SAND2019-13692

SANDIA REPORT

Printed September 2019

Sandia National Laboratories

# Information-Theoretically Secure Distributed Machine Learning

Timothy M. Shead, Jonathan Berry, Cynthia Phillips, Jared Saia

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

## ABSTRACT

A previously obscure area of cryptography known as Secure Multiparty Computation (MPC) is enjoying increased attention in the field of privacy-preserving machine learning (ML), because ML models implemented using MPC can be uniquely resistant to capture or reverse engineering by an adversary. In particular, an adversary who captures a share of a distributed MPC model provably cannot recover the model itself, nor data evaluated by the model, even by observing the model in operation. We report on our small project to survey current MPC software and judge its practicality for fielding mission-relevant distributed machine learning models.

## CONTENTS

1. Motivation 7

2. Analysis 10

References 14

References 14

## 1. MOTIVATION

Consider a group of bureaucrats in a meeting, who want to know how much the meeting will cost to conduct. To do so, they need to sum the hourly pay rate of each meeting participant, and multiply the total by the meeting duration. One way to accomplish this would be for each person to announce their pay rate aloud to the room; however, we assume that most people (even bureaucrats) wish to keep this information private. An alternate approach would be to identify a person outside the group who is trusted by each participant. The participants privately reveal their pay rates to the trusted outsider, who completes the calculation and announces the results to the group. In this case everyone learns the result of the computation, while no one other than the trusted outsider learns any private information. Unfortunately, finding an outsider who is trusted by every participant will necessarily be difficult, with the difficulty increasing rapidly as the number of participants increases. Further, the trusted outsider could become a significant liability if they betray the trust of the meeting participants by revealing or misusing their private information, whether intentionally or by accident.

Fortunately, a third alternative already exists, using a previously obscure area of cryptography known as Secure Multiparty Computation (MPC), the fundamentals of which have been known for decades [15][11][7][6]. The core concept in MPC is *secret sharing*, where a piece of data - the *secret* - which is known to only one *player* in a group of *players*, is split into a set of *shares* that are distributed among all players. Combined, the shares reveal the secret entirely, but any proper subset of the shares provably reveals nothing. Conceptually, it is as if the secret has been written on a piece of paper which is divided into pieces and distributed to all of the players. Only when the players cooperate to combine the pieces can the secret be revealed. More practically, if the secret is a number, it can be split into a set of randomly-chosen numbers that equal the secret value when summed, a strategy known as *additive secret sharing*. A player with access to every share can sum them to reveal the secret, but so long as any of the shares is missing, the secret remains unknowable. Note that in secret sharing, unlike other forms of cryptography, there is no key to break.

MPC builds upon this foundation by introducing the ability to perform computations on secret shares. That is, MPC allows players to cooperatively compute sums, products, and other mathematical operations on shares without knowing the underlying secrets. For example, to compute the sum of two additively secret-shared numbers, each player simply adds their shares of the two operands. The result of this computation is an additive secret share of the result, so that the final sum is only knowable if all players agree to reconstruct it by combining their shares. Thus, players can exchange shares of their secrets and use them for computation, without revealing the original secrets or intermediate results to one another. Additional protocols to perform operations other than addition (most notably multiplication) are well understood and will be introduced shortly. Once a computation is complete, the players can agree to reveal their shares of the result to one, a few, or all players, so that only the players that receive every share learn the final result.

Returning to the meeting in session, we now have a solution that should be satisfactory to everyone: each bureaucrat secret shares their pay rate with the other participants (which does not reveal their private information) and the group uses MPC to collaboratively compute the cost of the meeting. No trusted third party is required, and the final result can be shared with the group without revealing the private information that was used to compute it. Problem solved, we will leave the bureaucrats to continue with the rest of their agenda.

Despite the powerful protections that MPC can afford, practical implementations have only appeared in the past few years, due to the hardware and communication resources required, which are considerable - note that, unlike normal computation, MPC requires significant network communication between players, both for the initial sharing of secrets, and every non-trivial mathematical operation thereafter. For these reasons, the earliest known practical application of MPC occurred in 2008, in a nationwide auction of sugar beet contracts conducted in Denmark [8]. Since then, there has been significant interest in the application of MPC in a variety of contexts, including privacy preserving machine learning (ML).

A privacy preserving ML algorithm using MPC is appealing because it would have the following, novel properties: First, the algorithm could be trained using MPC on secret-shared data, guaranteeing that players cannot access each other's training data, allowing them to pool information that they otherwise couldn't or wouldn't. Second, because the resulting trained model would itself be a shared secret, it could only be used when every player uses their shares of the model cooperatively. A compromised player cannot reveal the model or related details. Third, new observations from individual players can also be evaluated collaboratively without sharing: one player ("Alice") generates shares for a new observation and distributes them. Next, all players use their shares of the model and observation to jointly compute a prediction. Shares of the prediction are sent to Alice, who alone knows the result. A player that is compromised does not have access to Alice's observation, or the model's predictions thereof.

The desirability of the above properties was the motivation for our small project to determine whether MPC enabled machine learning had advanced to a point where it could be practically applied to national security missions. To do so, we conducted a software survey focused on two mission use-cases, which we will refer to as "institional" and "remote sensing":

In the institutional use-case, imagine a group of analysts who would like to maximize the utility of their data by pooling it to train ML models, but are prevented from doing so by need-to-know, policy, security, privacy, provenance, licensing, or other issues. As described above, using MPC enabled ML could make it possible for the analysts to create a joint model that benefits from the union of each analyst's data, while maintaining privacy. This case assumes a small number of players (analysts) working with large existing datasets located in datacenter environments.

For the remote sensing use-case, consider a sensor network using ML to make decisions, choosing when to discard observations, when to share them, where to focus the network's attention, and so on. In denied environments, nodes in the network may be captured or compromised by adversaries, and we want to minimize the damage if this happens. If the observations made by individual nodes and the network model are secret-shared, we can guarantee that a compromised node cannot be used to reveal the decision making model, nor exfiltrate observations from the other members of the network. This case assumes a relatively large number of players (sensor nodes) exchanging frequent observations over unreliable networks in challenging environments.

## 2. ANALYSIS

As alluded to earlier, MPC has unusual communication requirements compared to more traditional computing frameworks. First among these are the communication required to share secrets among players before computation can begin. Suppose a group of $N$ players wish to perform a computation requiring one input from each player. Since there will be a share of each secret for each player, the total number of messages exchanged in this case will be $N^2$, and this will be necessary *every* time a computation requires inputs from all players.

Second, MPC requires *per operation communication* for certain mathematical operations. In our earlier example using additive secret sharing, addition was a purely local computation, i.e. once the operands for the addition had been secret-shared among players, each player could compute their share of the result by simply adding the local shares. In this case (addition), communication is only necessary to reveal the results, and then only if the addition isn't part of some larger computation. The same is true for subtraction and, thanks to the distributive property, it is also true in the case where a secret-shared value is multiplied by a public value (a non secret-shared value known to all players): given a secret-shared value $\langle x \rangle$ and public value $\varepsilon$, $\varepsilon\langle x \rangle$ can be computed by having each player multiply their local share of $\langle x \rangle$ by $\varepsilon$.

Unfortunately, this is not the case for other operations, most notably multiplication of two secret-shared values. In this situation it is common to use *Beaver multiplication* [6], which requires a set of *triples* to perform the operation. Here, a *triple* is a set of three secret-shared values $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$ where $a$ and $b$ are chosen at random and $c = ab$. Given a triple, two secret-shared values $\langle x \rangle$ and $\langle y \rangle$ can be multiplied as follows:

$$\varepsilon = \langle x \rangle - \langle a \rangle$$

$$\delta = \langle y \rangle - \langle b \rangle$$

$$\langle x \rangle \langle y \rangle = \delta\langle a \rangle + \varepsilon\langle b \rangle + \langle c \rangle + \varepsilon\delta$$

Note that communication is required to secret share the triple before computation can begin, and that $\varepsilon$ and $\delta$, though computed locally using subtraction, are publicly shared with all players, requiring two additional rounds of all-to-all communication. The final step in the computation is purely local, since it uses only addition, secret $\times$ public multiplication, and public $\times$ public multiplication. Note that triples are a type of *one time pad* that cannot be re-used in subsequent calculations; thus, the rate at which multiplications can be performed is limited by the rate at which new triples can be generated. It would be difficult to overstate the performance implications, as the communication involved is orders of magnitude slower than traditional computing involving modern CPUs performing on-chip calculations.

Nevertheless, once addition and multiplication have been defined, it becomes possible to implement nearly any algorithm using MPC, including ML algorithms. In practice, and unlike more typical computing tasks, an MPC computation is usually performed using a single program that runs simultaneously on multiple hosts, one for each player. In this respect, MPC programs are quite similar to programs written using the Message Passing Interface (MPI) [2] which is a de facto standard in high performance computing.

Much recent work in MPC is focused on addressing the bottlenecks, leading to an explosion in the number and variety of protocols, each typically optimized for a specific use-case, such as the type of secret sharing, the number of players (sometimes just two or three) and a specific set of supported computations. As one example, the SecureNN [16] protocol expands upon the popular SPDZ protocol [10] to provide efficient secret-shared addition and multiplication of matrices and a secret-shared approximation to the nonlinear ReLU function, which are important building blocks needed to create privacy preserving neural network models.

In addition to performance challenges, MPC introduces new opportunities for failure - data that is distributed among players using additive secret sharing is vulnerable to the loss of any one player, becoming unrecoverable by design. This can be a blessing or a curse, depending on circumstance: for a datacenter environment and the institutional use case this tradeoff would likely be acceptable, since the hardware can be monitored and maintained to reduce the likelihood of failure, and restarting a failed computation can be relatively easy. For the remote sensing use-case, the ramifications of MPC integration are more significant: here, devices can fail, run out of power, or be lost; and radio networks are inherently unreliable and vulnerable to range, line-of-sight, atmospheric conditions, or other issues, even when hardware is functioning properly. Alternative secret sharing techniques can (partially) address this issue, by making it possible to recover a secret using a strict subset of all players. For example, the secret sharing scheme of Shamir [15] represents a secret as a point on a randomly chosen degree-$M$ polynomial, distributing additional points from the same polynomial as shares. To recover the polynomial (and thus the secret) requires a minimum of $M$ shares (points). So long as $M <= N$, then any subset of $M$ players can recover the secret, and the system can absorb the loss of up-to $N - M$ players before the secret becomes unrecoverable.

In parallel with new protocols, many new MPC implementations have appeared in the past few years. We divide these implementations into three categories: domain-specific languages (DSLs), add-ons to existing machine learning platforms, and commercial MPC platforms. Since the focus of this project was on the practicality of MPC, we examined several of these implementations in detail.

For an overview of MPC DSLs, see [12]. Typically, these languages support general-purpose MPC computation by supplementing traditional computer programming language syntax with MPC-oriented language primitives such as explicit type checking for public and private (secret-shared) values. As a representative example, the following code fragment computes the inner product of two secret-shared values using SCALE-MAMBA [3], which implements a runtime virtual machine executing bytecodes compiled from a subset of the Python language:

```
sum = sint(0)
for i in range(3):
    x1 = sint(i)
    x2 = sint(i*2)
    prod = x1 * x2
    sum = sum + prod
print_ln("%s", sum.reveal())
```

While the syntax is recognizably that of Python, note the addition of the nonstandard *sint* type, which is used to store a secret-shared integer. By explicitly distinguishing secret-shared values from public values, the code can detect and eliminate a large class of progamming errors, while hiding many of the details of MPC computation behind traditional arithmetic syntax. In the final line of code, the *reveal* method gathers the shares of the secret shared “sum” variable, combining them to reveal the final answer. Like most of the solutions surveyed, SCALE-MAMBA provides a mechanism for running a single program across multiple processes, one process for each player.

A majority of the DSLs were based on C or C++ syntax, with custom compilers and varying support for number of players, MPC protocols, operations, and-so-on. It was clear that the DSLs were primarily research codes, poorly documented, with extremely limited support for I/O, making integration into real-world workflows difficult at best. We were sorry to see that none of the DSLs examined recognized the similarities between MPC and MPI computing, as many practical solutions developed in the HPC community (API design, job startup, batch management tools) could be extremely useful in managing MPC computations. Notably, none of the DSLs we examined had support for exceptions or other application layer error-handling mechanisms: that is, the implementations assumed at a language level that an operation (such as multiplication) would always succeed. Even SCALE-MAMBA - which has optional support for Shamir secret sharing and in theory could continue computation despite the failure of a subset of players - fails immediately when any one of its player processes stops functioning. While this assumption might be acceptable for computations running in a datacenter for the institutional use-case, we considered it a fatal flaw for the remote sensing use-case where nodes or networking could fail at any time, including in the middle of a mathematical operation.

We then turned to several add-ons for existing machine learning platforms, beginning with Tensorflow Encrypted (TFE) [9], which implements a variant of the SPDZ protocol [10] atop the Tensorflow toolkit [5]. Unlike the previously described DSLs where MPC players are peers, the TFE implementation uses a fixed-topology network consisting of two MPC servers and a cryptographic server responsible for generating Beaver triples. Rather than sharing secrets with each other, players in a TFE computation are clients, secret-sharing their data with the two servers, which are solely responsible for performing MPC computations and returning results. TFE does include much more flexible I/O capabilities, thanks to the Tensorflow toolkit upon which it is built. Unfortunately, like the DSLs we had examined previously, the loss of any TFE process caused the immediate failure of all others. Further, because Tensorflow (and thus TFE) is based on a static computational graph that is defined before computation begins, it is extremely impractical to react to errors at an application level. Thus, we found that while TFE could be a workable solution for the institutional use-case and a comfortable experience for existing Tensorflow users, it wasn’t a realistic architecture for the remote-sensing case.

The PySyft [14] project adds cryptographic computing to the popular PyTorch [13] toolkit and has aspirations similar to those of TFE, but their MPC implementation was not working at the time we examined it.

Sharemind [4] is a commercial platform that uses secret sharing to encrypt data stored in a distributed database and a proprietary DSL for specifying MPC computations. It did not include ML capabilities at the time of our study.

Fortuitously, and three weeks after the formal end of this project, the Crypten [1] project was released by Facebook Research as open source software, also providing MPC computation atop PyTorch. Importantly, while Crypten doesn't explicitly support unreliable hardware or communications out-of-the-box, it is the first MPC toolkit we've examined whose design does not preclude the kind of application level error handling necessary to address those challenges head-on. Notably, the Crypten and PyTorch APIs are explicitly aware of and inspired by MPI communication patterns, opening up interesting avenues for future integration and reducing programmer cognitive load.

Due to its importance as a mission use case and its lack of support in the current MPC literature and implementations, we plan to focus our future work on the remote sensing use case, using Crypten for our development. We have also begun development of an ML focused MPC protocol of our own with an emphasis on low bandwidth and high reliability, to be published in the coming year. For agencies interested in the institutional use case, we would recommend that developers use either Crypten or Tensorflow Encrypted, based on their personal affinity for PyTorch or Tensorflow respectively, as both provide a rich set of MPC-enabled ML primitives.

## REFERENCES

[1] Crypten. `https://github.com/facebookresearch/CrypTen`. Accessed: 2019-11-04.

[2] Mpi forum. `https://www.mpi-forum.org`. Accessed: 2019-11-04.

[3] Scale-mamba software. `https://homes.esat.kuleuven.be/~nsmart/SCALE`. Accessed: 2019-10-29.

[4] Sharemind technical overview. `https://repo.cyber.ee/sharemind/www/files/technology/sharemind-technical-overview.pdf`. Accessed: 2019-09-16.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, pages 420–432. Springer, Berlin, Heidelberg, Berlin, Heidelberg, August 1991.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[8] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 325–343, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[9] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. Private machine learning in tensorflow using secure computation. *arXiv preprint arXiv:1810.08130*, 2018.

[10] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[12] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 479–496, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[14] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

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

[16] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. *IACR Cryptology ePrint Archive*, 2018:442, 2018.

## DISTRIBUTION

### Hardcopy—External

| Number of Copies | Name(s) | Company Name and Company Mailing Address |
|---|---|---|
| | | |

### Hardcopy—Internal

| Number of Copies | Name | Org. | Mailstop |
|---|---|---|---|
| | | | |

### Email—Internal [redacted]

| Name | Org. | Sandia Email Address |
|---|---|---|
| Technical Library | 01177 | libref@sandia.gov |