Robust and Verifiable MPC with Applications to Linear Machine Learning Inference[⋆]

Tzu-Shen Wang[1], Jimmy Dani[1], Juan A. Garay[1], Soamar Homsi[2], and Nitesh Saxena[1]

[1] Department of Computer Science, Texas A&M University, College Station, TX, USA
`{jasonwang017, danijy, garay, nsaxena}@tamu.edu`

[2] Information Assurance Branch, Information Warfare Division, Air Force Research Laboratory / Information Directorate, Rome, NY, USA
`soamar.homsi@us.af.mil`

Abstract. In this work, we present an efficient secure multi-party computation MPC protocol that provides strong security guarantees in settings where potentially a majority of the participants may be malicious and behave arbitrarily. Our protocol achieves both *complete identifiability* and *robustness*. With complete identifiability, honest parties can detect and unanimously agree on the identity of any malicious party. Robustness allows the protocol to continue with the computation without requiring a restart, even when malicious behavior is detected. Additionally, our approach addresses the performance limitations observed in MPC protocols which also achieve strong security properties.
Finally, we benchmark our protocol on a ML-as-a-service scenario, wherein clients off-load the desired computation to the servers, and verify the computation result. Our benchmarking focuses on linear ML inference, running on various datasets.

Keywords: Secure Multi-Party Computation · Robustness and Public Verifiability · Machine Learning.

1 Introduction

Outsourcing computation to cloud servers has become an invaluable practice in today's digital landscape. By off-loading intensive computational tasks to remote data centers, clients gain a cost-effective solution that eliminates the need for significant investment in hardware and software infrastructure. Instead, they can leverage the vast, on-demand computing power of the cloud to scale resources as needed. This flexibility reduces initial capital expenses and enhances operational efficiency.

While outsourcing to the cloud offers numerous advantages, it also introduces significant security challenges. Cloud-hosted data and applications are susceptible to risks such as data breaches, unauthorized access, and service outages.

[⋆] The full version of this appears in the Cryptology ePrint Archive [28]. This work was supported by AFRL/RI contract number FA8750-22-2-0267.

Ensuring data privacy and regulatory compliance becomes more complex when data resides on remote servers. In this work, we focus on safeguarding client data privacy of outsourced computations.

Secure multi-party computation (MPC) [3, 6, 16, 30] is a powerful tool for enhancing the security of outsourced cloud computing. MPC enables multiple parties to jointly compute a function over their inputs while ensuring data confidentiality. This approach allows clients to securely delegate computational tasks to the cloud, protecting sensitive information from potential exposure. As a cryptographic technique, MPC is essential for mitigating the security risks associated with cloud outsourcing, by utilizing multiple service providers. However, existing MPC protocols have certain drawbacks—some lack efficiency, while others fall short in providing robust security guarantees.

In this paper we are interested in guaranteeing security even in the presence of a dishonest majority of service providers. In such a setting—MPC with a dishonest majority—protocols can be categorized into two main types:

- Protocols with common security guarantees, exemplified by efficient and widely deployed solutions like SPDZ [13]. These protocols offer security with abort, meaning that if misbehavior by a party is detected, the protocol execution is aborted. This is due to their utilization of an homomorphic MAC, which merely enables them to detect when malicious behavior happens, but not the misbehaving actor. As a result, the computation may fail to complete successfully.
- Stronger security guarantees, such as *robustness*, which guarantees that malicious parties cannot prevent the honest parties from obtaining the output of the computation, as well as *(complete) identification* of the misbehaving parties. As shown in [11], the latter can be achieved at the expense of efficiency (due to the utilization of DLOG-based commitments), or by more sophisticated, lattice-based cryptographic methods [26].

1.1 Our Contributions

In this work we enhance the approach in [26] by introducing an additional entity, namely, a *semi-honest trusted third party* (STTP, which can be the client) to achieve robustness for a number of corruptions of up to $n-2$ parties.[3]

In [26], a tradeoff is made between privacy and robustness. With threshold t used to reconstruct shares, their protocol fails to provide robustness if there are more than $n-t$ malicious parties, and fails to provide privacy if there are more than t malicious parties. In contrast, our protocol achieves privacy if there is at least one honest party, and achieves robustness when there are at least 2 honest

[3] Regarding the reason for dissimilar thresholds—i.e., $n-2$ vs. $n-1$—with a trusted dealer, we can identify every malicious server and open its share to the other servers. However, if we identify $n-1$ malicious servers and open their shares, it would lead to the only remaining server being able to combine the $n-1$ servers' shares and its own share to recover the input. This is a situation we wish to avoid; therefore, we "degrade" our guarantees to security with abort.

party. Moreover, our protocol does not need to restart the circuit evaluation when malicious behaviors are detected, which we achieve by means of homomorphic encryption. Further, we propose an optimistic approach: If there is no malicious behavior, then the recovery process involving the STTP and "heavy" cryptographic primitives (such as homomorphic encryption) do not need to be executed. We showcase the performance of our protocol by benchmarking it on a modified neural network Network-A [18, 23, 26], which consists of a sequence of *dense* and *square* layers. In more detail, a *neuron* in the dense layer includes a weighted sum of all previous layers (or the input in case of the first layer), and it captures how much influence of each value from previous layers should be considered (in other word, we modify the non-linear layer in Network A to square operations, just as in [26]). On the other hand, the square layer adds non-linearity to the output of the dense layer. Our implementation only supports linear operation, therefore, for multiplications, we utilize Beaver triples [2]. Further, we benchmark our protocol on a concrete linear ML application.

Our MPC protocol operates over polynomial rings, batch processing of multiple inputs is inherently enabled, as these rings can be decomposed into multiple slots, with each slot encoding an input (cf. [15]).

1.2 Related Work

Our work ensures public verifiability, complete identifiability, and robustness in the presence of a dishonest majority. Related works along the SPDZ line of work (e.g., [13,18–20]), improve the efficiency of the online/offline computation phase, while still only achieving security with abort.

Other related work, such as [27] also utilize a bulletin board, enabling public verifiability. Third parties use the information published on the bulletin board to verify the correctness of the computation.

Regarding security with identifiable abort, there are also works that enable honest parties to detect malicious behavior and identify the corresponding parties, such as [26]. The protocol in [26], however, only provides robustness when there is an honest majority; otherwise, privacy will be violated. In contrast, by adding an STTP, our protocol provides robustness even under a dishonest majority and enables honest parties to recover shares held by the malicious party without having to restart the protocol.

In addition, the presence of an STTP allows us to achieve fairness even in the presence of a dishonest majority. Specifically, if a malicious party refuses to open its secret share, the STTP and an arbitrary honest party can pool their shares and reconstruct it. Thus, a dishonest party cannot abort with an advantage. In contrast, if in the protocol in [26] there is a dishonest majority, those parties can learn the secret share themselves and from that point on refuse to participate in the protocol.

Similarly, the protocol in [12] also relies on a trusted party to achieve robustness without sacrificing privacy. However, their protocol assumes that only 1 out of the 4 servers is malicious.

Finally, the protocol in [7] also recovers from faulty behavior without restarting. That protocol, however, consists of multiple committees, which is a setting incomparable to ours; moreover, it requires an honest majority for all committees.

2 Preliminaries

2.1 System Model

As it is customary, we model protocol participants as probabilistic polynomial-time Turing machines (ITMs) and consider the client-server model of computation with an STTP. We assume a point-to-point synchronous communication network, a public-key infrastructure (PKI), and a bulletin board (for simplicity, as it can be realized from the PKI). Table 1 summarizes the notation used in our protocol descriptions. In our setting, the STTP is assumed to be semi-honest and to not collude with any of the servers. As for the malicious servers, once they are detected, they are removed from the computation.

Table 1. Summary of notation used in the paper.

Symbols	**Definition**
$\mathcal{C}$	A set of clients $\{C_1, ..., C_m\}$
$\mathcal{S}$	A set of servers $\{S_1, ..., S_n\}$
STTP	Semi-honest trusted third party
$\mathcal{B}$	Bulletin board (broadcast channel)
$[x]$	Secret share of value x (e.g., a client's input)
$\mathcal{P}$	Prover in ZK proof (e.g., Σ protocol)
$\mathcal{V}$	Verifier in ZK proof

2.2 Building Blocks

MPC. In secure multi-party computation (MPC) [3,16,31], n parties hold input $x_1, ..., x_n$ respectively, aiming to compute a given function $f(x_1, ..., x_n)$ privately and correctly. Below we list the basic security properties for MPC.

- *Privacy:* The parties' inputs remain private.
- *Security with abort:* All honest parties agree on abort.
- *Robustness:* The protocol always outputs a correct result regardless of the adversary $\mathcal{A}$'s behavior (also called *guaranteed output delivery*) (cf. [10,25]).
- *Complete identifiability:* When a corrupted party misbehaves, honest parties always identify and agree on the identities of the misbehaving party

Commitments. We define the commitment operation as $\mathrm{Comm}(x, r)$, where committer commits to a message x where r is the randomness used in the commitment (r also acts as part of the decommitment in the opening phase). The interface for the verification operation is given by $\mathrm{Ver}(\mathrm{Comm}, x, r)$, where the verifier takes a commitment and checks if it is consistent with the committed message x and the decommitment r. The two basic properties of a commitment scheme are as follows (cf. [17]):

- **Hiding:** $\mathrm{Comm}(x, r)$ leaks no trival info of x. An adversary $\mathcal{A}$ breaks hiding iff with non-negl probability
  1. Parameters params $\leftarrow \mathrm{Gen}(1^n)$ are generated.
  2. The adversary A is given input params, and outputs a pair of messages $m_0, m_1 \in \{0,1\}^n$.
  3. A uniform $b \in \{0,1\}$ is chosen, and com $\leftarrow \mathrm{Com}(m_b, r)$ is computed.
  4. The adversary A is given com and outputs a bit b'.
  5. The output of the experiment is 1 if and only if $b' = b$.
- **Binding:** An adversary $\mathcal{A}$ cannot open $\mathrm{Comm}(x, r)$ to x', except with negligible probability. $\mathcal{A}$ breaks binding iff with non-negl probability
  1. Parameters params $\leftarrow \mathrm{Gen}(1^n)$ are generated.
  2. A is given input params and outputs $(comm, m, r, m_0, r_0)$.
  3. The output of the experiment is defined to be 1 iff $m \neq m_0$ and $\mathrm{Comm}(m, r) = comm = \mathrm{Comm}(m_0, r_0)$.

In order to provide complete identifiability in our MPC scheme, committed values need to be updated as the computation proceeds. The opening server computes its share x to $x^{'}$ and opens it to the receiving server. With the homomorphic property, the receiving server updates the commitment $\mathrm{Comm}(x)$ to $\mathrm{Comm}(x^{'})$, then uses $\mathrm{Comm}(x^{'})$ to authenticate $x^{'}$. By the binding property of the commitment, the authentication succeeds if and only if $x^{'}$ is correct. As such, we will require the commitment scheme to be linearly homomorphically updateable, satisfying the following properties:

- $\mathrm{Comm}(x_1, r_1) + \mathrm{Comm}(x_2, r_2) = \mathrm{Comm}(x_1 + x_2, r_1 + r_2)$
- $\mathrm{Comm}(x_1, r_1) + c = \mathrm{Comm}(x_1 + c, r_1)$
- $\mathrm{Comm}(x_1, r_1) * c = \mathrm{Comm}(cx_1, cr_1)$

Further, for efficiency reasons, we will be employing homomorphic lattice-based commitments [26]. Such commitments can (and will) be used to authenticate messages, in particular during the course of the computation server S_i opens a message to other servers $S \backslash S_i$; if S_i cheats, its misbehavior is identified. With the binding property, the cheating server can not be opened to a different message without getting detected.

With the homomorphic property, a server S_i can update a commitment locally to any layer of the computation circuit. Thus, when another server intends to open its share, the server S_i holds the commitment at the same circuit layer as the layer of opening. Further, due to the binding property, the decommitment verification passes if and only if the opened share is correct. Lattice-based commitments require only simple operations, such as multiplication and addition, whereas discrete log-based commitments involve costly exponentiation.

Σ protocols. This building block, proposed by Cramer *et al.* [9], can be used to provide a ZK proof that both a given encryption and a Pedersen commitment correspond to the same value, say, x, without revealing x.

Σ-protocols can be realized based on lattices [21, 22]. Please refer to those papers of Σ protocol for lattice-based signatures (with a similar approach for commitments) and it achieves non-interactivity via the Fiat-Shamir heuristic (cf. [21]). [26] shows that simulation proof can be constructed by allowing the simulator to generate a “fake” ZK proof, assuming a programmable random oracle (RO).

Homomorphic encryption. To provide the robustness property in our MPC scheme, we require encryption to be homomorphic, so that the encryption can be updated along with the computation of the circuit:

- $\mathrm{Enc}(x_1) + \mathrm{Enc}(x_2) = \mathrm{Enc}(x_1 + x_2)$
- $\mathrm{Enc}(x_1) + c = \mathrm{Enc}(x_1 + c)$
- $\mathrm{Enc}(x_1) \cdot c = \mathrm{Enc}(cx_1)$

BGV encryption. BGV encryption [4] is a fully homomorphic encryption scheme We also utilize the distributed decryption approach from [26] (see the full version for details [28].)[4]

3 Robust and Verifiable MPC

In this section, we first describe the relevant ideal functionalities and then describe how our protocol securely realizes these functionalities following the simulation paradigm (cf. [5]).

3.1 Ideal Functionalities

The ideal functionality for MPC is depicted (in the dishonest majority setting) in Fig. 1. Each party P_i provides its input in_i for circuit C, then obtains output $\mathrm{OUT} = C(in_0, in_1, ..., in_{n-1})$. Fig. 2 depicts the ideal functionality for MPC with completely identifiable abort ($\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$); when malicious behavior is detected, the functionality will abort and output the identity of the misbehaving party. Combining the $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$ approach with a “trusted dealer,” robustness can be achieved for a number of corruptions of up to $n-2$ parties. (For the reason for dissimilar thresholds (i.e., $n-2$ vs. $n-1$) please refer to Section 1.1.) The functionality $\mathcal{F}^f_{\mathsf{CIDA\text{-}RV\text{-}MPC}}$ for robust MPC with public verifiability is shown in Fig. 3.

3.2 Protocol Description

At a high level, the protocol consists of an offline phase and an online phase. In the offline phase, the client generates commitment and encryption parameters (including commitment's public parameters and encryption's public/private

[4] We remark that we do not utilize additively homomorphic encryption such as Paillier's [24] because our lattice-based commitment scheme works on polynomial rings, and therefore Paillier's is not compatible. Further, we do not utilize a *leveled* BGV implementation since the ciphertexts are generated by the client and given to the online servers. Having the online servers themselves reduce the ciphertext error does not seem straightforward, and we leave it for future work.

Functionality $\mathcal{F}^f_{\mathsf{MPC}}$

- **INIT:** On input (init, C_f, p) from all parties (where C_f is a circuit with n inputs and one output computing f, consisting of addition and multiplication gates over Z_p)
  1. Store C_f and p
  2. Wait for $\mathcal{A}$ to provide the set $\mathcal{I}$ of adversarially controlled party indices
  3. Store OUT $:= \perp$
- **INPUT:** On input $(\mathsf{input}, P_i, in_i)$, store (INPUT, P_i, in_i)
- **EVAL:** On input (eval) from all parties:
  1. If not all input values have been provided, output REJECT
  2. Evaluate the circuit C_f on inputs $(in_1, ..., in_n)$. When the evaluation is completed, store the resulting value as OUT
- **OUTPUT**: On input (output) from all parties:
  1. Send $(\mathsf{output\text{-}result}, \mathrm{OUT})$ to all parties P_i

Fig. 1. The ideal functionality for secure multi-party computation (MPC).

keys) and hands them to the STTPs. Next, the client and STTP collaboratively generate input shares, along with the corresponding commitments and homomorphic encryptions. The objective is to ensure that the STTP does not possess all the input shares and their associated encryptions, but instead holds only the commitments to the input shares. After that, shares, commitments, and encryptions are distributed to the corresponding server. We call our protocol $\Pi_{\text{RV-MPC}}$, which is split into two parts: offline and online.

In the online phase, the servers are responsible for carrying out the computation. If a malicious behavior is detected by any of the servers, the server makes an accusation to STTP. STTP uses the commitment to validate the accusation; if the accusation is valid, STTP broadcasts the encryption's secret key of the accused server. Next, all the servers use the received secret key to recover the malicious share held by the accused server. We now turn to a more detailed specification of the protocol.

Offline phase. As previously noted, the offline protocol is to compute the randomness utilized in the online phase, such as Beaver triples. In this phase, random elements rs are generated, to be used in masking inputs and distributing input shares accordingly. We note that we can also have the clients generate the public and secret key pair (pk_i, sk_i), hand sk_i to online s_i and also to STTP (to enable recovering in the online phase), for all i, and broadcast pk_i, and then have the online servers run the offline protocol from [26]. However, by letting the clients directly generate the secret shares and authentication data, the construction is simpler.

This approach simplifies security by leveraging the semi-honest assumption for STTPs and the clients, as our setting does not incur the additional complexity required in [26]. Our offline protocol simply has the clients generate the homomorphic ciphertexts, commitments and secret shares, and hand them over

Functionality $\mathcal{F}^{f}_{\mathsf{CIDA\text{-}MPC}}$

- **INIT:** Same as $\mathcal{F}^{f}_{\mathsf{MPC}}$. In addition, receive and record the identity of trusted client C. Set $L_{\mathsf{cheat}} := \emptyset$.
- **INPUT, EVAL:** Same as $\mathcal{F}^{f}_{\mathsf{MPC}}$.
- **OUTPUT**: On input (output) from all parties:
  1. Send $(\mathsf{output\text{-}result}, \mathrm{OUT})$ to all adversarially controlled parties $P_i \in \mathcal{I}$.
  2. Run ABORT, waiting for each adversarially controlled party to send either $(\mathsf{abort}, \mathrm{ACCEPT})$ or $(\mathsf{abort}, \mathrm{ABORT})$.
  3. Send $(\mathsf{output\text{-}result}, \mathrm{OUT}, L_{\mathsf{cheat}})$ to all parties, where OUT may now be $\perp$
- **ABORT** : On input (abort, x_i) from an adversarial server S_i
  1. $L_{\mathsf{cheat}} := L_{\mathsf{cheat}} \cup S_i$
  2. Set OUT $:= \perp$

Fig. 2. Ideal functionality for MPC with *completely identifiable abort.*

to the corresponding party. Due to space limitations, details are presented in the full version [28].

Online phase. The online protocol comprises a set of servers $\mathcal{S}$ carrying out a computation without leaking the input to any of them. *CESS* stands for *commitment-enhanced secret sharing*, and was introduced in [11]. Its objective is to realize, in dishonest majority settings, *completely identifiable abort*, meaning that *all* parties that misbehave are flagged as malicious. In order to make a CESS-type protocol practical, Rivinius *et al.* [26] proposed a lattice-based commitment scheme, which only takes approximately 20x time as MP-SPDZ [13] when there are 2 servers, whereas the approach in [11] using Pedersen commitment would be roughly 800x. The protocol in [26] completely identifiability, and combining [26] with a STTP, robustness can be achieved for a number of corruptions of up to $n-2$ parties. For the reason for dissimilar thresholds (i.e., $n-2$ vs. $n-1$) please refer to Section 1.1

The input secret-sharing phase allows the client to secret-share its input and broadcast the commitment and encryption of all inputs to the computation parties. The robust protocol's precondition is as follows: For each client input x, each server S_i holds ($[x]_i, r_i$, Comm(x_1),...,Comm(x_n), Enc(x_1),..., Enc(x_n)). The semi-trusted third party (STTP) holds all commitments and the secret keys. Additionally, each S_i holds the Beaver triples received from the offline phase as well as the commitments of all Beaver triple shares. We describe our protocol, which achieves robustness up to $n-2$ malicious parties in Fig. 4. Our protocol achieves a stronger security property than protocols in [11,26], which only achieve completely identifiable abort in the dishonest majority setting. Regarding the STTP, we argue that since it is semi-honest and non-colluding, then holding the secret keys without the ciphertexts does not violate privacy.

Next, we describe the optimized commitment opening protocol presented in [26]. When opening a commitment, it is intuitive to just decommit (i.e., di-

Functionality $\mathcal{F}^f_{\mathsf{CIDA\text{-}RV\text{-}MPC}}$

- **INIT:** Same as $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$, additionally receive and record the identity of trusted client $\mathcal{T}$. Set $L_{\mathsf{cheat}} = \emptyset$
- **INPUT:** Same as $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$
- **EVAL:** On input (eval) from all parties:
  1. If (eval) messages are not provided by more than $n-2$ parties, output REJECT
  2. If (eval) is not provided by party P_i, wait for ($\mathsf{recover}$) from STTP
  3. Evaluate the circuit C_f on inputs $(in_1, ..., in_n)$. When the evaluation is completed, store the resulting value as OUT
- **OUTPUT**: Same as $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$
- **ABORT** : On input (abort, x_i) from an adversarial server S_i
  1. Add S_i to L_{cheat}
  2. If $|L_{\mathsf{cheat}}| \geq n-1$, set OUT $= \bot$; else, wait for ($\mathsf{recover}$) from STTP
- **AUDIT CESS:** On input ($\mathsf{audit\text{-}CESS}$) from ($\mathsf{audited\text{-}CESS}$), output $(\mathsf{audited\text{-}CESS}, L_{\mathsf{cheat}})$

Fig. 3. Ideal functionality for robust MPC with completely identifiable abort and public verifiability, run by the computing parties in the dishonest majority setting and a semi-honest STTP.

rectly send the committed message and the randomness that generates the commitment). However, directly decommitting will require the commitment scheme to be equivocal in order to prove simulation-based security [11].

The equivocation property enabled the simulator to open to any message but leads to larger parameters and worse efficiency. To get rid of the necessity of equivocation properties of the commitment scheme, the authors in [26] introduced a new commitment opening protocol: The sender makes a new commitment, committing to the same message as the original commitment. Then the sender proves in zero-knowledge that the new and original commitment commit to the same message. The opening protocol in [26] only requires a programmable RO instead of an equivocal commitment. As a result, without the equivocation property, the parameters of the commitment can be much smaller, leading to improved efficiency. For more details, refer to [26].

We are able to show:

Theorem 1. *$\Pi_{\mathsf{RV-MPC}}$ realizes $\mathcal{F}^f_{\mathsf{CIDA\text{-}RV\text{-}MPC}}$ in the $(\mathcal{F}_{\mathsf{PKI}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid model.*

Due to space limitations, the proof of the theorem can be found in the full version [28].

Protocol $\Pi^{on}_{\text{RV-MPC}}$

- **Input secret sharing:** Client $\mathcal{C}$ intends to distribute input x. The simplest way is for $\mathcal{C}$ to generate randomness r in the offline protocol and compute secret shares. Then $\mathcal{C}$ chooses parameters for the homomorphic encryption, computes the homomorphic encryptions and lattice-based commitments, broadcasts the lattice-based commitments, sends the homomorphic encryptions to all the servers, sends each share to the corresponding server, and sends the decryption keys of the homomorphic encryption to the online STTP that monitors the online computation.
  1. $\mathcal{C}$ creates a share $x_1j = x - r + r_1j$, and $x_k = r_k$ where $rx = \sum rx_j$ and $1 \leq j \leq n, 2 \leq k \leq n$ (n is the number of servers)
  2. $\mathcal{C}$ computes and broadcasts $\text{Enc}(sk_j, x_j)$
  3. $\mathcal{C}$ computes and broadcasts $\text{Comm}(x_j)$
  4. $\mathcal{C}$ sends sk_j to the online STTP
- **Preconditions:**
  - Let x_j denote the share held by server $\mathcal{S}_j$
  - All servers and the STTP hold $\text{Comm}(x_i)$, $1 \leq i \leq n$.
  - All servers hold $\text{Enc}(x_i)$ and $\text{Enc}(r_i)$, $1 \leq i \leq n$.
- **Online computation:**
  1. All servers update $\text{Enc}(x_i)$ and $\text{Enc}(r_i)$ as the computation proceeds; denote the updated ciphertexts as $\text{Enc}(x'_i), \text{Enc}(r'_i)$.
  2. When $\mathcal{S}_k$ is identified as malicious, STTP broadcasts $\mathcal{S}_k$'s decryption key. The other servers decrypt $\text{Enc}(x'_k), \text{Enc}(r'_k)$, and obtain and x'_k, r'_k.
  3. To recover the computation from failure, a designated server (e.g. $\mathcal{S}_0$), adds x_k to its share (e.g., $x'_1 = x_1 + x_k$). Since x_k is a now a constant, all parties can locally update $\text{Comm}(x_1)$ and $\text{Enc}(x_1)$ by the homomorphic property of the commitment and encryption schemes.
  4. All servers send all x'_k and r'_k of the malicious server to STTP. Denote by (x'_k, r'_k) sent from server S_i as (x'_{ki}, r'_{ki}).
  5. STTP then checks if there exists inconsistency between all (x'_{ki}, r'_{ki}). If there is no inconsistency, accept $x'_{ki}, r'_{ki})$, and update the commitment.
  6. Else, do as follows:
     1. Set $M \leftarrow \emptyset$
     2. While $(\exists\ (x'_{kj}, r'_{kj})!= (x'_{ki}, r'_{ki})$:
        (a) Check the specific (x'_{kj}, r'_{kj}) and (x'_{ki}, r'_{ki}).[a]
        (b) Identify the malicious pair (x'_{km}, r'_{km}) with the commitment.
        (c) Ignore the malicious pair (x'_{km}, r'_{km}) and update $M := M \cup m$.
        (d) Accepts (x'_{ki}, r'_{ki}) that remain honest, and update the commitments.
     3. For those $m \in M$, go back to step 3.

[a] This happens at most $n - 2$ times, since each check eliminates at least one party.

Fig. 4. Our robust and publicly verifiable MPC protocol (online phase).

4 Applications and Experimental Results

4.1 Network-A Benchmark

In this section we benchmark a modified neural network Network-A [18,23] with our protocol (following the same benchmarking as in [26]). For the environment, we have three computation servers, where up to two can be malicious. In addition, we set up a STTP party to provide robustness. We use the same parameters for lattice cryptography primitives of [26], where the parameter of computation security is 40 bits. Furthermore, we calculate the size of homomorphic encryption we used with our robust approach by [1], we have BGV encryption with 350 bits.

We ran our experiments on machines with 32GB RAM and 16 vCPUs. Below is the benchmark of our computation online run time (in seconds), compared to the SPDZ and [26] protocols. We observe that, compared to SPDZ, the running time of our protocol is about 65x (see table 2).

SPDZ (LowGear)	[26]	Our protocol
≈ 0.0036s	≈ 0.135s	≈ 0.227s

Table 2. Comparison of efficiency of different protocols, benchmarking on network-A. Columns 2 and 3 represent amortized computation times.

Furthermore, we also show that our protocol recovers quickly when a malicious server is detected (see table 4.1). Since the malicious server will be eliminated from the computation, the recovery time can be offset by having one less server in the computation. In the experiment, we show the time to recover the share from the malicious party plus the time of the computation continues with the two remaining parties is not much different compared to the three-party computation when no malicious behavior is detected.

Recovery time	Recovery time + remaining computation with two parties	Our protocol with 3 parties
≈ 0.096s	≈ 0.211s	≈ 0.227s

Table 3. Time of recovery from malicious shares, recovery time plus remaining computation with two parties and run time if three party behaves honestly (All in amortized measurement).

4.2 ML Inference Framework

In this subsection, we first present the design of a framework for privacy-preserving machine learning (ML) inference, employing MPC protocols under malicious-dishonest majority security settings, followed by the evaluation of the lattice-based MPC protocol proposed in our study.

Framework design. Our framework enables secure inference using a pre-trained linear model, while ensuring the confidentiality of both the model and the inference input data.

The linear model is trained on publicly available data, and model parameters comprising weights (`w`) and biases (`b`), which are subsequently secret-shared among computation parties involved in MPC protocol. Similarly, inference normalized input data point(s) (x_i) are transformed into secret shares to safeguard user privacy. These secret shares are shared and distributed among computational parties, ensuring that no single computation party has access to the original data or model parameters.

Although the client holds both the data and the model, heavy computation is outsourced to MPC servers to address client resource limitations and to enable privacy-preserving computation in distributed settings. The client's local post-processing is minimal compared to the outsourced computation. The client initiates the process by sending secret shares of the data, which are to be processed, along with the secret shares of the weights and biases to the MPC servers. These servers perform linear computations, specifically computing $\texttt{w}\cdot\texttt{x}+\texttt{b}$, where w represents the weights, `x` represents the input data, and `b` represents the bias. This computation is performed on encrypted secret shares, ensuring the privacy of the data.

Once the MPC servers have completed the necessary computations, the results are securely transmitted back to the client in encrypted form. Upon receipt, the client decrypts these results to proceed with further data processing specific to the model used. This includes the application of activation functions and thresholding to finalize the inference process. For instances utilizing the Logistic Regression model, the decrypted output is first processed through a logistic function to map the computed values to probabilities, followed by a thresholding step to categorize these probabilities into discrete class labels.

Framework evaluation. In this section, we define the evaluation framework to evaluate the Lattice-based MPC protocol proposed in our study. We describe the datasets, experiment settings, and metrics for validating the correctness and efficiency of the proposed MPC protocol in performing ML inferences. Additionally, we compare the performance of our Lattice-based protocol with MASCOT [19], MASCOT* [5], SPDZ2k [8], and LowGear [20] MPC protocols, all evaluated under the Malicious Dishonest Majority security setting.

Description of the datasets In this study, we assessed the performance of the proposed MPC protocol on the Wisconsin Breast Cancer dataset [29] and a subset of the Iris flower dataset [14].

The Wisconsin Breast Cancer dataset contains 569 instances with 30 numerical features derived from digitized images of fine needle aspirates, labeled as either benign or malignant. The Iris dataset consists of 150 samples across three Iris species, each described by four morphological features. For this study, we

[5] MASCOT* refers to the MASCOT protocol configured with multiple MACs to enhance the security parameter, making it a multiple of the prime length [18].

used only the Iris-setosa and Iris-versicolor classes to create a balanced binary classification task, selecting 30 samples per class for training and 20 per class for testing. This setup ensures balanced training and an unbiased evaluation on unseen data.

Experiment settings In this study, we conducted experiments to evaluate the performance of a lattice-based MPC protocol, utilizing linear ML classifier, Logistic Regression. Our experiments were performed on Amazon Web Services (AWS) Cloud Virtual Machines (VMs) under two configurations: first, with all VMs situated within the same Cloud Service Provider (CSP) to ensure uniform computational resources and network conditions; second, with each computational party hosted on different CSPs to simulate a distributed environment with varying network conditions.

The Logistic Regression model was trained using the `ml` package available in MP-SPDZ. Following the training phase, the weights and biases of these models are extracted for evaluating lattice-based MPC protocol.

Assessing computation correctness To validate the correctness of computations performed by the lattice-based MPC protocol, we used Accuracy as the evaluation metric, which measures the proportion of correctly predicted instances out of the total evaluated.

In our experiments, we compared the accuracy achieved in centralized settings (evaluating plaintext data directly) with that obtained using the MPC protocol. This comparison confirmed the correctness of computations under MPC and highlighted any potential efficiency losses due to its distributed nature.

The accuracy achieved with the MPC protocol (88.33% for the Wisconsin Breast Cancer dataset and 100% for the Iris Flower dataset) matched the centralized settings. This demonstrates that our protocol performs computations correctly, maintaining high precision comparable to traditional centralized methods while ensuring secure computation.

4.3 Comparative analysis of MPC protocols.

We conducted a comparative analysis of MPC protocols, focusing on three key metrics: inference times, size of data exchange, and number of communication rounds. The detailed analysis for each metric is presented as follows.

Inference Time. The inference time is defined as time required to compute an output from a trained model using MPC. This performance metric is essential for assessing the efficiency of MPC protocols in privacy-preserving application.

All VMs hosted on same CSP: Our analysis of inference times for various Multi-Party Computation (MPC) protocols across the Iris and Breast Cancer datasets reveals no clear pattern in performance superiority except in the case of Lattice-based protocol. The MACOT, MASCOT* (mama), SPDZ2k, and LowGear protocols display closely competitive inference times on both datasets.

As shown in Table 4, the inference times observed for the Iris dataset are similar across MASCOT, MASCOT*, SPDZ2k, and LowGear protocols. The difference among these protocols is less than 0.0003 seconds. A similar trend is observable in the Breast Cancer Dataset.

	All the VMs hosted on same CSPs		**All the VMs hosted on different CSPs**	
Protocol	**Iris Dataset**	**Breast Cancer Dataset**	**Iris Dataset**	**Breast Cancer Dataset**
MASCOT	0.00260	0.00470	0.36145	0.41660
MASCOT*	0.00290	0.00510	0.36160	0.41675
SPDZ2k	0.00270	0.00490	0.36095	0.41599
LowGear	0.00280	0.00500	0.36157	0.41672
Lattice	0.01430	0.07710	0.01760	0.11070

Table 4. Inference times (in seconds) of Malicious-Dishonest Majority MPC protocols

In contrast, the Lattice-based protocol exhibits higher inference times on both datasets. However, it uniquely ensures operational continuity by allowing computations to proceed without restart in the presence of malicious behavior. Furthermore, it can detect and handle dishonest participants, guaranteeing protocol completion even under adversarial conditions. These robustness features, not offered by other protocols, make the Lattice-based approach particularly well-suited for security-critical applications.

All VMs on different CSPs: To simulate the scenario where each computation party is located on different Cloud Service Providers (CSPs), we utilized virtual machines (VMs) on the same cloud service but deployed them in different geographic locations. Specifically, we selected three AWS regions: N. Virginia, N. California, and Ohio.

Table 4 shows inference times for the settings when all the VMs are located in different geographic locations. For both the Iris and Breast Cancer datasets, SPDZ2k achieved the lowest inference times among the traditional protocols, with MASCOT, MASCOT*, and LowGear exhibiting only marginally higher values, indicating similar computational overheads. The lattice-based protocol only runs for 0.0176 seconds. This shows how amortizing many instances leads to better utilization of hardware resources. With batching, multiple instances are executed upon receiving an input element, and as result there will not exist a situation where a party finishes its computation early and waits for the next message, making the batched setting more robust to network delays.

Data Exchange. For the Iris Flower Dataset, each party in the Lattice-based protocol sends 0.223 MB of data per party, resulting in global data exchange of just 0.669 MB. This significantly contrasts with the other protocols, where the global data sent ranges from 0.021928 to 0.022576 MB. Similarly, for the Wisconsin Breast Cancer Dataset, each party in the Lattice-based protocol sends 2.25 MB of data, resulting in global data exchange of just 6.75 MB. For other MPC protocols, the global data exchange for MASCOT, MASCOT*, and LowGear protocols is 0.309856 MB, and 0.31048 MB for the SPDZ2k protocol.

Number of Rounds. Table 4.3 shows the comparative analysis of number of rounds of various MPC protocols from Malicious Dishonest Majority security settings considered in our study. MASCOT, MASCOT* (mama), SPDZ2k, and LowGear all follow a consistent pattern, with party 1 engaging in more rounds compared to parties 2 and 3. This asymmetry is due to party 1's additional role as the coordination server, responsible for data distribution and result aggregation. In contrast, the lattice-based protocol requires substantially more rounds (specifically, 1860 for each party). However, the rounds are only due to implementation considerations. Since we batch a lot of elements in one polynomial ring, we send/open ring elements for each multiplication using Beaver triples. This is different from the SPDZ protocol, where the wait is for multiple multiplications and then they are sent in batch.

	Rounds for Iris Dataset			**Rounds for Breast Cancer Dataset**		
MPC Protocol	**Party 1**	**Party 2**	**Party 3**	**Party 1**	**Party 2**	**Party 3**
MASCOT	17	13	13	21	15	15
MASCOT* (mama)	17	13	13	21	15	15
SPDZ2k	17	13	13	21	15	15
LowGear	17	13	13	21	15	15
Lattice	200	200	200	1860	1860	1860

Table 5. Number of Rounds of Malicious Dishonest Majority MPC Protocols

It is important to note that when we conducted the experiments under both configurations—computation parties located in the same region and those in different regions- the only metric that exhibited variation was the inference time, which increased when computation parties were located in different regions due to the added network latency. This observation underscores that while the efficiency of the protocols in terms of data exchanged and rounds required remains unaffected by geographic distribution, the actual performance time is influenced by the network conditions between the participating parties.

4.4 Discussion

The lattice-based protocol exhibits slightly lower efficiency compared to SPDZ protocols under normal conditions. However, in scenarios involving network latency, it demonstrates greater stability due to its amortized characteristics. Combined with its enhanced security properties, the lattice-based protocol holds certain advantages over SPDZ protocols.

5 Conclusions

In this paper we implement an MPC protocol that remains robust even under a dishonest majority. Additionally, we explore the execution of batched MPC instances and demonstrate the efficiency of our protocol. An interesting direction for future work is to optimize the protocol's efficiency for specific machine learning algorithms, thereby enhancing its practicality in real-world applications.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
2. Beaver, D., Goldwasser, S.: Multiparty computation with faulty majority (extended announcement). In: 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989. pp. 468–473. IEEE Computer Society (1989), https://doi.org/10.1109/SFCS.1989.63520
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC'88. pp. 1–10 (1988)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (2014), https://doi.org/10.1145/2633600
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, Las Vegas, Nevada, USA, October 14-17, 2001. pp. 136–145. IEEE Computer Society (2001), https://doi.org/10.1109/SFCS.2001.959888
6. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (abstract). In: STOC'88. pp. 11–19 (1988)
7. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: secure multiparty computation with dynamic participants. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12826, pp. 94–123. Springer (2021), https://doi.org/10.1007/978-3-030-84245-1_4
8. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: Spdz2k: Efficient mpc mod 2k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 769–798. Springer (2018), https://doi.org/10.1007/978-3-319-96881-0_26
9. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994), https://doi.org/10.1007/3-540-48658-5_19
10. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques. p. 311–326. EUROCRYPT'99, Springer-Verlag, Berlin, Heidelberg (1999)
11. Cunningham, R.K., Fuller, B., Yakoubov, S.: Catching MPC cheaters: Identification and openability. In: Shikata, J. (ed.) Information Theoretic Security - 10th International Conference, ICITS 2017, Hong Kong, China, November 29 - December 2, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10681, pp. 110–134. Springer (2017), https://doi.org/10.1007/978-3-319-72089-0_7

12. Dalskov, A.P.K., Escudero, D., Keller, M.: Fantastic four: Honest-majority four-party secure computation with malicious security. In: Bailey, M.D., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 2183–2200. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov
13. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012), https://doi.org/10.1007/978-3-642-32009-5_38
14. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of eugenics **7**(2), 179–188 (1936)
15. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer (2012), https://doi.org/10.1007/978-3-642-32009-5_49
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC. pp. 218–229. ACM (1987)
17. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007), http://www.cs.umd.edu/%7Ejkatz/imc.html
18. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1575–1590. ACM (2020), https://doi.org/10.1145/3372297.3417872
19. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 830–842. ACM (2016), https://doi.org/10.1145/2976749.2978357
20. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 158–189. Springer (2018), https://doi.org/10.1007/978-3-319-78372-7_6
21. Lyubashevsky, V.: Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5912, pp. 598–616. Springer (2009), https://doi.org/10.1007/978-3-642-10366-7_35
22. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 738–755. Springer (2012), https://doi.org/10.1007/978-3-642-29011-4_43

23. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 19–38. IEEE Computer Society (2017), https://doi.org/10.1109/SP.2017.12
24. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999), https://doi.org/10.1007/3-540-48910-X_16
25. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. p. 73–85. STOC '89, Association for Computing Machinery, New York, NY, USA (1989), https://doi.org/10.1145/73007.73014
26. Rivinius, M., Reisert, P., Rausch, D., Küsters, R.: Publicly accountable robust multi-party computation. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 2430–2449. IEEE (2022), https://doi.org/10.1109/SP46214.2022.9833608
27. Scholl, P., Simkin, M., Siniscalchi, L.: Multiparty computation with covert security and public verifiability. In: Dachman-Soled, D. (ed.) 3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA. LIPIcs, vol. 230, pp. 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPIcs.ITC.2022.8
28. Wang, T., Dani, J., Garay, J., Homsi, S., Saxena, N.: Robust and verifiable MPC with applications to linear machine learning inference. IACR Cryptol. ePrint Arch. p. 786 (2025), https://eprint.iacr.org/2025/786
29. Wolberg, William, M.O.S.N., Street, W.: Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository (1995), DOI: https://doi.org/10.24432/C5DW2B
30. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164. IEEE Computer Society (1982)
31. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167. IEEE, Toronto, Ontario, Canada (1986)