CSEP 590: Applied Cryptography University of Washington
Prof. Rachel Lin Final Report, Spring 2023

An Exploration of MPC: Current Implementations and Challenges

Karuna Sagar Krishna (kskrish)
Sam Kirsch (kirschsd)

Introduction

Multi-party computation (MPC) is the process by which a set of independent entities all possessing data on some subject can get the results of some operation performed over the collective data, without any of the private input data being shared. This means that entities that have privacy or security concerns about their data can collaborate to produce statistical results or models that are more powerful than any single entity could produce with its data alone. MPC is a powerful technology that has only recently begun to show its potential payoffs in the applied domain.

Project goal

As engineers we are interested in the practical applications of MPC and as computer science students we are also interested in studying the details of MPC protocols. So we explore both these aspects in our project.

We start with understanding how close MPC is to being readily useable by engineers in an accessible and abstracted format, where the cryptographic expertise is largely guaranteed by "under the hood" functionality. As engineers ourselves, the progress towards generalized, performant toolkits for MPC is the metric we are most interested in, as barring some very specialized projects, having ready-made tooling for MPC is what would enable us to use this fantastic technology in our day-to-day work.

We then explore secret sharing, a building block used in various MPC protocols. We also study couple of fundamental MPC protocols - GMW and BGW protocols. We'll look at how these protocols work and how they guarantee privacy and correctness.

Finally, we wrap up with few reflections and takeaways from our project.

Historical applicability of MPC

MPC has classically suffered on the practical front because the requirements for implementing protocols spanned multiple broad areas of computer science [4], requiring expertise in cryptography, systems design, communication stacks, and others. As a technology, MPC has incredible potential applications that solve many modern data privacy issues. MPC essentially allows data maintainers and processors to have their cake and eat it too: you can get all the benefits and statistical strength of a massively centralized data store while still allowing disparate entities who have collected and maintained the data full ownership and control of the input data itself. Current questions about ownership of training data in large ML models could be greatly simplified by the secret nature of MPC inputs, and siloed data hordes held by jealous private entities could be used to produce valuable results without requiring the private entities give up their singular access to their data.

An especially interesting element of MPC research is that much of the formalization of the field was done well before there were many inklings of practical applicability over 30 years ago now. The origin of the modern understanding of MPC is frequently attributed to Andrew Yao's seminal papers in the 80s [1, 5], where he outlined how two (or m) parties could perform shared computation on secret values without those secrets being known to either party, until both parties shared the secrets.

Expansion of Yao's work by Lindell and Pinkas

Standing out amongst follow-up research to Yao's work was done in 2008, where Lindell and Pinkas expanded Yao's garbled circuits to work for malicious adversaries, instead of just dishonest ones [3]. The main issue moving from a simply dishonest, but passive, adversary using garbled circuits to an actively malicious adversary is that in the scenario since garbled circuits requires that one party constructs that circuit, a malicious constructor could build the circuits in such a way that the result obtained by the solver of the circuits is incorrect (”maliciously guided”).

Lindell and Pinkas solved this by creating a solution similar to the one we saw in class for the intuitive example of zero-knowledge proofs. They leveraged what they called a ”folklore” technique (adorable) called ”cut and choose”, where the producer of the garbled circuits would create many versions of the circuits, and then the solver would randomly ”open” and interrogate some subset sections of them. If the circuits behaved as they were supposed to under interrogation in all cases, the solver could with high confidence assume that the producers was being honest. The simple ”cut and choose” explanation is intuitively satisfying, but the most compelling part of Lindell and Pinkas's work is to identify the many subtleties that must be addressed in an actual implementation to avoid creating security or privacy holes: requiring that the solver evaluate and validate many versions of the garbled circuit could effectively give the producer an ”input oracle” into the solver's input, which to preserve privacy in the protocol must be kept secret. understanding

Investigations into Current Implementations

MPC as an applied area of research is fairly recent [2], with a large increase in projects cropping up attempting to create a ”generalized toolkit” in the last 5 years [7, 8, 9, 10, 11]. Most current practical implementations of MPC rely on more permissive versions of the MPC adversary threat model that we discussed in class. In scenarios where the protocol allows for a dishonest majority (where there are n participants and $n-1$ dishonest participants), in all implementations we reviewed it was only if the adversaries are simply passively dishonest. For implementations that claimed security against an actively malicious adversary, they generally required that the majority of adversaries be honest [7, 8, 9]. Interestingly, in many implementations, this weaker security model is justified by something along the lines of ”being standard in the literature”, aka ”that's how everyone else does it”. [11] This is definitely an element of modern MPC implementation that needs to be developed further, as currently this tends to limit real-world applications to scenarios where an overarching system provides the impetus for honesty (like laws), or the cost of data being exposed is not high (like peoples' genes, only because no one knows the true cost of leaking your genome yet).

One especially interesting implementation that we investigated is Sequre, which aims to be an accessible, high-level MPC framework to enable operations over biological data for things like genetic research. [10] Sequre is an implementation of MPC using additive secret sharing and abstracts data to elements of algebraic data (we can't claim to fully understand Ring Theory, but at least now we're aware of it). They do rely on a majority-honest, passive adversary model (fairly weak), but this does seem appropriate to their use case and risk profile (creating classifier models using genomic data). It also incorporates a shared "trusted dealer" construct to provide correlated randomness to computational parties engaging in oblivious transfers, allowing clients to pre-compute necessary values for later OTs, making the part of the computation that is performed online much more efficient. [12] The program uses metaprogramming decorators to indicate segments of computation that need to be performed securely over MPC, and also leverages a specialized implementation of Python called Codon that targets AOT compilation performance, which makes sense since the best-case deployment strategy for a Sequre implementation would be to distribute the pre-compiled Sequre executables to dockerized client environments where they can perform their computation. Overall, Sequre seems fairly easy to use (taking a look at their example code on github), and while its performance still does not approach classical, non-secret computations, it is magnitudes of 10-100x better than other available frameworks.

Fundamental MPC Protocols

In class we looked at Yao's Garbled Circuits protocol which allowed for 2 parties to securely compute a boolean circuit. Below tables lists few other MPC protocols with different properties and guarantees [13, 14]. Both GMW and BGW protocols use a secret sharing scheme based on which they perform secure computation while maintaining certain invariants that help achieve privacy and correctness.

Protocol	**Circuits**	**Parties**	**Adversary capacity, power**	**Rounds**
Yao's garbled circuits	Boolean	2	threshold capacity, bounded adversary	constant
GMW protocol	Boolean or Arithmetic	n	threshold capacity, bounded adversary	circuit depth
BGW protocol	Arithmetic	n	threshold capacity, unbounded adversary	circuit depth

Secret Sharing

A secret sharing scheme is a pair of algorithms $(Share, Recover)$. The $Share(s)$ algorithm takes a secret s and produces n random looking shares $[s]_i$ and privately distributes this with party P_i, which can be achieved using Authenticated encryption channels. The $Recover(\{a : a \in [s]\})$ takes in a subset of $t+1$ (or more) secret shares and recovers or reconstructs the secret s. Such a scheme is called (n,t) secret sharing scheme since we break up a secret into n parts and any $t+1$ such parts can be used to reconstruct the secret. Also such a scheme should be randomized so that any party holding t shares cannot learn anything about the secret s.

A secret sharing scheme is linear if the shares are computed as a linear function of secret s and some random values. More concretely, a secret share can be expressed as $[s]_i = c_{i0}s + c_{i1}r_1 + c_{i2}r_2 + \cdots + c_{in}r_n$ where s is the secret, r_i are random values and c_{ij} are publicly known constants. Such schemes are also said to be additive and homomorphic. Such linear secret sharing schemes (LSSS) allows one to efficiently compute a linear function by locally applying the same linear function on individual shares. Further, with some interaction they can also be used to compute non-linear functions as well. Below are few examples of linear secret sharing schemes that are both correct and preserve privacy of the secret.

Additive secret sharing for binary inputs used to evaluate boolean circuits

Consider a binary valued secret as $s \in \{0,1\}^k$. Say we want to generate a $(n, n-1)$ secret share i.e. full threshold secret shares. To do this we can randomly generate $n-1$ secret shares as $[s]_i \in \{0,1\}^k$ for $i > 1$. The share $[s]_1$ is computed as $s \oplus (\oplus_{i>1}([s]_i))$. To recover the secret s, we need all n parties to come together and the recover algorithm simply xor-s all the secret shares to recover secret s.

The correctness of this scheme is obvious. This scheme maintains privacy of s. All secret shares (except $[s]_1$) are randomly and independently generated. Secret share $[s]_1$ is also random and independent of s since the secret s is xor-ed with random values (this is the same as the masking trick we used in CTR mode encryption in class).

Additive secret sharing for group element used to evaluate arithmetic circuits

Consider a secret s as group element G. A $(n, n-1)$ secret sharing scheme would be - generate random $n-1$ secret shares as $[s]_i \in G$ for $i > 1$. The share $[s]_1$ is computed as $s - \sum_{i>1}([s]_i)$. To recover the secret s, we need all n parties to come together and the recover algorithm simply adds all the secret shares to recover secret s.

The correctness of this scheme is obvious and based on group properties of closure and existence of inverse of each group element. This scheme also maintains privacy of s since each share (except $[s]_1$) are randomly and independently generated. And the secret share $[s]_1$ is also random and independent of s since it is a linear combination of secret and random values.

Shamir's secret sharing scheme used to evaluate arithmetic circuits

Consider a secret s as field element F. Unlike previous schemes, Shamir's secret can produce a (n, t) shares for any $t < n$. So to recover the secret we need a subset of any $t+1$ secret shares. The $Share(s)$ algorithm takes in a secret s and generates a random polynomial $f(x)$ of degree t such that $f(0) = s$. All co-efficients of the polynomial are generated randomly from a field F, except the constant term which would be the secret s. The secret shares are then computed as $[s]_i = f(x_i)$ for $x_i \in F$. The x_i are called evaluation or interpolation points and are publicly known. The secret share $[s]_i$ are shared privately to P_i. To recover the secret s from subset of $t+1$ shares, we use Lagrange's interpolation formula which represents the polynomial as $f(x) = \sum_{i=1}^{t+1} \delta_i(x) f(x_i)$ where $f(x_i) = [s]_i$ and $\delta_i(x) = \prod_{j=1 \wedge j \neq i}^{t+1} [(x - x_j)/(x_i - x_j)]$.

The correctness of this scheme is based on Lagrange's interpolation formula. This scheme maintains privacy of s since each share is the evaluation of the random polynomial $f(x)$. Though the polynomial is evaluated at a certain publicly known evaluation points, since $f(x)$ is a random polynomial the outputs are random and independent of s.

GMW Protocol

This protocol was proposed by Goldreich, Micali and Wigderson in late 80s [15]. This is one of the first protocols to address generic multi party computation problem. Though the protocol can work with both boolean and arithmetic circuits, we'll look at boolean circuits below.

Consider that we want to securely compute a publicly known boolean circuit $\mathcal{C}$ whose inputs are held privately by each party $P_1, P_2, \ldots P_n$. Each party P_i holds a private input $x_i \in \{0,1\}^k$.

The protocols starts by having the parties share their inputs using a secret sharing scheme. So for example, P_i will get a share of x_j from P_j as $[x_j]_i$. Then each party computes the circuit $\mathcal{C}$; gates such as NOT and XOR can be computed locally while AND gate requires interaction between the parties. Note, AND and NOT gates together for a complete/universal set i.e. these 2 gates can be used to construct any other boolean gate. This protocol maintains the following invariant for each gate - if gate inputs are random (n, t) secret shares then the gate outputs are random (n, t) secret shares. This ensures privacy of the inputs and intermediate values during the secure computation of $\mathcal{C}$. Finally, each party would have a share of the circuit output which can be publicly announced to reconstruct the final answer/output.

The secret sharing scheme used here is the full threshold, additive homomorphic scheme described above. So, each party P_i secret shares x_i with all other parties.

NOT gate: Each party P_i has a secret share $[y]_i$ of secret y and needs to output z. Party P_1 outputs $[z]_1 = \neg[y]_1$, while all other parties P_i output $[z]_i = [y]_i$. It is straightforward to see if we run the *Recover* algorithm of secret sharing scheme we would indeed recover $z = \neg y$. Note, $[z]_i$ are random secret shares.

XOR gate: This is simple since the sharing scheme using XOR operation. The gate takes two inputs x and y. So each party P_i has secret share of these inputs say $[x]_i$ and $[y]_i$ and it outputs $[z]_i = [x]_i \oplus [y_i]$. Recovering the output z shows that $z = x \oplus y$. Note, $[z]_i$ are random secret shares.

AND gate: Unlike NOT and XOR gate computation described above, AND gate requires interaction between parties. The gate takes two inputs x and y. So each party P_i has secret share of these inputs say $[x]_i$ and $[y]_i$. Note, AND is distributive over XOR.

$$\begin{aligned}
z &= x \wedge y \\
&= x \wedge (\oplus_{i=1}^{n}([y]_i)) \\
&= (x \wedge [y]_1) \oplus (x \wedge [y]_2) \oplus \cdots \oplus (x \wedge [y]_n) \\
&= ((\oplus_{i=1}^{n}([x]_i)) \wedge [y]_1) \oplus ((\oplus_{i=1}^{n}([x]_i)) \wedge [y]_2) \oplus \cdots \oplus ((\oplus_{i=1}^{n}([x]_i)) \wedge [y]_n) \\
&= (\oplus_i([x]_i \wedge [y]_i)) \oplus (\oplus_{i \neq j}([x]_i \wedge [y]_j)) \\
&= (\oplus_i([x]_i \wedge [y]_i)) \oplus (\oplus_{i<j}(([x]_i \wedge [y]_j) \oplus ([x]_j \wedge [y]_i)))
\end{aligned}$$

Clearly, party P_i can locally compute $\oplus_i([x]_i \wedge [y]_i)$ without any interactions. However, P_i doesn't have all the secret shares to compute $\oplus_{i<j}(([x]_i \wedge [y]_j) \oplus ([x]_j \wedge [y]_i))$. However, using 1-out-of-4 Oblivious Transfer (1-4 OT) protocol, we can compute this value.

In 1-4 OT protocol, party P_j presents 4 random value and party P_i provides 2-bit input using which one of the 4 random values sent by P_j is chosen and returned to P_i. Specifically, P_j generates random value $r_{ji} \in \{0,1\}^k$ and presents the following 4 random values which correspond to lookup table for $\oplus_{i<j}(([x]_i \wedge [y]_j) \oplus ([x]_j \wedge [y]_i))$ given $[x]_i$ and $[y]_i$. So P_i would present 2-bit input as $([x]_i[y]_i)$. With this 1-4 OT transfer would return $r_{[x]_i[y]_i}$ to P_i.

$$\begin{aligned}
r_{00} &= r_{ji} \oplus ((0 \wedge [y]_j) \oplus ([x]_j \wedge 0)) \\
r_{01} &= r_{ji} \oplus ((0 \wedge [y]_j) \oplus ([x]_j \wedge 1)) \\
r_{10} &= r_{ji} \oplus ((1 \wedge [y]_j) \oplus ([x]_j \wedge 0)) \\
r_{11} &= r_{ji} \oplus ((1 \wedge [y]_j) \oplus ([x]_j \wedge 1))
\end{aligned}$$

So for every $i < j$, the above 1-4 OT transfer is performed where P_i receives $r_{[x]_i[y]_i}$ and P_j records r_{ji}. Each party then computes $[z]_i = ([x]_i \wedge [y]_i) \oplus (\oplus_{i<j} r_{[x]_i[y]_i}) \oplus (\oplus_{i>j} r_{ij})$. Finally, $[z]_i$ is a random secret share of $z = x \wedge y$ because

$$
\begin{aligned}
z &= \oplus_i [z]_i \\
&= \oplus_i(([x]_i \wedge [y]_i) \oplus (\oplus_{i<j} r_{[x]_i[y]_i}) \oplus (\oplus_{i>j} r_{ij})) \\
&= \oplus_i(([x]_i \wedge [y]_i) \oplus (\oplus_{i<j}(r_{ji} \oplus ([x]_i \wedge [y]_j) \oplus ([x]_j \wedge [y]_i))) \oplus (\oplus_{i>j} r_{ij})) \\
&= \oplus_i(([x]_i \wedge [y]_i) \oplus (\oplus_{i<j}(([x]_i \wedge [y]_j) \oplus ([x]_j \wedge [y]_i)))) \\
&= (\oplus_i([x]_i \wedge [y]_i)) \oplus (\oplus_{i \neq j}([x]_i \wedge [y]_j)) \\
&= x \wedge y
\end{aligned}
$$

BGW Protocol

This protocol was proposed by Ben-Or, Goldwasser and Wigderson in late 80s [16, 17, 18, 19, 20]. This protocol allows us to securely compute arithmetic circuits over a finite field F. Consider that we want to securely compute a publicly known arithmetic circuit $\mathcal{C}$ whose inputs are held privately by each party $P_1, P_2, \ldots P_n$. Each party P_i holds a private input $x_i \in F$. The circuit $\mathcal{C}$ is made up of addition and multiplication gates. The protocol maintains the following invariant for each gate - if gate inputs are random (n, t) secret shares then the gate outputs are random (n, t) secret shares. This ensures privacy of the inputs and intermediate values during the secure computation of $\mathcal{C}$. Finally, each party would have a share of the circuit output which can be publicly announced to reconstruct the final answer/output.

The secret sharing scheme used here is Shamir's secret sharing scheme described above. So each P_i shares its private input x_i as (n, t) secret shares where $2t < n$. This is called honest majority and is a necessary condition for any generic perfectly secure (i.e. targeting unbounded adversary) MPC protocol.

Addition gate: To securely compute addition of two private inputs a and b, the parties share their inputs using Shamir's secret sharing scheme. So each P_i has share $[a]_i$ and $[b]_i$. Then each party P_i locally computes $[c]_i = [a]_i + [b]_i$. If we recover the output c from t secret shares $[c]_i$, we get $c = a + b$. This is because $[a]_i$ are evaluation of some random polynomial $A(x)$ and $[b]_i$ are evaluation of some random polynomial $B(x)$. So $[c]_i$ are evaluation of $C(x) = A(x) + B(x)$. Since $[a]_i$ and $[b]_i$ are random secret shares, $[c]_i$ are also random (n, t) secret shares.

Addition with constant: A specific case of addition gate is when one of the inputs is a publicly known constant k. Here each P_i locally adds k to its secret share $[a]_i$ to get $[c]_i = [a]_i + k$. Note, $[c]_i$ are random (n, t) secret shares. Recovering c give us $a + k$ as expected.

Multiplication with constant: This is similar to addition with constant where we have a publicly known constant k. Each P_i locally computes $[c]_i = [a]_i \cdot k$. Note, $[c]_i$ are random (n, t) secret shares. Recovering c give us $a \cdot k$ as expected.

Multiplication gate: Using the same technique as above, each P_i can compute $[a]_i \cdot [b]_i$ which are secret shares of $c = a \cdot b$. However, there are couple of problems that violate the gate invariant:

- $[c]_i$ are not random since the corresponding polynomial $C(x) = A(x) \cdot B(x)$ is not random. Since $C(x)$ is reducible i.e. it can be expressed as product of $A(x)$ and $B(x)$, it may leak information about a and b. [18] has an example that demonstrates this leak. This impacts the privacy of the protocol.

- $[c]_i$ are $(n, 2t)$ secret shares. This is because both $A(x)$ and $B(x)$ are degree t polynomials hence their product would be a degree $2t$ polynomial. So if $2t \geq n$, then we will not have sufficient points to uniquely interpolate using Lagrange's formula thus affecting the correctness of the protocol. Further, this is also a problem when there are more gates to be evaluated after this multiplication gate where the degree of the output polynomial would increase further. [18] demonstrates this with an example.

To address these problems, the protocol uses a degree reduction method. There are couple of degree reduction methods - original BGW method [21, 22] and GRR reduction method [20]. The complexity of this protocol largely depends on the complexity of degree reduction method used.

The original BGW method of degree reduction has 2 steps - randomization and truncation, where each step addresses the 2 problems independently. The randomization step involves generating a random degree $2t$ polynomial $R(x)$ such that the constant term is zero i.e. $R(0) = 0$. This is added to the non-random degree $2t$ polynomial $C(x)$ to produce $C'(x) = C(x) + R(x)$ and we have seen how to compute this securely. So $R(x)$ needs to be $(n, 2t)$ secret shared among all parties and then we know how to perform addition without any interaction. So the resultant secret share of $C'(x)$ are random and hence addresses the first problem. The truncation steps addresses the second problem by converting $(n, 2t)$ secret shares to (n, t) secret shares or in other words converting degree $2t$ polynomial to degree t polynomial. As the name suggests, this involves only keeping t co-efficients and removing the others. In other words, $C'(x) = h_0 + h_1 x + \cdots + h_{2t}x^{2t}$ then truncating results in $K(x) = h_0 + h_1 x + \cdots + h_t x^t$. This is simple and straightforward, however we don't know the polynomial and instead have its secret shares. So we have a n vector $P = ([c']_1, \ldots, [c']_n)$ and we want to compute secret share as n vector $Q = ([k]_1, \ldots, [k]_n)$. Let $H = (h_0, \ldots, h_{2t}, 0, \ldots, 0)$ be a n vector and $J = (h_0, \ldots, h_t, 0, \ldots, 0)$. Let V be the Vandermonde matrix, then we have $H \cdot V = P$ and $J \cdot V = Q$. Say L is a linear project then we have $H \cdot L = J$. Putting this together, we get $Q = J \cdot V = H \cdot L \cdot V = P \cdot V^{-1} \cdot L \cdot V = P \cdot M$ where M is a constant n x n matrix. This can viewed as yet another instance of secure computation involving only linear operations, but this requires the private inputs to be first shared though the linear operations can be computed locally. So effectively the (n, t) secret share are a linear combination of $(n, 2t)$ secret shares which requires interaction between parties to securely compute.

The GRR reduction method is attributed to Gennaro, Rabin and Rabin [20] and is more efficient than BGW reduction method since it addresses both problems in one interactive step. The main idea is the free term of the polynomial is expressed as a linear function of secret shares of $C(x) = A(x) \cdot B(x)$. Using Langrange's formula we can express $C(x) = \delta_1(x)[c]_1 + \cdots + \delta_{2t+1}(x)[c]_{2t+1}$. The final product can be expressed as $c = ab = A(0) \cdot B(0) = C(0) = \delta_1(0)[c]_1 + \cdots + \delta_{2t+1}(0)[c]_{2t+1}$ where $\delta_1(0), \ldots \delta_{2t}(0)$ are constants. This can be viewed as yet another instance of secure computation involving private inputs $[c]_i$ held by P_i and we need to compute a linear function which can be non-interactively once the secrets are shared. This is a linear function because it involves only addition and multiplication with constant operations. The first problem is addressed since the private inputs $[c]_i$ are randomly (n, t) secret shared and hence the output $[c']_i$ is random (n, t) secret shares (due to gate invariant). And the second problem is addressed since we get (n, t) secret shares. This well demonstrated in [20].

Final Remarks

MPC protocols are fascinating and almost magical in their ability to securely compute functions over private data. The GMW and BGW protocols are clean and elegant to understand. With a good understanding of the building blocks of MPC, we are confident to explore more complex topics and follow the research in this area. Further, this opens our minds to think about developing tools and libraries that can help developers to easily use MPC in their applications.

Based on our research, we'd optimistically say that MPC is starting to climb its ”slope of enlightenment” on the Gartner Hype Cycle [6]. There are current toolkits that offer high-level programming language APIs to allow users to leverage MPC to do shared computations, which then are automatically compiled to use low-level cryptographic primitives that guarantee certain standards of security and privacy.

However, there is still significant ground to cover from both the research and implementation side. Most current implementations rely on a fairly weak security model containing only passive adversaries, and many require an honest majority. For the true ”golden apple” applications of MPC hardness against a majority of corruptible, active adversaries is generally necessary. Furthermore, we believe that the current forefront of research in MPC is represented in the newer works using MPC to train machine learning models. ML progress constantly requires more data, but societal trends currently are towards greater privacy protections for data holders and generators, as evidenced in laws in Europe.

Sources

References

[1] A. Yao. Protocols for Secure Computations (Extended Abstract). 23rd Annual Symposium on Foundations of Computer Science, IEEE CSP 160-164, 1982.

[2] C. Orlandi. Is MultiParty Computation Any Good In Practice? ICASSP, 2011.

[3] Y. Lindell B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries, iacr.org, 2008.

[4] Y. Lindell. Secure Multiparty Computation (MPC), iacr.org, 2020

[5] A. Yao. How to Generate and Exchange Secrets, 27th FOCS, pages 162–167, 1986.

[6] Wikipedia, Hype Cycle, en.wikipedia.org, 2023

[7] A. Dalskov D. Escudero M. Keller Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security, iacr.org, 2020.

[8] M. Keller K. Sun Secure Quantized Training for Deep Learning, iacr.org, 2022.

[9] Y. Bao et al. HACCLE: Metaprogramming for Secure Multi-Party Computation, GPCE, 2021

[10] H. Smajovic A. Shajii et al. Sequre: a high-performance framework for secure multiparty computation enables biomedical data sharing, Genome Biology, 2023

[11] S. Wagh S. Tople et al. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning, arxiv.org, 2020

[12] M. Hastings B. Hemenway D. Noble S. Zdancewic. SoK: General Purpose Compilers for Secure Multi-Party Computation, IEEE Symposium on Security, 2019.

[13] D. Evans V. Kolesnikov M. Rosulek. A Pragmatic Introduction to Secure Multi-Party Computation

[14] A. Choudhury, NPTEL IIIT Bengaluru, Secure Computation: Part 1

[15] J. Spenger, MPC Study Group: GMW Protocol

[16] J. Spenger, MPC Study Group: BGW Protocol

[17] T. Rabin, Secure Multiparty Computation

[18] A. Choudhury, NPTEL IIIT Bengaluru, Lec 22 The BGW MPC Protocol: The Case of Non-Linear Gates

[19] A. Choudhury, NPTEL IIIT Bengaluru, Lec 23 The Degree-Reduction Problem

[20] A. Choudhury, NPTEL IIIT Bengaluru, ec 24 The Gennaro-Rabin-Rabin (GRR) Degree-Reduction Method

[21] G. Asharov Y. Lindell, A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation, 2022

[22] M. Ben-Or S. Goldwasser A. Wigderson, Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation