being that computations done by SMPC are generally Boolean circuits, whose representations are fairly large. SMPC is also often used on incredibly large data sets, such as when completing computations over medical or financial data. The size of the data sets and the poor scaling of Boolean circuit representations will sometimes lead to SMPCs trying to run circuits with billions of gates. However, using SMPC over large scale data sets isn't the only case where its poor space scaling becomes an issue. There are many uses for SMPC on devices such as smart watches, Internet of Things devices, etc. however they would struggle with the space inefficiency seen in current SMPC.

The biggest reason behind the scaling issues within active SMPC comes from a trade that modern active SMPC protocols make for performance benefits. In order to be constant-round, these protocols trade the size of the circuit representation $|Circuit|$ space used at all times. Because of this, any actively secure SMPC protocols that have high time efficiency have a massive and poorly scaling space requirement. [7]

4.2 NanoPI

In order to attempt to tackle the issue of creating an actively secure SMPC protocol that scales well in both time efficiency and space efficiency, Ruiyu Zhu, Darion Cassel, Amyu Sabyr, and Yan Huang developed nanoPI [7]. NanoPI is based on WRK, which is one of the most efficient active SMPC protocols that remains secure against an arbitrary number of active adversaries. NanoPI was developed by looking for the

Memory Budget		20 MB		200 MB		2 GB	
Protocol		nanoPI	WRK	nanoPI	WRK	nanoPI	WRK
Speed (AND/s)	20 Mbps 40 ms	795.03	1.73K	2.73K	2.75K	3.12K	3.23K
	200 Mbps 40ms	825.18	2.76K	6.94K	12.94K	20.94K	22.38K
	2 Gbps <1 ms	20.27K	20.53K	46.66K	46.84K	49.34K	50.64K

Table 1. The tests were completed with 3 different memory budgets: 20MB, 200MB, and 2 GB. They were also tested in 3 different network environments: 20 Mbps 40 ms, 200 Mbps 40 ms, and 2 Gbps <1 ms. Square pretains to one of these memory budgets and one of these network environments, and the data being measured is the speed of the protocol in AND gates per second.

most space heavy steps in WRK, and attempting to lower their space requirements drastically. NanoPI has a modified version of WRK's protocol used for processing circuits, which executes binary gates in the protocol in batches, and deallocates binary wires which aren't currently in use. This saves vast amounts of space while still keeping a high level of efficiency.

NanoPI introduces a change to a sub-protocol of WRK called Authenticated Bit. When values are the same across different instances of the authenticated bit protocol, it batches these computations, in turn reducing the space requirement of this sub-protocol.

NanoPI also introduces a change to a sub-protocol of WRK called Authenticated AND. These changes stem from the use of a pool-based cut-and-choose system, which in some situations would always choose certain types of results from the pool. Therefore, in cases where these choices are present nanoPI is able to create as many Authenticated ANDs as needed while maintaining both time efficiency and constant space. [7]

The results of all these changes makes the space efficiency of nanoPI $O(u_n)$, which is much better than the previous $O(|Circuit|)$ space efficiency, where $|Circuit|$ is the size of the circuit representation and u_n is a user-set constant. The amount of rounds required became $O(n|Circuit|/u_n)$ where n is the number of participants. NanoPI has also been formally proven to be an actively secure implementation of SMPC. [7]

NanoPI's efficiency has been tested in practice, running a 4 party actively secure logistic regression with 4.7 billion AND gates in less than 28 hours on "mediocre machines" (4GB memory). NanoPI was also run on 40.8 billion ANDs and 122 billion XORs which took 16 days, but incredibly the peak memory usage for the entire computation was 398MB. This shows that nanoPI is able to keep up with the efficiency of modern SMPC protocols, while keeping strict memory usage.

When comparing nanoPI to its mother protocol, WRK, it is clear that WRK is still the fastest active SMPC protocol. The tests in table 1 were done with three different memory budgets: 20 MB, 200 MB, and 2 GB, as well as 3 different sets of network conditions: 20 Mbps/40 ms, 200 Mbps/40 ms, and 2 Gbps/<1 ms. NanoPI is still able to almost match the speed of WRK, especially when the network bandwidth is no longer a bottleneck, as seen in the 2Gbps row. While the changes made to the sub-protocols of WRK allowed nanoPI to scale better with space, they do require more computing power. Because SMPC is inherently a distributed net-based system, this increase in computing requirements lead to a bigger bottleneck when faced with poor network conditions than WRK had to face. NanoPI is still an advancement in SMPC technology, because it's the first active SMPC protocol which can run arbitrarily large circuits, whereas WRK can only run smaller circuits due to the requirement of holding the entire size of the circuit representation in memory at all times. [6]

4.3 Conclave

This entire section is derived from "Conclave: Secure Multi-Party Computation on Big Data" [5]. Another proposed solution to the SMPC scaling problems comes in the form of a query compiler prototype named Conclave, developed by Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Whereas nanoPI took a pre-existing SMPC protocol (WRK) and made changes to the sub-processes within the protocol in order to improve the space efficiency, the biggest changes Conclave makes to SMPC protocols is done outside of the SMPC itself.

Conclave takes advantage of the fact that often, many of the relational analytics queries performed within SMPC don't necessarily need cryptographic techniques in order to ensure total security. Conclave processes these queries in cleartext which greatly increases efficiency by allowing for parallel processing and smaller SMPC steps. There are also times when parties trust each other with subsets of the data being used in a computation. For example, when computing queries for medical research the only data that needs to be private during the computation is a patient's personal

information. Any other data doesn't need to be computed privately. When parties trust each other with a subset of data, Conclave also takes advantage of this by running steps on this data in the clear outside of the SMPC protocol. This hybrid SMPC setup caused by Conclave greatly improves scalability for cases where it's applicable.

In order to take advantage of Conclave, data analysts write relational queries for the data as if they had access to all of the inputs. Conclave takes these queries, and transforms them into a set of local processing steps and modified SMPC steps. The local processing steps are any data that can be computed in the clear, such as marked data in hybrid data sets, or parts of queries which don't reveal sensitive information. The modified SMPC steps are what compute all sensitive data privately. Through this method, Conclave is often able to return the results of an SMPC protocol within just minutes, even when the size of the input is magnitudes larger than some SMPC protocols can support due to space restrictions.

There are a couple of factors which help Conclave scale as successfully as it does. Conclave reviews the queries and modifies them in ways which improve efficiency without compromising security. However, this process burdens individual parties with more computational work in the short term than classic SMPC would. Conclave uses the optional hybrid process mentioned earlier, computing safe data many times more efficiently. The final factor that Conclave leverages is its combination of using fast but insecure data-processing systems for SMPC steps that need not be obscured and fully secure systems for SMPC steps that are required to be private.

Conclave only supports two SMPC frameworks, Obliv-C and Sharemind, meaning that Conclave only supports two-party and three-party SMPC. Using Obliv-C and Sharemind, Conclave is able to generate code for Garbled Circuit Evaluation, Secret Sharing SMPC frameworks, and data-parallel local processing systems. One important point to make about Conclave is that it is a passively secure protocol, only safe when in the presence of honest and semi-honest participants. Much like how nanoPI used WRK as a starting point for development, Conclave was developed with an SMPC protocol named SMCQL in mind. Conclave's Secret Sharing based back-end is more efficient when computing the arithmetic operations involved in the relational queries that it needs than SMCQL's Garbled Circuit back-end, helping Conclave outperform its closest competitor.

The performance of Conclave was measured using end-to-end analytics queries, as well as micro-benchmarks. Using minimal amounts of market insecure inputs, and data-sets which only take advantage of the basic optimizations Conclave introduces, Conclave was able to scale queries to inputs orders of magnitudes larger in size than current SMPC frameworks support. Taking advantage of Conclave's hybrid operations on non-private inputs, it was able to speed up important operators such as join and aggregate operators to 7 times faster (or more) even when compared to Sharemind, which is a fast and commercial framework used in real life applications of SMPC. When compared to SMCQL, Conclave's optimizations were able to help it scale medical research queries for inputs orders of magnitudes larger while still maintaining passive security.

5 Conclusion

NanoPI attempts to create a scaleable actively secure SMPC protocol, and while it is able to match the speed of its competitors in many situations, it wasn't able to achieve the large improvements in performance. The main advancement that nanoPI was able to achieve was the ability to run arbitrarily large circuits. Even if this advancement didn't translate into faster SMPC protocols in practice, it's a powerful advancement and may help achieve greater space/time efficiency for future SMPC solutions.

Conclave shows a far larger improvement in efficiency, with the achievement of speeding up the use of operators to at least 7 times faster than commercial SMPC frameworks, as well as being able to compute medical research queries with inputs magnitudes larger than competitors are able to compute. However, while Conclave has a high level level of efficiency, it has many limitations. The majority of the improvements in speed that Conclave is able to achieve are only achievable when a subset of the inputs can be marked as non-private and therefore can be computed in the clear. Conclave only supports two-party and three-party SMPC. Finally, Conclave only achieves passive security, meaning that it isn't suitable for any real life applications where participants of the SMPC may be untrustworthy. Conclave is a real advancement in SMPC, but only for situations such as medical research where there are both subsets of inputs which aren't required to be computed privately, as well as when malicious adversaries aren't likely to exist.

The solutions discussed in this paper both had large impacts on understanding the scalability issues within SMPC, as well as attempting to solve them. However, both nanoPI and Conclave are still only partial solutions. SMPC research has a long way to go for there to exist an actively secure framework which scales with both space and time efficiently.

Acknowledgments

I would like to thank my advisor Elena Machkasova for her continued support, advice, and revisions. I would like to thank Melissa Helgeson for her feedback and advice. Finally, I would like to thank the course instructor, K.K. Lamberty, for her guidance and feedback throughout the course.

References

[1] Fattaneh Bayatbabolghani and Marina Blanton. 2018. Secure Multi-Party Computation. In *Proceedings of the 2018 ACM SIGSAC Conference*

on Computer and Communications Security (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2157–2159. https://doi.org/10.1145/3243734.3264419

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of Garbled Circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 784–796. https://doi.org/10.1145/2382196.2382279

[3] David Byrd and Antigoni Polychroniadou. 2020. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. In *Proceedings of the First ACM International Conference on AI in Finance* (New York, New York) *(ICAIF '20)*. Association for Computing Machinery, New York, NY, USA, Article 16, 9 pages. https://doi.org/10.1145/3383455.3422562

[4] Claudio Orlandi. 2021. MPC techniques series, part 2: Security taxonomy and active security. https://medium.com/partisia-blockchain/mpc-techniques-series-part-2-security-taxonomy-and-active-security-6b5f14a15217

[5] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: Secure Multi-Party Computation on Big Data. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 18 pages. https://doi.org/10.1145/3302424.3303982

[6] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 39–56. https://doi.org/10.1145/3133956.3133979

[7] Ruiyu Zhu, Darion Cassel, Amr Sabry, and Yan Huang. 2018. NANOPI: Extreme-Scale Actively-Secure Multi-Party Computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 862–879. https://doi.org/10.1145/3243734.3243850