Robust and Verifiable MPC with Applications to Linear Machine Learning Inference

Tzu-Shen Wang*, Jimmy Dani*, Juan Garay*, Soamar Homsi†, Nitesh Saxena*

*Department of Computer Science, Texas A&M University, College Station, TX, USA

†Air Force Research Laboratory, Wright-Patterson AFB, OH, USA

Emails: {jasonwang017, danijy, garay, nsaxena}@tamu.edu, 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 with a dishonest majority of participants who may behave arbitrarily. Unlike the popular MPC implementation known as *SPDZ* [Crypto '12], which only ensures security with abort, 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 the protocol by Cunningham *et al.* [ICITS '17], which, while achieving complete identifiability, is hindered by the costly exponentiation operations required by the choice of commitment scheme.**

Our protocol is based on the approach by Rivinius *et al.* [S&P '22], utilizing lattice-based commitment for better efficiency. We achieves robustness with the help of a semi-honest trusted third party. We benchmark our robust protocol, showing the efficient recovery from parties' malicious behavior.

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. We benchmark on linear ML inference, running on various datasets. While our efficiency is slightly lower compared to SPDZ's, we offer stronger security properties that provide distinct advantages.

I. 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 not only reduces initial capital expenses but also enhances operational efficiency, enabling organizations to focus on core activities and innovation rather than managing complex IT systems. In an age defined by data-driven decisions and optimized resource use, cloud outsourcing is now essential for businesses and individuals alike.

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. Clients must depend on the cloud provider's security protocols, which may not always align with their unique requirements and standards. Ensuring data privacy, regulatory compliance, and control over sensitive information becomes more complex when data resides on remote servers. In this work, we focus on safeguarding client data privacy and ensuring the accuracy of outsourced computations.

Secure multi-party computation (MPC) [1]–[4] 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 preserving the privacy of those inputs, ensuring data confidentiality within a cloud environment. This approach allows clients to securely delegate computational tasks to the cloud, protecting sensitive information from potential exposure, even if the cloud service provider is untrusted. 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 [5]. These protocols offer security with abort, meaning that if misbehavior by a party is detected, the protocol execution is aborted. 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 [6], the latter can be achieved at the expense of efficiency, or by more sophisticated, lattice-based cryptography methods [7].

A. Our Contributions

In this work we enhance the approach in [7] by introducing an additional entity, namely, a *semi-honest trusted third party*

(STTP, which can be one of the clients) to achieve robustness for a number of corruptions of up to $n-2$ parties. (For 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, which is a situation we wish to avoid; therefore, we "degrade" our guarantees to security with abort)

In [7], 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 less than $n-2$ malicious parties. Moreover, our protocol does not need to restart as malicious behavior is detected, which we achieve by means of homomorphic encryption.

We showcase the performance of our protocol by benchmarking it on neural network Network-A [7]–[9], 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. On the other hand, the square layer adds non-linearity to the output of the dense layer; it provides features such as avoiding over-fitting and capturing more relationships that cannot be explained by using linear relations on the inputs. Further, we also benchmark our protocol on a linear ML application, where we show that accuracy is not lost, while achieving reasonable efficiency.

In ML-as-a-service scenarios, clients secret-share their inputs with the computation servers. Once the computation is complete, the servers return the output to the client. Our MPC protocol, is "batch-based," in the sense that it achieves *amortized* efficiency by running computations on collections (batches) of input data; as such it is well-suited for ML-as-a-service applications, where, in order to optimize hardware utilization and reduce costs, the computation servers may prefer processing client requests in batches rather than individually. Since 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. [10]).

B. 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., [5], [9], [11], [12]), improve the efficiency of the online computation phase, while still only achieving security with abort.

Other related work, such as [13] also utilize a bulletin board, enabling public verifiability. Third parties use the information published on the bulletin board with the messages opened during the computation 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 [7]. The protocol in [7], however, only provide 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 with 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 [7] 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 to [7], our approach applies to amortized settings, where multiple requests are served in tandem. In addition to providing better hardware utilization, the approach is a suitable candidate for MPC-as-a-service, as argued above.

C. Organization of the Paper

The organization of the rest of the paper is as follows. Section II describes the network and computational model and lists the building blocks that are used by our construction, including but not limited to lattice-based commitments, homomorphic encryption, and distributed decryption. Our robust and verifiable MPC protocol is presented in detail in Section III, together with its security analysis. Section IV focuses on experimental results: The benchmarking of Network A appears in Section IV-A, while the benchmarking of linear ML computations appears in Section IV-B.

II. Preliminaries

A. 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 I summarizes the notation used in our protocol descriptions.

TABLE I
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

B. Building Blocks

a) MPC.: In secure multi-party computation (MPC) [2], [3], [14], 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. [15], [16]).
- *Complete identifiability:* When a corrupted party misbehaves, honest parties always identify and agree on the identities of the misbehaving party

b) Commitments.: We define the commitment operation as $\text{Comm}(x, r)$, 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 $Ver(\text{Comm}, x, r)$, 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 (c.f. [17]):

- **Hiding:** $\text{Comm}(x, r)$ leaks no trival info of x. An adversary $\mathcal{A}$ breaks hiding iff with non-negl probability
  1) Parameters params $\leftarrow \text{Gen}(1^n)$ 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 $\leftarrow \text{Com}(m_b, r)$ 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 $\text{Comm}(x, r)$ to x', except with negligible probability. $\mathcal{A}$ breaks binding iff with non-negl probability
  1) Parameters params $\leftarrow \text{Gen}(1^n)$ 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 if and only if $m \neq m_0$ and $\text{Comm}(m, r) = comm = \text{Comm}(m_0, r_0)$.

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 $\text{Comm}(x)$ to $\text{Comm}(x^{'})$, then uses $\text{Comm}(x^{'})$ 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:

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

Further, for efficiency reasons, we will be employing lattice-based commitments [7] , which satisfy our homomorphic updates requirement. In addition, 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. Refer to Fig. **??** for details. Lattice-based commitments require only simple operations, such as multiplication and addition, whereas discrete log-based commitments involve costly exponentiation.

c) Σ protocols.: This building block, proposed by Cramer *et al.* [18], 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. We remark that such functionality can be converted into a non-interactive form using the Fiat-Shamir heuristic.

Σ-protocols can be realized based on lattices [19], [20]. 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. [19]). [7] shows that simulation proof can be constructed by allowing the simulator to generate a "fake" ZK proof, assuming a programmable random oracle (RO).

d) 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:

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

e) BGV encryption.: BGV encryption [21] is a fully homomorphic encryption scheme We also utilize distributed decryption from [7].

III. 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. [22]).

A. Ideal Functionalities

The ideal functionality for MPC is depicted in Fig. 1. Each party P_i (note that input parties may be different from server $S_i \in \mathcal{S}$) provides its input in_i for circuit C, then obtain output $\text{OUT} = C(in_0, in_1, ..., in_{n-1})$. Figure 2 depicts the ideal functionality for MPC with completely identifiableabort MPC

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 $(\mathsf{output\text{-}result}, \text{OUT})$ to all parties P_i

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

($\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$); 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 I-A. The functionality $\mathcal{F}^f_{\mathsf{CIDA\text{-}RV\text{-}MPC}}$ for robustness with public verifiability at Fig. 3.

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

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

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 $L_{cheat} = \emptyset$
- **INPUT:** Same as $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$
- **EVAL:** Same as $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$
- **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 $|L_{cheat}| \geq n-1$, set OUT = $\perp$
- **AUDIT CESS:** On input (audit-CESS) from (audited-CESS), outputs $(\mathsf{audited\text{-}CESS}, L_{cheat})$

Fig. 3. Ideal functionality for robust MPC with completely identifiable abort with public verifiability.

B. 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 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.

a) Offline phase.: As previously noted, the offline protocol is designed to allow the STTPs (In our settings, we have multiple offline phase STTPs and one online phase STTP) 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.

To ensure that no single STTP can access the secret input, the offline protocol employs homomorphic encryption and distributed decryption schemes (see Section 2). This approach simplifies security by leveraging the semi-honest assumption for STTPs, as our setting does not require the additional complexity presented in [7]. Here we also assume that the client remains honest, solely providing inputs and receiving outputs without further involvement. The structure of the offline phase is illustrated in Fig. 4.

Here's an extension of C offloading the computation work to STTPs. Suppose there are n servers (denote each server as S_i) and n STTPs (denote each STTP as $STTP_i$), the goal is to have each server S_i have share x_i along with commitments and homomorphic encryptions for all shares.

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

- Parties set up BGV parameters (similarly as in the distributed decryption protocol).
- For each party P_i:
  1) Sample $W_i \leftarrow U(R_p^{(I+3m)*t})$ and $y_i \leftarrow U(R_p^{I+3m})$
  2) Encrypt W_i and y_i to get Enc(W_i) and Enc(y_i), then broadcast Enc(W_i) and Enc(y_i)
  3) Commit to y_i with $R_c(y_i)$, gets Comm(y_i) then broadcasts Comm(y_i)
  4) Compute Enc(W) = Enc(W_i)
  5) For each $P_j \in P$:
     a) Define encrypted share of v = $W[,0]$ as Enc($[v]_j$) = $\sum_{l=0}^{t-1} j^l$Enc(W)[$\cdot, l$]
     b) m_j =Dist-Decrypt (Enc(y_i) − Enc($[v]_j$))
  6) Construct $\langle v_i \rangle = (y_i - m_i, R_c(y_i), \text{Comm}(y_1) - m_1, \ldots, \text{Comm}(y_n) - m_n)$
  7) Split $\langle v \rangle_i$ and Enc(v) in parts of size I, M, M, M to get views and ciphertexts for r, a, b, d.
  8) For the Beaver triples:
     a) Compute Enc(c) with Enc(a) $*$ Enc(b) (using the homomorphic property of encryption)
     b) Compute $\langle c \rangle_i = \langle d \rangle_i$+Dist-Decrypt(Enc($c$) − Enc($d$))
     c) Each party P_i sends its share to $STTP_i$
     d) STTPs then sends shares of r, a, b, c to the client $\mathcal{C}$.

Fig. 4. The protocol's offline phase.

$\mathcal{C}$ sends the private homomorphic encryption keys to all STTPs, then distributes share x_i to $STTP_i$. Each $STTP_i$ computes the homomorphic encryptions and lattice-based commitments of x_i, broadcasts the lattice-based commitments, sends the homomorphic encryptions to all the servers, and delivers each share to the corresponding server. Note that each $STTP_i$ only gets a single share x_i, which is random respective to the real value x.

Another extension involves having the servers compute the offline phase rather than the STTPs. In this scenario, it is necessary to prevent servers from behaving malicious without getting identified. To address this, the client can generate a secret and private key pair for each party. The client then broadcasts the public key and sends each secret key to the respective parties and the STTP. Once the secret and public keys have been distributed, the parties can execute the offline protocol as described in [7].

b) 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 [6]. 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.* [7] proposed a lattice-based commitment scheme, which only takes approximately 20x time as MP-SPDZ [5] when there are 2 servers, whereas the approach in [6] using Pedersen commitment would be roughly 800x. The protocol in [7] realizes the completely identifiable approach, and combining [7] with 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 I-A The input secret sharing phase allows the client to secret shares its input and broadcasts 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. 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. 5. Our protocol achieves a stronger security property than protocols in [6], [7], which only achieve a completely identifiable abort in a dishonest majority setting.

Additionally, we describe the optimized commitment opening protocol in [7] here.When opening a commitment, it is intuitive to just decommit (directly send the committed message and randomness that generates the commitment). However, directly decommit will require the commitment scheme to be equivocal in order to prove simulated secure [6].

The equivocation property enabled the simulator to open to any message (e.g., open Comm(x) to $x^{'}$, where $x^{'} \neq x$) but led to larger parameters and worse efficiency. To get rid of the necessity of equivocation properties of the commitment scheme, [7] introduced a new commitment open 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 [7] 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, please refer to [7].

C. Security Proof

In this section, we argue the security of protocol $\Pi_{\text{RV-MPC}}$.

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

In [5], the authors prove that Π_{SPDZ} realizes $\mathcal{F}^f_{\mathsf{MPC}}$; further, in [6], it is proven that Π_{CESS} realizes $\mathcal{F}^f_{\mathsf{CIDA\text{-}MPC}}$. This subsection aims to prove $\Pi_{\text{RV-MPC}}$ realizes $\mathcal{F}^f_{\mathsf{CIDA\text{-}RV\text{-}MPC}}$. One can observe the additional feature $\Pi_{\text{RV-MPC}}$ does compared to Π_{CESS} is to provide public verifiability and robustly open some malicious shares. For public verifiability, credited from [6], since all exchanged messages are public, the simulation is trivial. As for robustly open malicious share, the simulation could be achieved by having the simulator choose the public and private keys of all servers in $\mathcal{S}$ then generates/opens the homomorphic encryptions. Given the indistinguishable property of homomorphic encryption under different keys, one

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 $x_1 j = x - r + r_1 j$, 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]Happens at most $n-2$ times, since each check eliminates at least one party.

Fig. 5. Our robust and publicly verifiable MPC protocol (online phase).

can not distinguish the homomorphic encryptions generated by the simulator from the homomorphic encryptions in the real world.

Another difference is that our scheme is a client-to-server mode (the client sends the inputs to the server), [5]–[7] are in a pure server mode. To build a transform from pure server mode to client and server mode, we observe the following: In the pure server mode, we have two types of input shares distributing: malicious server input distributing and honest server input distributing. In our case, we are in the model that the client provides the input shares. As we assume the client is always honest, we could view our model the same as honest server input distributing.

Proof sketch: Our proof is a combination of techniques used in [5]–[7]. For the simulator to simulate, it initially provides dummy input shares that add up to zero. Then after getting back the output from the ideal function, the simulator adjusts the output shares held by the honest parties to be consistent with the output. For example, the simulator has output $y^{'}$ that is computed using dummy input shares, and y that is returned by the ideal function, the simulator picks one of the honest parties and then adds $y - y^{'}$ to the share held by the selected input party, so now the output will also be y instead of $y^{'}$ (c.f. Appendix A.3 of [5]). For the homomorphic encryption, we do not need to adjust it as discussed above. For opening the commitments, as discussed in [7], [23], with a programable random oracle, the simulator can fake a zero-knowledge proof for the final open, which is indistinguishable from the real protocol.

The simulator for $\Pi_{\text{RV-MPC}}$ is shown in Fig. 6. □

IV. Applications and Experimental Results

A. Network-A Benchmark

In this section we first benchmark Network-A [7]–[9] with our protocol. 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 [7], where the parameter of computation security is 40 bits. Furthermore, we calculate the size of homomorphic encryption we used with our robust approach by [24], 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 [7] protocols. We observe that, compared to SPDZ, the running time of our protocol is about 65x (see table II).

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

SPDZ (LowGear)	[7]	Our protocol
$\approx$ 0.0036s	$\approx$ 0.135s	$\approx$ 0.227s

Simulator $S_{\text{RV-MPC}}$

- **INIT:** $S_{\text{RV-MPC}}$ provides a CRS, such that it knows the lattice-based commitment parameters. The simulator then chooses the public/private key of the homomorphic encryption used for robustness.
  Then $S_{\Pi_{\text{RV-MPC}}}$ distributes shares, commitments, and homomorphic encryptions to each server as below:
- **INPUT:** For each input, $S_{\text{RV-MPC}}$ generates dummy shares of 0, generates the commitments and homomorphic encryptions, and then distributes to each server.
- **EVAL:** $S_{\text{RV-MPC}}$ evaluates circuit C gate by gate
- **OUTPUT:** $S_{\text{RV-MPC}}$ gets the output from functionality $\mathcal{F}_{\text{RV-MPC}}$
  then the simulator $S_{\text{RV-MPC}}$ proceeds as follows:
  1) If the $L_{cheat} >= n-2$, the simulator aborts.
  2) Else, modify the output share of one of the honest servers to be consistent with the output out. (Suppose the output from $\mathcal{F}_{\text{RV-MPC}}$ is out, and with the dummy 0 share, we get out', add out-out' to a output share of one of the honest servers)
  3) For the zero-knowledge proof for the final output, as stated in [7], the simulator can fake a ZKP with the programmable random oracle.
- **OPEN:** The simulator is able to do so since it holds the public and private keys for all homomorphic encryptions. The simulator sends the decryption key to servers when detecting a server that is malicious. Furthermore, the dishonest parties' shares are distributed uniformly, therefore, the open view between the real and ideal world will be indistinguishable.
- **AUDIT** : The simulator can do an audit since the commitments can be opened to the public (with hiding property, the commitment does not leak info about the share).

Fig. 6. Simulator for $\Pi_{\text{RV-MPC}}$'s proof.

TABLE III
THE THREE COLUMNS FROM LEFT TO RIGHT INDICATES: 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).

Recovery time	Recovery time + remaining computation with two parties	Our protocol with 3 parties
$\approx 0.096s$	$\approx 0.211s$	$\approx 0.227s$

Furthermore, we also show that our protocol recovers quickly when a malicious server is detected (see table III). 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.

B. 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.

C. 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 overall framework design is depicted in Figure 7.

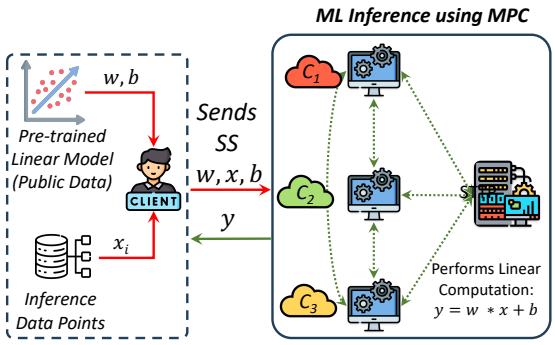

Fig. 7. A Design of ML Inference Framework

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 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.

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 w·x+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.

D. 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 [25], MASCOT* [1], SPDZ2k [26], and LowGear [27] MPC protocols, all evaluated under the Malicious Dishonest Majority security setting.

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

The Wisconsin Breast Cancer dataset, made available by Wolberg *et al.* [28], is widely used in ML research and includes features derived from digitized images of fine needle aspirates (FNA) of breast masses. These masses are categorized into two classes: benign or malignant. The dataset contains 569 instances, each with 30 attributes or features. These gestures describe various metrics for the tumors: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. This dataset is frequently utilized in ML research making it a benchmark for comparing the performance of various classification algorithms for binary classification tasks.

The Iris flower dataset, introduced by Fischer [29], is a classical dataset in ML research. It contains 150 samples, distributed across three classes of Iris species: Iris-setosa, Iris-versicolor, and Iris-virginica. Each instance is described by four features: sepal length, sepal width, petal length, and petal width. Due to its simplicity and balanced structure, the dataset is commonly used for classification tasks, particularly for exploring the performance of algorithms in multiclass classification scenarios.

For our study, we focused on a subset of the Iris dataset, considering only the Iris-setosa and Iris-versicolor classes to formulate a binary classification task. From the 100 relevant samples (50 samples per class), we used 30 samples from each class for training and reserved 20 samples from each class for testing. This setup ensured a balanced dataset for training while providing an unseen test set for evaluating the performance of the classification model. This selection of balanced number of samples for the samples from each class ensures that the classifier is trained on a balanced data set from both classes, thereby avoiding bias for a particular class and allowing robustness evaluation on unseen data.

b) 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. More specifically, we assess the ability of the MPC protocol to perform secure and efficient computations.

c) 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.

d) 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 below:

1) **Inference Time:** The inference time is defined as time required to compute an output from a trained model under an MPC setup. This performance metric is essential for assessing the efficiency of MPC protocols as it influences its scalability 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.
   For the Iris Dataset, the inference times are remarkably close, with the fastest (MASCOT at 0.0026122 seconds) and the slowest (MASCOT* at 0.0028984 seconds) among them differing by less than 0.0003 seconds. A similar trend is observable in the Breast Cancer Dataset, where the range between the fastest (MASCOT at 0.00472605 seconds) and the slowest among the traditional protocols (MASCOT* at 0.00506337 seconds) remains narrow.
   Conversely, the Lattice-based protocol records the highest inference time at 0.0771 seconds on Wisconsin Breast

[1]MASCOT* refers to the MASCOT protocol configured with multiple MACs to enhance the security parameter, making it a multiple of the prime length [9].

Cancer Dataset and 0.0143 seconds on Iris Flower Dataset. While this may seem less performance-driven, the Lattice-based protocol offers several advantages in malicious scenarios. Unlike other protocols, Lattice-based protocol is designed to continue computations without restart even if a malicious behavior is detected, thus maintaining the integrity of the computation process without the need for time-consuming restarts. This ensures operational continuity and is not available in other MPC protocols. Additionally, Lattice incorporates mechanisms to identify and handle cheaters effectively, ensuring that the computation completes successfully. While other protocols cannot identify cheaters leading potentially to indefinite computation loops. This robustness assurance that computations will complete regardless of adversarial behavior within the protocol participants provides a stronger argument for its use.

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. Each of the three computation parties was hosted in one of these regions. This configuration emulates a multi-cloud environment by introducing network variability and latency similar to what would be experienced if the parties were on different CSPs.

Our results on Iris dataset show that SPDZ2k achieved lowest inference time at 0.36095 seconds. The other protocols—MASCOT, MASCOT*, and LowGear—exhibited slightly higher inference times, ranging from 0.36145 to 0.36160 seconds. The minimal differences suggest that these protocols have comparable computational overheads in a distributed cloud environment with geographically dispersed VMs. The lattice protocol only runs for 0.0176 seconds. This is an example of how amortizing many instances leads to better utilization of hardware resources. Lots of instances will fill the network buffer/blocks, therefore, when considering network latency, it does not suffer so much. Furthermore, batching multiple instances also makes it more computation-intensive, and suffers less when considering network latency.

Similarly, for the Breast Cancer dataset, SPDZ2k demonstrated the lowest inference time at approximately 0.41599 seconds. The inference times for MASCOT, MASCOT*, and LowGear were slightly higher, between 0.41660 and 0.41675 seconds. This consistency across datasets reinforces the efficiency of SPDZ2k in handling distributed computations over disparate geographic regions. The lattice protocol only runs for 0.1107 seconds. This also shows that the lattice protocol is more robust when introducing network latency.

2) **Data Exchange:** The data exchange refers to the amount of data that needs to be communicated or exchanged between the computation parties during the execution of the protocol. This could include sharing of secret-shares, exchanging encrypted data, or transferring computed values.

   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 the SPDZ2k protocol.

3) **Number of Rounds:** The ”number of rounds” refers to the number of sequential communication steps required between the parties involved in the computation to complete a given MPC protocol.

   Our results show that MASCOT, MASCOT*(mama), SPDZ2k, LowGear, and Lattice-based protocols shows varying number of rounds distributed across three computation parties. MASCOT, MASCOT*(mama), LowGear, and spdz2k show consistent pattern where party 1 engages in more communication rounds (21 rounds for Breast Cancer Dataset, and 17 rounds for Iris Dataset) compared to parties 2 and 3 (15 rounds for Breast Cancer Dataset, and 13 rounds for Iris Dataset each). This higher number of rounds for party 1 is because the party 1 is act as coordination server as well which distributes the data to other parties for computation and also accumulates the results from other parties once the computation is completed. Lattice-based protocol requires significantly more rounds (1860 rounds for the Wisconsin Breast Cancer Dataset and 200 rounds for Iris Flower Dataset for each party), but this is offset by its unique ability to identify cheating, ensuring that computations always complete successfully. This property of Lattice-based protocol guaranteeing completion of the computation despite presence of malicious behavior, makes is ideal choice for scenarios demanding high security and reliability.

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 data exchange and number of rounds remained unchanged across all protocols. This consistency is expected, as the protocols' communication patterns and computational steps are predefined and independent of the physical locations of the computation parties. 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 inherent in cross-regional communication. 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.

E. Discussion

The variations in performance observed among the different MPC protocols in our study can be largely attributed to their underlying cryptographic mechanisms and computational models. While MASCOT, SPDZ2k and LowGear differ significantly in their preprocessing approaches, it is important to recognize that these protocols have an identical online phase. This similarity allows for a streamlined analysis of performance variations, as the primary differences originate from their distinct preprocessing states.

LowGear introduces significant enhancements by optimizing the preprocessing phase with semi-homomorphic encryption (SHE), specifically the BGV scheme, which claims to be faster than MASCOT in both LAN and WAN settings by reducing communication and computational load [27]. These improvements in the preprocessing phase enable the online phase to operate more efficiently, leveraging the precomputed data more effectively.

The choice between MASCOT, SPDZ2k, and LowGear might involve a trade-off between preprocessing efficiency and the potential benefits of a specific computational model during the online phase. While MASCOT and LowGear typically demonstrate better communication efficiency in preprocessing compared to SPDZ2k, the latter could offer advantages in certain online computations, particularly those involving comparisons or bitwise operations. This highlights how SPDZ2k's approach to modulo 2^k computations, which aligns closely with standard CPU architectures, might be particularly beneficial for operations common in many practical applications.

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. We conclude that the amortized design of the lattice-based protocol allows for better hardware resource utilization, making it more robust across varying hardware and network environments. Combined with its enhanced security properties, this suggests that the lattice-based protocol holds certain advantages over SPDZ protocols.

ACKNOWLEDGMENT

This work was supported by AFRL/RI contract number FA8750-22-2-0267.

REFERENCES

[1] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*, pp. 160–164, IEEE Computer Society, 1982.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC*, pp. 218–229, ACM, 1987.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC'88*, pp. 1–10, 1988.

[4] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (abstract)," in *STOC'88*, pp. 11–19, 1988.

[5] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings* (R. Safavi-Naini and R. Canetti, eds.), vol. 7417 of *Lecture Notes in Computer Science*, pp. 643–662, Springer, 2012.

[6] R. K. Cunningham, B. Fuller, and S. Yakoubov, "Catching MPC cheaters: Identification and openability," in *Information Theoretic Security - 10th International Conference, ICITS 2017, Hong Kong, China, November 29 - December 2, 2017, Proceedings* (J. Shikata, ed.), vol. 10681 of *Lecture Notes in Computer Science*, pp. 110–134, Springer, 2017.

[7] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "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.

[8] P. Mohassel and Y. Zhang, "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.

[9] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020* (J. Ligatti, X. Ou, J. Katz, and G. Vigna, eds.), pp. 1575–1590, ACM, 2020.

[10] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," *IACR Cryptol. ePrint Arch.*, p. 99, 2012.

[11] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," *IACR Cryptol. ePrint Arch.*, p. 1230, 2017.

[12] M. Keller, E. Orsini, and P. Scholl, "MASCOT: faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016* (E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, eds.), pp. 830–842, ACM, 2016.

[13] P. Scholl, M. Simkin, and L. Siniscalchi, "Multiparty computation with covert security and public verifiability," *IACR Cryptol. ePrint Arch.*, p. 366, 2021.

[14] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *FOCS*, (Toronto, Ontario, Canada), pp. 162–167, IEEE, 1986.

[15] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, (New York, NY, USA), p. 73–85, Association for Computing Machinery, 1989.

[16] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin, "Efficient multiparty computations secure against an adaptive adversary," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, (Berlin, Heidelberg), p. 311–326, Springer-Verlag, 1999.

[17] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[18] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings* (Y. Desmedt, ed.), vol. 839 of *Lecture Notes in Computer Science*, pp. 174–187, Springer, 1994.

[19] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *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* (M. Matsui, ed.), vol. 5912 of *Lecture Notes in Computer Science*, pp. 598–616, Springer, 2009.

[20] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings* (D. Pointcheval and T. Johansson, eds.), vol. 7237 of *Lecture Notes in Computer Science*, pp. 738–755, Springer, 2012.

[21] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[22] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Cryptology ePrint Archive,*

Report 2000/067, December 2005. Latest version at http://eprint.iacr.org/2000/067/.
[23] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, pp. 43:1–43:35, 2013.
[24] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015.
[25] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), p. 830–842, Association for Computing Machinery, 2016.
[26] R. Cramer, I. Damgård, D. E. Escudero, P. Scholl, and C. Xing, "Spd2k: Efficient mpc mod 2k for dishonest majority," in *IACR Cryptology ePrint Archive*, 2018.
[27] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making spdz great again," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 1230, 2018.
[28] M. O. S. N. Wolberg, William and W. Street, "Breast Cancer Wisconsin (Diagnostic)." UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5DW2B.
[29] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.