Attaining GOD Beyond Honest Majority With Friends and Foes*

Aditya Hegde[1]**, Nishat Koti[2], Varsha Bhat Kukkala[2], Shravani Patil[2], Arpita Patra[2], and Protik Paul[2]

[1] Johns Hopkins University ahegde@cs.jhu.edu
[2] Indian Institute of Science, Bangalore
{kotis,varshak,shravanip,arpita,protikpaul}@iisc.ac.in

Abstract. In the classical notion of multiparty computation (MPC), an honest party learning private inputs of others, either as a part of protocol specification or due to a malicious party's unspecified messages, is not considered a potential breach. Several works in the literature exploit this seemingly minor loophole to achieve the strongest security of guaranteed output delivery via a trusted third party, which nullifies the purpose of MPC. Alon et al. (CRYPTO 2020) presented the notion of *Friends and Foes* (FaF) security, which accounts for such undesired leakage towards honest parties by modelling them as semi-honest (friends) who do not collude with malicious parties (foes). With real-world applications in mind, it's more realistic to assume parties are semi-honest rather than completely honest, hence it is imperative to design efficient protocols conforming to the FaF security model.

Our contributions are not only motivated by the practical viewpoint, but also consider the theoretical aspects of FaF security. We prove the necessity of semi-honest oblivious transfer for FaF-secure protocols with optimal resiliency. On the practical side, we present QuadSquad, a ring-based 4PC protocol, which achieves fairness and GOD in the FaF model, with an optimal corruption of 1 malicious and 1 semi-honest party. QuadSquad is, to the best of our knowledge, the first practically efficient FaF secure protocol with optimal resiliency. Its performance is comparable to the state-of-the-art dishonest majority protocols while improving the security guarantee from abort to fairness and GOD. Further, QuadSquad elevates the security by tackling a stronger adversarial model over the state-of-the-art honest-majority protocols, while offering a comparable performance for the input-dependent computation. We corroborate these claims by benchmarking the performance of QuadSquad. We consider the application of liquidity matching that deals with sensitive financial transaction data, where FaF security is apt. We design a range of FaF secure building blocks to securely realize liquidity matching as well as other popular applications such as privacy-preserving machine learning. Inclusion of these blocks makes QuadSquad a comprehensive framework.

Keywords: Friends and Foes · Multiparty Computation · Oblivious Transfer

* Full version available at https://eprint.iacr.org/2022/1207.pdf
** Work done while at International Institute of Information Technology Bangalore.

1 Introduction

The classical notion of multiparty computation (MPC) enables n mutually distrusting parties to compute a function over their private inputs, such that an adversary controlling up to t parties does not learn anything other than the output. Depending on its behaviour, the adversary can be categorized as *semi-honest* or *malicious*. A maliciously-secure MPC protocol may offer security guarantee with *abort, fairness* or *guaranteed output delivery (GOD)*. While security with abort may allow the adversary alone to receive the output (leaving out the honest parties), fairness makes sure either all or none receive the output. The strongest guarantee of GOD ensures that all honest participants receive the output irrespective of the adversarial behaviour. It is well known that honest majority is necessary to achieve GOD, whereas a dishonest majority setting can at best offer security with abort for general functionalities [28]. GOD is undoubtedly one of the most attractive features of an MPC protocol. Preventing repeated failures, it upholds the trust and interest of participants in the deployed protocol and saves a participant's valuable time and resources. Moreover, it also captures unforeseeable scenarios such as machine crashes and network delay.

It is well-known that the honest majority setting lends itself well for constructing efficient protocols for a large number of parties [33,2,1,18,45] and has been shown to be practical [64,6]. In this setting, MPC for a small number of parties [5,40,4,58,27,63,61,49,63,22] has gained popularity over the last few years due to applications such as financial data analysis [17] and privacy-preserving statistical studies [15] which typically involve 3 parties. This is corroborated by the popularity of MPC framework such as Sharemind [16] which works over 3 parties. In the literature, of all MPC protocols for a small population, several achieve the highest security guarantee of GOD [49,21,44,20,23,55,56]. In most of these protocols, when any malicious behaviour is detected, parties identify an *honest* party, referred to as Trusted Third Party (TTP) and make their inputs available to it in clear. Thereafter, TTP computes the desired function on parties' private inputs and returns the respective outputs. Such learning of inputs by an honest party is allowed in the traditional definition of security, although it nullifies the main purpose of MPC. In many real-world applications that deal with highly sensitive data, such as those in financial and healthcare sectors, information leak, even to an honest party, is unacceptable. Further, in the secure outsourced computation setting, where servers (typically run by reputed companies such as Amazon, Google, etc.) are hired to carry out the computation, it may be unacceptable to reveal private inputs to the server identified as a TTP.

Another issue that persists in traditionally secure MPC protocols is the following. The malicious adversary can potentially breach privacy of protocols by sending its view to some of the honest parties. However, traditional definitions do not acknowledge this view-leakage as an attack as honest parties are assumed to discard any non-protocol messages. In this way, traditional definition fails to account for the possibly curious nature of honest parties, which is a given in real-world scenarios. Consequently, many well-known protocols relying on threshold secret sharing (such as BGW [14]), satisfying traditional security against t ma-

licious corruptions, immediately fall prey to this view-leakage attack. Indeed, an honest party on receiving the view of any t corrupt parties can learn the inputs of all the parties. Note that the traditional MPC protocols are vulnerable to this view-leakage attack which are not just restricted to GOD protocols but also protocols with weaker security notion of fairness. We emphasize that such a view-leakage attack is not irrational on the part of adversary's behaviour as it can be motivated by monetary incentives.

We showcase how reliance on a TTP and the view-leakage attack inherent in traditionally secure MPC is detrimental to data privacy in real-world applications via the example of liquidity matching. Consider a set of banks that have outstanding transactions that need to be settled among themselves. Liquidity matching enables settlement of inter-bank transactions while ensuring that each bank has sufficient liquidity. Here, liquidity means the balance of a bank, and matching requires that each bank, upon processing of the outstanding transactions, has non-negative balance. Since transactions comprise sensitive financial data, it is required to perform liquidity matching in a privacy-preserving manner. Hence, when designing MPC protocols for the same. It is imperative for the protocol to provide GOD, owing to the real-time nature of such transactions. That is, aborting the execution is not an acceptable option as it may lead to an indefinite delay in processing the transactions. The work of [7] has explored this application in the traditionally secure MPC setting. However, given the sensitive nature of the application, reliance on a TTP to attain GOD, and the view-leakage attack, render the traditionally secure MPC solution futile.

Inspired by the above compelling concerns of reliance on a TTP and view-leakage, [3] proposed a new MPC security definition, Friends & Foes (FaF). In this definition, an honest party's input is required to be safeguarded from quorums of other honest parties, in addition to the standard security against an adversary. This dual need is modelled through a decentralized adversary. Specifically, there is one malicious adversary that corrupts at most t out of n parties *(Foes)* and another semi-honest adversary, controlling at most h^* parties *(Friends)* out of the remaining $n-t$ parties. A protocol secure against such adversaries is said to be (t, h^*)-FaF secure. Technically, in the FaF model, not only should the views of t malicious parties, but also the views of every (disjoint) subset of h^* semi-honest parties, be simulatable separately. Moreover, FaF requires security to hold even when the malicious adversary sends its view to some of the other parties (semi-honest). Thus, FaF-security is a better fit for applications that deal with highly sensitive data, as in the case of liquidity matching.

Alon et al. showed in [3] that any functionality can be computed with fairness and GOD in the (t, h^*)-FaF model, iff $2t + h^* < n$ holds. Since protocols with a small number of parties are pragmatic, from the above condition it is evident that a minimum of 4 parties is necessary to achieve the desired level of FaF-security. This implies that $t = 1, h^* = 1$. While the sufficiency of $t = 1$ is well established in the literature [59,67,55,56,31,44,20,21,23], we trust that $h^* = 1$ also suffices for most practical purposes, assuming honest parties do not collude. Thus, we design protocols in the 4PC setting providing $(1, 1)$-FaF security. It is worth

noting that relying on a 4PC protocol with 2 malicious corruptions to achieve this goal is insufficient, since GOD is known to be impossible in this setting. On the other hand, although the 4 party honest-majority setting tackling a single corruption can offer GOD security, it is susceptible to the view-leakage attack.

Keeping practicality in mind, for the optimal 4PC setting considered, we describe the design choices made to attain an efficient protocol. To obtain a fast-response time as required for real-time applications, we operate in the preprocessing paradigm which has been extensively explored [35,9,34,54,55,56]. Here, the protocols are partitioned into two phases, a function dependent (input independent) *preprocessing phase* and an input dependent *online phase*. Following recent works [16,34,36,32] we build our protocols over 32 or 64 bit rings to leverage CPU optimizations. Further, to aid resource constrained clients in performing computationally intensive tasks, the paradigm of secure outsourced computation (SOC) has gained popularity. In this setting, clients can avail computationally powerful servers on a 'pay-per-use' basis from Cloud service providers. In this work, we provide secure protocols for performing computations in the 4-server SOC setting. The servers here are mapped to the parties of our 4PC.

When designing FaF-secure protocols in a given setting, it is both theoretically profound and practically important to know, whether information-theoretic security is possible to be achieved. If not, it is important to identify what cryptographic assumption is required. [3] shows impossibility of information-theoretic FaF-secure MPC with less than $2t + 2h^*$ parties and presents a protocol relying on semi-honest oblivious transfer (OT) with at least $2t + h^* + 1$ parties. However, the necessity of OT in the latter setting was not known. We settle this question, showing the necessity of semi-honest OT. This proves the tightness of the protocol of [3] in terms of assumption, and implies that any 4PC in $(1, 1)$-FaF setting requires semi-honest OT. This requirement puts FaF security closer to the dishonest majority setting where the same necessity holds [43,50], than the honest majority setting which is known to offer even the strongest security of GOD information-theoretically.

1.1 Related Work

We restrict the discussion to practically-efficient secret-sharing based (high throughput regime) MPC protocols over small population for arithmetic and Boolean world, since this is the regime of focus in this work.

In the honest-majority setting, we restrict to protocols achieving fairness and GOD over rings. The GOD protocol offering the best *overall* communication cost is that of [20]. [24,67,55], present 3PC protocols in the preprocessing paradigm, and thus have faster online phase than [20]. Of these, [55] elevates the security of the former two, from fairness to GOD. In the 4PC regime, [56] presents the best GOD protocol improving over the previously best-known fair protocol of [25] and GOD protocol of [21,55].

The work that comes closest to ours in terms of security achieved is that of Fantastic Four [31] which is devoid of function dependent preprocessing. It attempts to offer a variant of GOD, referred to as *private robustness* without the

honest party learning other parties' inputs. However, this work does not capture the behaviour of a malicious adversary which allows it to send its complete view to an honest party, thus falling short of satisfying the FaF security notion.

In the dishonest-majority setting, the study of practically-efficient protocols started with the work of [35] which was followed by [53,54]. This line of work culminated with [13] which has the fastest online phase. However, these protocols work over fields. The works that extend over rings are [30,65] and of these the latter is a better performer. In this regime, all the protocols work in preprocessing paradigm, where the common trend had been to generate Beaver multiplication triples [11] in the preprocessing and consume them in the online phase for multiplication. The majority of the works focus on bettering the preprocessing and choose either Oblivious Transfer (OT) [53] or Somewhat Homomorphic Encryption (SHE) [35,30,65] to enable triple generation.

1.2 Our Contribution

QuadSquad: A $(1,1)$**-FaF Secure 4PC.** We propose the first, efficient, $(1,1)$-FaF secure, 4PC protocol in the preprocessing paradigm, over rings (both $\mathbb{Z}_{2^\lambda}$ and $\mathbb{Z}_2$), that achieves fairness and GOD. Casting our protocol in the preprocessing paradigm allows us to obtain a fast online phase, with a cost comparable to the best-known dishonest as well as honest majority protocols. Furthermore, we achieve GOD, without incurring any additional overhead in the online phase, in comparison to our fair protocol. This is depicted in Table 1.

Here, with respect to honest-majority protocols, we compare QuadSquad's multiplication with the best-known 4PC of Tetrad [56] which relies on a TTP, and the protocol of Fantastic Four [31] which offers *private robustness* without relying on a TTP. With respect to dishonest-majority protocols, we compare with the best-known OT-based protocol of MASCOT [53] since our protocol also relies on OTs in the preprocessing. While QuadSquad, [31] and [56] work over ring, [53] exploits field ($\mathbb{F}$) structure. Further, the protocol in [31] does not have a separate preprocessing phase. We indicate this in Table 1 by "NA" (Not Applicable). As per the table, QuadSquad is comparable to both the honest-majority and dishonest-majority protocols in the online phase and outperforms [53] in the preprocessing. Our offer over [56], [31] is stronger security against an additional semi-honest corruption, with a comparable online cost. Our offer over [53] is the stronger guarantee of fairness/GOD with comparable online cost (and better preprocessing cost).

Ref.	Preproc.	Online		Model	Security
	Comm.	Rounds	Comm.		
Tetrad ($\mathbb{Z}_{2^\lambda}$)	2	1	3	HM	GOD
Fantastic Four ($\mathbb{Z}_{2^\lambda}$)	NA	1	6	HM	GOD
MASCOT ($\mathbb{F}$)	7713	2	12	DM	abort
QuadSquad ($\mathbb{Z}_{2^\lambda}$)	1558	3	7	FaF	Fair
QuadSquad ($\mathbb{Z}_{2^\lambda}$)	3110	3	7	FaF	GOD

– The comm. complexity is given in terms of elements from $\mathbb{Z}_{2^\lambda}/\mathbb{F}$ (of size 2^{64}), as applicable. HM: Honest majority; DM: Dishonest majority.

Table 1: Comparison of mult of MASCOT, Fantastic Four and Tetrad with QuadSquad

Necessity of OT. FaF is closer to dishonest majority (with 2 corruptions out of 4), and hence, public-key primitives are inevitable. We back this up by proving the necessity of OT. We prove the necessity of semi-honest OT for (t, h^*)-FaF (abort) secure protocol with $n \leq 2t + 2h^*$ (by constructing the former from the latter). The goal of this result is to justify that a protocol, including ours, in FaF-model will require public-key primitives. Given this, we use semi-honest OT, but restrict its use to preprocessing alone[3].

Building blocks and applications. We consider the application of liquidity matching where FaF security is more apt. We design a range of FaF secure building blocks to securely realize liquidity matching, as well as other popular applications such as privacy-preserving machine learning (PPML). The description of the building blocks appears in Table 2. Although these can be naively obtained by extending techniques from the literature, the resultant building blocks have a heavy communication overhead. We therefore go one step ahead and design customised building blocks which are efficient and help in improving the response time of these applications.

Protocol	Input	Output	Description
$[\![\cdot]\!]$-$\mathsf{Sh}^{\mathsf{SOC}}$	v	$[\![\mathsf{v}]\!]$	User $[\![\cdot]\!]$-shares input v with the servers
$[\![\cdot]\!]$-$\mathsf{Rec}^{\mathsf{SOC}}$	$[\![\mathsf{v}]\!]$	v	Servers reconstruct v to U
BitExt	$[\![\mathsf{v}]\!]$	$[\![\mathsf{msb}(\mathsf{v})]\!]^{\mathbf{B}}$	Extracts most significant bit of an arithmetic shared value v
Bit2A	$[\![\mathsf{b}]\!]^{\mathbf{B}}$	$[\![\mathsf{b}]\!]$	Converts boolean sharing of a bit b to arithmetic sharing
BitInj	$[\![\mathsf{b}]\!]^{\mathbf{B}}$, $[\![\mathsf{v}]\!]$	$[\![\mathsf{b} \cdot \mathsf{v}]\!]$	Outputs $[\![\cdot]\!]$-shares of $\mathsf{b} \cdot \mathsf{v}$, where bit b is $[\![\cdot]\!]^{\mathbf{B}}$-shared and v is $[\![\cdot]\!]$-shared
DotPTr	$\{[\![\mathsf{x}^s]\!], [\![\mathsf{y}^s]\!]\}_{s\in[n]}$	$[\![\sum_{s\in[n]} \mathsf{x}^s \cdot \mathsf{y}^s]\!]$	Outputs $[\![\cdot]\!]$-shares of dot product of $[\![\cdot]\!]$-shared vectors $\{\mathsf{x}^s\}_{s\in\mathsf{n}}, \{\mathsf{y}^s\}_{s\in\mathsf{n}}$

Table 2: Build blocks for various applications

Benchmarks. We showcase the practicality of QuadSquad by benchmarking its MPC, as well as the performance of secure liquidity matching and PPML inference for two Neural Networks (NN). We implement and benchmark our 4PC protocol over a WAN network using the ring $\mathbb{Z}_{2^{64}}$, and report the latency, throughput and communication costs in the preprocessing and online phase. We observe that the throughput of our GOD protocol is comparable to that of the fair protocol, and has an overhead of up to 4.5× in the online phase over [56] and [31]. This overhead indicates the cost to achieve the stronger notion of FaF-security. On the other hand, QuadSquad outperforms [53] by a factor of up to 4.5× in the online phase. With respect to the applications, we observe a runtime of 6 and 10 seconds for the two NNs, and a runtime of 15 seconds for liquidity matching. The reported runtime for both applications is practical.

[3] As mentioned in §1.1, SHE offers an alternative to OT. However, relying on the heels of recent interesting work on OT [72] and the huge effort on improving OT in the last decade [19,52], we opt for OT based approach. Translating our approach in the SHE regime is left for future exploration.

1.3 Technical Highlights

In this section, we elaborate on the design choices of our protocol, the challenges involved and the approach taken to tackle them. One approach to achieving $(1,1)$-FaF security in the 4PC setting is via a 4-party identifiable abort (IA) protocol, where upon detecting misbehaviour, the protocol can be re-run with a default input for the identified corrupt party. However, we deviate from this approach and choose dispute pair identification for achieving the desired security due to the following reasons. First, note that there is no customised 4PC IA protocol in the literature. Moreover, since the threshold of corruption in $(1,1)$-FaF considering malicious as well as semi-honest parties corresponds to a dishonest majority, we have to consider IA protocols in the same setting to prevent susceptibility to view-leakage attack. This would inherently require us to consider generic n-party dishonest majority IA protocols, instantiated for the specific case of $n = 4$ and $t = 2$, which do not offer a practically efficient solution. Specifically, the state-of-the-art protocol in this setting [10] requires online communication of 24 elements per multiplication-gate, which is significantly higher than the online communication cost of our protocol. Designing a customised 4PC IA protocol is an orthogonal question which is left as an open problem.

Necessity of OT. To prove the necessity of semi-honest OT for a generic n-party (t, h^*)-FaF secure (abort) protocol with $t + h^* < n \leq 2t + 2h^*$, we construct the former from the latter. Recall that the necessity of $n > t + h^*$ for abort security and $n > 2t+h^*$ for GOD in the FaF model is known from [3]. Note that our proof holds up to $n \leq 2t + 2h^*$, which subsumes the optimal bound on n for the GOD setting. We show that an n-party (t, h^*)-FaF secure protocol π_f for computing the function $f((m_0, m_1), \perp, \ldots, \perp, b) = (\perp, \perp, \ldots, \perp, m_b)$, where $n \leq 2t + 2h^*$, can be used to construct a semi-honest OT. We give the formal proof in §3.

QuadSquad: Robust $(1,1)$-FaF Secure 4PC. The core idea of our 4PC protocol lies in designing the sharing, reconstruction and multiplication primitives.

Sharing: To facilitate operating over rings and ensure privacy in FaF model with 1 malicious and 1 semi-honest party, we rely on Replicated Secret Sharing (RSS) with a threshold of 2. This requires 6 components where each pair of parties holds a common component. This is higher than the 4 components in RSS with threshold 1 and 3 which are typically used in honest and dishonest majority settings respectively. In QuadSquad, each party has only 3 components of a sharing which poses the challenge in ensuring an efficient reconstruction.

Reconstruction: Although a naive reconstruction towards all would require a communication of 12 elements, our protocol reduces this to an *amortized* cost of 7 elements. Both our sharing and reconstruction protocols extensively rely on primitives which leverage the honest behaviour of at least 3 parties to ensure dispute pair (DP) identification.

Multiplication: The higher number of components in our sharing semantics makes our multiplication protocol non-trivial. At a high level, the multiplication of

2 shared values results in 36 summands, which we broadly categorize into 3 types based on the number of parties which can locally compute each summand. We give separate treatment to each category, of which the summands that can be computed by a single party and those which cannot be computed by any party are of particular interest. The main challenge in the former is ensuring the correctness of a party's computation, for which we build upon the distributed Zero-Knowledge (ZK) protocol of [20]. The latter requires a new *distributed multiplication* protocol where two distinct pairs of parties hold the inputs to the multiplication and the goal is to additively share the product between the pairs. This primitive relies on OT. Here, the main challenge is ensuring the correctness of inputs to OT, for which we leverage the (semi) honest behaviour of at least 3 parties and the fact that every pair of parties holds a common component. Apart from several optimization techniques, the primary technical highlight in this part includes the new batch reconstruction and the distributed multiplication, both of which contribute to a highly efficient multiplication protocol.

Online: For efficiency, we follow the masked evaluation paradigm by tweaking RSS as follows. We share a value using a mask which is RSS shared and a masked value which is public. Circuit evaluation is then performed on the public masked values which are required to be reconstructed in the online phase [44,13,66].

Fair to GOD: In the optimistic run (where all parties behave honestly) of our 4PC protocol the function output is computed correctly. However, in case any malicious behaviour is detected during protocol execution, a dispute pair (DP) is identified which is assured to include the malicious party. The protocol that we obtain by terminating at the earliest point of dispute discovery, offers fairness. Note that the fair protocols existing in the literature [67,55,56] are susceptible to the view-leakage attack and thus are not FaF secure. Further, to extend the security guarantee to GOD without incurring additional communication overhead in the online phase, we follow the commonly used approach of segmented evaluation of a circuit. Specifically, we segment the circuit and execute the above protocol in a segment-by-segment manner. In case malicious behaviour is detected in any segment, as in our fair protocol, we identify a DP. Following this, for computation of the remaining segments, we resort to a single instance of a semi-honest 2PC which is executed by parties outside DP, which we refer to as the trusted pair (TP). We use the semi-honest 2PC in a black-box manner, and this can be instantiated with the state-of-the-art protocol. We use ABY2.0 [66], for this purpose, which is also designed in the preprocessing paradigm. To extend support for the online phase of [66], each pair of parties executes an instance of the preprocessing of [66], along with the preprocessing of QuadSquad. This ensures that in case DP is identified during the online phase, parties have the necessary preprocessed data for the 2PC.

Key differences from Tetrad, Fantastic Four and MASCOT. The best known honest-majority 4PC given in Tetrad differs from our construction in many aspects starting with reliance on RSS with threshold 1. This ensures every party misses a single (as opposed to 3 for us) component, offering a very efficient

reconstruction. They further utilize high redundancy (every component is held by 3 parties) and heavily rely on isolating one of the parties from most of the computation. This, together with the threshold of 1 guarantees that, in case malicious behaviour is detected during the computation, the isolated party is honest. This honest party is then elevated to a TTP. The protocol of [55] follows a similar approach for efficiency. In FaF-model, we fall short of the first and the latter paradigm fails due to the presence of an additional semi-honest party. Thus, our multiplication protocol involves all four parties and enforces different mechanisms to detect and handle malicious behaviour compared to the Tetrad protocol. Similar to [56,55], the efficiency of Fantastic Four can be attributed to the benefits of redundancy offered by RSS with threshold 1. Their work achieves a variant of GOD referred to as *private robustness* by first identifying a dispute pair in the execution involving all 4 parties, followed by reducing the computation to a 3-party malicious protocol. For this, their work eliminates one party from the dispute pair arbitrarily. Any malicious behaviour hereafter, asserts that the party from the dispute pair included in the 3PC is malicious. To achieve robustness, they execute a semi-honest 2-party protocol using the parties guaranteed to be honest. Although their approach circumvents revealing private inputs to a TTP for achieving robustness, it falls short of offering FaF-security. In particular, it is susceptible to the view-leakage attack in all the instances of its sub-protocols involving $2, 3$ and 4 parties. Moreover, in [31], the switch from 4PC to 3PC upon identifying malicious behaviour is non-interactive. This can be attributed to the threshold of 1 which ensures that any three parties together possess all the components of the sharing. However, in our case, if any malicious behaviour is detected we fall back on a semi-honest 2PC. The sharing semantics of our protocol (required to prevent view-leakage attack) are such that a pair of parties does not hold all the shares. Hence we need additional interaction for converting from 4PC sharing to a 2PC sharing.

On the other hand, MASCOT [53] relies on RSS with threshold 3 (same as additive sharing). Though every party misses 3 shares like our case, riding on the advantage of shooting for a weaker guarantee of abort, they are able to leverage king-based approach [33] for reconstruction (only one party/king is enabled to reconstruct, which later sends the value to the rest) which only ensures detection, but falls short of recovery, from a malicious behaviour. [53] delegates checks to detect malicious behaviour to the end of the protocol whereas we need to verify correct behaviour at each step to ensure fairness/GOD.

Our work leaves open several interesting questions. We elaborate on these and the challenges involved therein in the full version of the paper.

2 Preliminaries

Setting and Security. We consider a set of four parties $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ which are connected by pair-wise private and authenticated channels in a synchronous network. The function to be computed is expressed as a circuit whose topology is public and is evaluated over a ring $\mathbb{Z}_{2^\lambda}$ of size 2^λ. Our protocols are

designed in the FaF model with a static malicious adversary and a (different) semi-honest adversary each corrupting at most one (distinct) party. We make use of broadcast channel for simplicity of presentation, which can be instantiated using any protocol such as [38]. Our constructions achieve the strongest security guarantee of GOD, wherein parties receive the protocol output irrespective of the malicious adversary's strategy. We prove the security of our protocols in the ideal world/real world simulation paradigm. The security definitions and proofs appear in the full version of the paper.

In the SOC setting, the four servers execute our protocol. For client-server based computation, a client secret-shares its data with the servers. Servers perform the required operations on secret-shared data and obtain the secret-shared output. Finally, to provide the client's output, servers reconstruct the output towards it. The underlying assumption here is that the corrupt server can collude with a corrupt client. We consider computation over $\mathbb{Z}_{2^\lambda}$ and $\mathbb{Z}_{2^1}$. To deal with decimal values, we use Fixed-Point Arithmetic (FPA) [62,60,24,67] in which a value is represented as a λ-bit integer in signed 2's complement representation. The most significant bit (msb) denotes the sign bit, and d least significant bits are reserved for the fractional part. The λ-bit integer is then viewed as an element of $\mathbb{Z}_{2^\lambda}$, and operations are performed modulo 2^λ. We set $\lambda = 64, d = 13$, leaving $\lambda - d - 1$ bits for the integer part. Our protocols are cast in the preprocessing paradigm, wherein a protocol is divided into (a) function dependent (input independent) *preprocessing phase* and (b) input dependent *online phase*.

Notation 1 *Wherever necessary, we denote $\mathcal{P}$ by the unordered set $\{P_i, P_j, P_k, P_m\}$ and $\{P_i, P_{i+1}, P_{i+2}, P_{i+3}\}$. Note that $i, j, k, m \in [4]$ do not correspond to any fixed ordering, only constraint being $i \neq j \neq k \neq m$. Similarly for $i, i+1, i+2, i+3$, corresponding to a P_i, say P_2, $P_{i+1} = P_3$, $P_{i+2} = P_4$, $P_{i+3} = P_1$.*

Standard Building Blocks. Parties make use of a one-time key setup captured by functionality $\mathcal{F}_{\mathsf{setup}}$ (Fig. **??**), to establish pre-shared random keys for pseudo-random functions (PRF) among them. This functionality incurs a one-time cost, and thus can be instantiated using any FaF-secure protocol such as that of [3]. We make use of a *collision-resistant* hash function H and a commitment scheme Com.

Advanced Building Blocks. Here we discuss 4 primitives at a high-level: (a) 3-party joint message passing (jmp) from [55], with minor modifications (b) a related 4-party jmp primitive, (c) oblivious product evaluation (OPE) and (d) distributed zero-knowledge protocol.

3-Party Joint Message Passing (jmp3). The jmp primitive from [55] allows two parties P_i, P_j holding a common value v, to send it to a party P_k such that either P_k receives the correct v, or TTP is identified. For our purpose, we trivially modify their protocol to give out a dispute pair (DP) instead of a TTP to all the 4 parties. In [55], the jmp primitive is invoked for sending each value independently and the verification is amortized over many sends. Their protocol allows for such a decoupling due to its asymmetry and a pre-specified order of verification. For our protocol however, postponing verification causes security issues. Specifically, batching the verification of different layers of the

circuit together allows an adversary to follow a strategy which ensures that DP comprises of two (semi) honest parties. This is contrary to the requirement that DP must include the malicious party. To avoid this problem, we compress the send and verification of jmp so that an optimistic (no error) run takes one round and batch them together for many instances corresponding to a pair of senders. That is, a pair of parties, say P_i, P_j invoke jmp to send a vector $\overrightarrow{\mathsf{v}}$ to P_k, and in parallel verification of correctness takes place. We call the modified variant as jmp3. It requires an amortized communication of 1 element.

4-Party Joint Message Passing (jmp4). jmp4 allows two parties P_i, P_j holding a common value v, to send it to the other two parties P_k, P_m such that, either both the parties receive the correct v or all the parties identify DP.

Notation 2 *We refer to the invocation of* $\mathsf{jmp3}(P_i, P_j, \mathsf{v}, P_k)$ *as "*P_i, P_j jmp3-send v *to* P_k*" and* $\mathsf{jmp4}(P_i, P_j, \mathsf{v}, P_k, P_m)$ *as "*P_i, P_j jmp4-send v *to* P_k, P_m*".*

Oblivious Product Evaluation (OPE). OPE (adapted from [53]) allows two parties holding $x \in \mathbb{Z}_{2^\lambda}$ and $y, z \in \mathbb{Z}_{2^\lambda}$ respectively, to compute an additive sharing of the product xy, such that one party holds $xy + z \in \mathbb{Z}_{2^\lambda}$ and the other holds $z \in \mathbb{Z}_{2^\lambda}$. We rely on techniques from [42,53] to obtain an OPE for λ-bit strings by running a total of λ 1-out-of-2 OTs on λ bits strings. In this work, we instantiate OTs using the protocol from Ferret [72], which incurs an (amortized) cost of 0.44 bits for generating one random correlated OT (amortized over batch generation of 10^7 correlated OTs). We can obtain an input-dependent OT (using techniques from [12,48]) at an additional cost of 2 elements and 1 bit. This results in a cost of $2\lambda + 1.44$ bits per OT. So an instantiation of OPE requires an amortised cost of $\lambda(2\lambda + 1.44)$ bits and 4 rounds. Note that we use OT in a black-box manner; thus, any improvement in OT, will improve the efficiency of our construction. Further, although OPE can be realised with oblivious linear evaluation (OLE), we opt for the approach of [53] due to better efficiency of OT. Hence, any improvements in OLE that surpasses OT can be translated to improving our protocol by replacing OPE with OLE.

Distributed Zero-knowledge (ZK). To verify a party P_i's correct behaviour, we extend the distributed zero-knowledge proofs introduced first in [18] offering *abort* security, and further optimized by Boyle *et al.* [20] to provide *robust* verification of degree-two relations. Such proofs involve a single prover and multiple verifiers, where the prover intends to prove the correctness of its (degree-two) computation over data which is *additively* distributed among the verifiers. In [20], the authors provide a distributed ZK protocol with sub-linear proof size, which is adapted for the verification of messages sent in a 3PC protocol with one corruption. Their ZK protocol extends in a straightforward manner to the 4-party case with one malicious corruption and one semi-honest corruption in the FaF model where a dispute pair is identified in case the verification fails. This is identical to extending the distributed ZK protocol to the case of 4 parties with 1 malicious corruption in the classical model and does not incur any overhead in our setting. Since the protocol in [20], and correspondingly ours, is constructed over fields, to support verification over rings, as in [20] verification operations

are carried out on the extended ring $\mathbb{Z}_{2^\lambda}/f(x)$, which is the ring of all polynomials with coefficients in $\mathbb{Z}_{2^\lambda}$ modulo a polynomial f, of degree η, irreducible over $\mathbb{Z}_{2^1}$. Each element in $\mathbb{Z}_{2^\lambda}$ is lifted to a η-degree polynomial in $\mathbb{Z}_{2^\lambda}[x]/f(x)$ (which results in blowing up the communication by a factor η).

The details of the building blocks, including functionalities, protocols and proofs appear in the full version.

3 Necessity of Oblivious Transfer

Here, we show that semi-honest OT is necessary for a $\mathtt{FaF}$-secure protocol. Our claim holds for $n \leq 2t + 2h^*$ which subsumes the case of n-party (t, h^*)-$\mathtt{FaF}$ security with optimal threshold of $t + h^* + 1$ and $2t + h^* + 1$ for abort and GOD [3] respectively, and the special case of 4-party $(1, 1)$-$\mathtt{FaF}$ security. The theorem and proof sketch are given below.

Theorem 3. *An n-party (t, h^*)-$\mathtt{FaF}$ secure (abort) protocol with $n \leq 2t + 2h^*$ implies 2-party semi-honest OT.*

Proof. Without loss of generality, we consider $n = 2t+2h^*$. Let π_f be an n-party (t, h^*)-$\mathtt{FaF}$ secure abort protocol for computing the function $f((m_0, m_1), \perp, \ldots, \perp, b) = (\perp, \perp, \ldots, \perp, m_b)$. We construct a 2-party semi-honest OT protocol π_{OT} between a sender P_S with inputs (m_0, m_1) and a receiver P_R with input b using π_f. In π_{OT}, P_S emulates the role of $Q_S = \{P_1, P_2, \ldots P_{t+h^*}\}$ while P_R emulates the role of $Q_R = \{P_{t+h^*+1}, \ldots, P_n\}$ to run π_f. P_R outputs the same m_b as output by party P_n which it emulates while P_S outputs $\perp$. To prove the security of π_{OT}, we construct simulators $\mathcal{S}_S$ and $\mathcal{S}_R$ that generate the view of P_S and P_R respectively from their inputs.

Let P_S be corrupted by the semi-honest adversary $\mathcal{A}_{\mathsf{OT}}$ and let $H = \{P_1, \ldots, P_{h^*}\}$ and $I = Q_S \backslash H$. We now map $\mathcal{A}_{\mathsf{OT}}$ to an adversarial strategy against π_f as follows. Consider a malicious adversary $\mathcal{A}$ for π_f that corrupts parties in I but does not deviate from the protocol (since $\mathcal{A}_{\mathsf{OT}}$ is semi-honest). However, it sends the random tape, inputs and messages of all parties in I to every other party in H at the end of the protocol execution. Note that such an attack of leaking the view of the maliciously corrupted parties to the semi-honest adversary is valid in the $\mathtt{FaF}$ model. The semi-honest adversary $\mathcal{A}_{\mathcal{H}}$ for π_f runs $\mathcal{A}_{\mathsf{OT}}$ on the joint view of the parties in $I \cup H$ ($\mathcal{A}_{\mathcal{H}}$ receives the view of parties in I from $\mathcal{A}$) and outputs the same value as $\mathcal{A}_{\mathsf{OT}}$. Since $|I| = t$ and $|H| = h^*$, the security of π_f ensures that there exist simulators $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{A}_{\mathcal{H}}}$ corresponding to the adversaries $\mathcal{A}$ and $\mathcal{A}_{\mathcal{H}}$. We construct the simulator $\mathcal{S}_S$ to run $\mathcal{S}_{\mathcal{A}}$ followed by $\mathcal{S}_{\mathcal{A}_{\mathcal{H}}}$ on P_S's input (m_0, m_1) and output the view generated by $\mathcal{S}_{\mathcal{A}_{\mathcal{H}}}$. Since $\mathcal{A}_{\mathcal{H}}$ receives the view of parties in I, the view generated by $\mathcal{S}_{\mathcal{A}_{\mathcal{H}}}$ includes the view of parties in $I \cup H$. Note that although $\mathcal{A}$ considered is malicious in π_f, it is emulated by a semi-honest adversary in the outer π_{OT} protocol and hence does not deviate from the protocol. Corresponding to such adversarial strategy of $\mathcal{A}$, the simulator $\mathcal{S}_{\mathcal{A}}$ may need to choose the input on behalf of $\mathcal{A}$. A simulator for a semi-honest adversary is not allowed to choose the input on behalf of the adversary, as discussed in [46].

However, since the parties in I controlled by the adversary $\mathcal{A}$ do not have inputs for f, this does not pose a problem in the proof and $\mathcal{S}_S$ can thus use $\mathcal{S}_\mathcal{A}$.

This proves the necessity of semi-honest-OT for (t, h^*)-$\mathtt{FaF}$ secure protocol where $t+h^* < n \leq 2t+2h^*$. Moreover, the sufficiency of OT for the same is given in [3, Theorem 4.1]. The detailed constructions of the OT protocol, simulators and the corresponding indistinguishabilty argument appears in the full version.

Corollary 1. *An n-party (t, h^*)-$\mathtt{FaF}$ secure abort protocol with $n = t + h^* + 1$ implies 2-party semi-honest OT.*

Corollary 2. *An n-party (t, h^*)-$\mathtt{FaF}$ secure GOD protocol with $n = 2t + h^* + 1$ implies 2-party semi-honest OT.*

Both Corollary 1 and 2 follow directly from Theorem 3. For Corollary 1, the sender emulates $t+h^*$ parties and the receiver emulates 1 party. For the corrupt receiver we consider $I = \phi$ and $H = \{P_n\}$. For Corollary 2, the sender emulates $t + h^*$ parties and the receiver emulates $t + 1$ parties. For the corrupt receiver we consider $I = \{P_{t+h^*+1}, \ldots, P_{2t+h^*}\}$ and $H = \{P_n\}$.

4 Input Sharing and Reconstruction

To enforce security, we perform computation on secret-shared data. This section starts with the various sharing semantics we use, followed by a sharing and a reconstruction protocol for secret-shared computation. We further present an efficient batch reconstruction for a second type of sharing, which in turn, will act as the primary building block for our efficient (batch) multiplication protocol.

We begin with the motivation for the choice of our sharing semantics. As explained earlier, we rely on RSS with threshold 2 to tackle view-leakage attack where the semi-honest adversary may receive the view of the malicious adversary. Instead of using RSS directly, we slightly augment our sharing to RSS-share a random mask and make the masked secret available to all. This sharing style makes the online cost of a multiplication one reconstruction instead of two. If we use RSS directly for sharing a secret, then relying on the Beaver's multiplication triple technique [11], we would need reconstructing $\mathsf{x}+\alpha_\mathsf{x}$ and $\mathsf{y}+\alpha_\mathsf{y}$, where x, y are the inputs and $\alpha_\mathsf{x}, \alpha_\mathsf{y}$ are the corresponding random masks. However, as per the latter sharing, we include the masked values $\beta_\mathsf{x} = \mathsf{x} + \alpha_\mathsf{x}$, $\beta_\mathsf{y} = \mathsf{y} + \alpha_\mathsf{y}$ along with RSS shares of α_x and α_y respectively in our sharing semantics. So the only reconstruction needed now is that of the masked valued of xy. This idea goes back to [66]. We now describe the sharing semantics.

1. $[\cdot]$-*sharing:* A value $\mathsf{v} \in \mathbb{Z}_{2^\lambda}$ is said to be $[\cdot]$-shared (additively shared) among parties P_i, P_j, if P_i holds $[\mathsf{v}]_i \in \mathbb{Z}_{2^\lambda}$ and P_j holds $[\mathsf{v}]_j \in \mathbb{Z}_{2^\lambda}$ such that $\mathsf{v} = [\mathsf{v}]_i + [\mathsf{v}]_j$.
2. $\langle\cdot\rangle$-*sharing:* A value $\mathsf{v} \in \mathbb{Z}_{2^\lambda}$ is said to be $\langle\cdot\rangle$-shared among $\mathcal{P}$ if, each pair of parties (P_i, P_j), where $1 \leq i < j \leq 4$, holds $\langle\mathsf{v}\rangle_{ij} \in \mathbb{Z}_{2^\lambda}$ such that $\mathsf{v} = \sum_{(i,j)} \langle\mathsf{v}\rangle_{ij}$. This is equivalent to RSS of a value among 4 parties with

threshold 2. Note that since $\langle \mathsf{v} \rangle_{ij}$ represents the common share held by P_i, P_j, throughout the protocol we assume the invariant that $\langle \mathsf{v} \rangle_{ij} = \langle \mathsf{v} \rangle_{ji}$, for all $1 \leq i < j \leq 4$. $\langle \mathsf{v} \rangle_i$ denotes P_i's share in the $\langle \cdot \rangle$-sharing of v.

3. $[\![\cdot]\!]$-*sharing:* A value $\mathsf{v} \in \mathbb{Z}_{2^\lambda}$ is $[\![\cdot]\!]$-shared if
   - there exists $\alpha_\mathsf{v} \in \mathbb{Z}_{2^\lambda}$ that is $\langle \cdot \rangle$-shared amongst $\mathcal{P}$ and
   - each $P_i \in \mathcal{P}$ holds $\beta_\mathsf{v} = \mathsf{v} + \alpha_\mathsf{v}$.

   Note that the value α_v acts as the mask for v. We denote by $[\![\mathsf{v}]\!]_i$, P_i's share in the $[\![\cdot]\!]$-sharing of v.

Note that all these sharings are linear i.e. given sharings of values $a_1, \ldots, a_m$ and public constants $c_1, \ldots, c_m$, sharing of $\sum_{i=1}^{m} c_i a_i$ can be computed non-interactively for an integer m.

4.1 $[\![\cdot]\!]$-sharing: Sharing and Reconstruction

Sharing. Protocol $[\![\cdot]\!]$-Sh either allows a party P_s to share a value v or ensures dispute pair (DP) detection. To generate $[\![\mathsf{v}]\!]$, in the preprocessing phase, P_s together with every other party P_i, samples a random $\langle \alpha_\mathsf{v} \rangle_{si} \in \mathbb{Z}_{2^\lambda}$, while P_s samples a random $\langle \alpha_\mathsf{v} \rangle_{ij} \in \mathbb{Z}_{2^\lambda}$ with every pair of parties P_i, P_j. This allows P_s to learn α_v in clear. In the online phase, P_s computes $\beta_\mathsf{v} = \mathsf{v} + \alpha_\mathsf{v}$ and sends it to P_t. Parties P_s, P_t then jmp4-send β_v to the rest. This step either allows the sharing to complete or identifies a DP. The protocol appears in Fig. 1.

Protocol $[\![\cdot]\!]$-Sh

- **Input, Output:** P_s has v. The parties output $[\![\mathsf{v}]\!]$.
- **Primitives:** jmp4-send (§2).

Preprocessing: P_s together with (a) P_i, for each $P_i \in \mathcal{P} \backslash P_s$ samples random $\langle \alpha_\mathsf{v} \rangle_{si} \in \mathbb{Z}_{2^\lambda}$; (b) $P_i, P_j \in \mathcal{P} \backslash P_s$, where $i \neq j$, samples random $\langle \alpha_\mathsf{v} \rangle_{ij} \in \mathbb{Z}_{2^\lambda}$.

Online: P_s computes $\beta_\mathsf{v} = \mathsf{v} + \sum_{(i,j)} \langle \alpha_\mathsf{v} \rangle_{ij}$ and sends it to P_t, where $s \neq t$. P_s, P_t jmp4-send β_v to $\mathcal{P} \backslash \{P_s, P_t\}$.

Fig. 1: $[\![\cdot]\!]$-sharing a value

Reconstruction. Protocol $[\![\cdot]\!]$-Rec allows parties to reconstruct v from $[\![\mathsf{v}]\!]$ such that either v is obtained by all the parties or a DP is identified. As observed, a party misses three shares of $\langle \alpha_\mathsf{v} \rangle$, which are needed for reconstructing v, each of which is held by two other parties. To reconstruct v towards a party P_s, in the preprocessing each pair (P_i, P_j) jmp3-send a commitment of their common share $\mathsf{Com}(\langle \alpha_\mathsf{v} \rangle_{ij})$ to P_s. The common source of randomness (generated via the shared key setup) can be used for generating the commitments, so that it is identically generated by both the senders. Then in the online phase all the parties open the commitments sent during preprocessing. P_s first reconstructs

α_{v} from consistent openings and then computes $\mathsf{v} = \beta_{\mathsf{v}} - \alpha_{\mathsf{v}}$. Due to the use of jmp3, the preprocessing may fail, however once it is successful the online phase is robust. Hence, this reconstruction ensures fairness i.e. either all or none receive the output (in the latter case DP has been identified). In case the reconstruction protocol terminates with a dispute pair, to extend security to GOD, parties perform the circuit evaluation using a semi-honest 2PC protocol.

Protocol $[\![\cdot]\!]$-Rec

- **Input, Output:** The parties input $[\![\mathsf{v}]\!]$. The parties output v.
- **Primitives:** jmp4-send and Com (§2).

Preprocessing: Each P_i, P_j, $1 \leq i < j \leq 4$ compute $\mathsf{Com}(\langle \alpha_{\mathsf{v}} \rangle_{ij})$ and jmp4-send it to P_k, P_m.

Online: Each P_i, P_j, $1 \leq i < j \leq 4$ open $\mathsf{Com}(\langle \alpha_{\mathsf{v}} \rangle_{ij})$ to P_k and P_m. Each P_i accepts the opening consistent with the commitment received earlier and computes $\mathsf{v} = \beta_{\mathsf{v}} - \sum_{(i,j)} \langle \alpha_{\mathsf{v}} \rangle_{ij}$.

Fig. 2: Reconstructing a $[\![\cdot]\!]$-shared value

4.2 $\langle\cdot\rangle$-sharing: Reconstruction

In our MPC protocol, for each multiplication gate we require to reconstruct a $\langle\cdot\rangle$-shared value in the online phase. Note that a party misses three shares of $\langle \mathsf{v} \rangle$ needed for reconstruction, each of which is held by two other parties. For reconstructing v towards all the parties, naively, each pair can jmp4-send their common share to the other two parties. This requires 6 invocations of jmp4, thus a communication of 12 elements. Since reconstructing $\langle\cdot\rangle$-shared value is the only communication bottleneck in the online phase of our multiplication protocol, it is imperative to improve its efficiency.

Taking a step towards this, we allow two parties, say P_3, P_4 (w.l.o.g) to first reconstruct v and use jmp4-send to send it to the other two parties. Naively, the reconstruction towards P_3, P_4 requires 6 instances of jmp3-send, three per party to send its missing shares. To improve the communication cost further, we improve the cost of the second instance of the reconstruction of v (towards P_4 in our case), to 2 jmp3-send instances, leveraging the communication already done for the reconstruction towards P_3. This reduces the communication cost to 7 elements. Our protocol appears in Fig. 3.

Since jmp3 is defined for a vector of values, in $\langle\cdot\rangle$-Rec, parties execute reconstruction of multiple values together. The protocol is described for a single value. Extending it to a vector is straightforward. In our multiplication protocol, this translates to reconstruction of the output of all multiplication gates in a level of the circuit simultaneously.

Note that we can reconstruct v from $[\![\mathsf{v}]\!]$ using $\langle\cdot\rangle$-Rec to reconstruct α_{v}. However, while $[\![\cdot]\!]$-Rec offers fairness, $\langle\cdot\rangle$-Rec does not. This implies if we use

$\langle\cdot\rangle$-Rec for the final output, it is possible that the adversary gets the output while the honest parties do not. Further, when the computation is rerun in 2PC mode, the adversary can use a different input and obtain another evaluation, thus breaching security.

Protocol $\langle\cdot\rangle$-Rec

- **Input, Output:** The parties input $\langle \mathsf{v} \rangle$. The parties output v.
- **Primitives:** jmp3-send and jmp4-send (§2).

Online:

- **(Reconstructing v to P_3.)** P_1, P_2 jmp3-send $\langle \mathsf{v} \rangle_{12}$ to P_3. P_1, P_4 jmp3-send $\langle \mathsf{v} \rangle_{14}$ to P_3. P_2, P_4 jmp3-send $\langle \mathsf{v} \rangle_{24}$ to P_3.
- **(Reconstructing v to P_4.)** P_1, P_3 jmp3-send $\langle \mathsf{v} \rangle_{13}$ to P_4. P_2, P_3 jmp3-send $\langle \mathsf{v} \rangle_{12} + \langle \mathsf{v} \rangle_{23}$ to P_4.
- **(Reconstructing v to P_1, P_2.)** P_3, P_4 jmp4-send $\mathsf{v} = \sum_{(i,j)} \langle \mathsf{v} \rangle_{ij}$ to P_1, P_2.

Fig. 3: Reconstructing a $\langle\cdot\rangle$-shared value

5 Multiplication

In this section, we present a multiplication protocol. Taking a top-down approach, we first present our multiplication protocol relying on a triple generation protocol in a black-box way. We then conclude with a triple generation protocol. To gain efficiency, several layers of amortisation are used. We mention them on the go and summarise at the end of the section.

5.1 Multiplication Protocol

The multiplication protocol (Fig. 4) allows parties to compute $[\![\mathsf{z}]\!]$, given $[\![\mathsf{x}]\!]$ and $[\![\mathsf{y}]\!]$, where $\mathsf{z} = \mathsf{x} \cdot \mathsf{y}$. We reduce this problem to that of reconstructing a $\langle\cdot\rangle$-shared value, assuming that parties have access to (a) $\langle\cdot\rangle$-sharing of a multiplication triple $(\alpha_\mathsf{x}, \alpha_\mathsf{y}, \alpha_\mathsf{x}\alpha_\mathsf{y})$ for random α_x, α_y and (b) $\langle\cdot\rangle$-sharing of a random α_z. Both the requirements are input (i.e. x, y) independent and can be fulfilled during the preprocessing phase. The former requirement is obtained via a triple generation protocol tripGen (Fig. 6), discussed subsequently. The latter requirement can be achieved non-interactively using the shared key setup. The reduction works as follows. The random and independent secret α_z is taken as the mask for the $[\![\cdot]\!]$-sharing of product z. Since α_z is already $\langle\cdot\rangle$-shared, to complete $[\![\mathsf{z}]\!]$, parties only need to obtain the masked value $\beta_\mathsf{z} = \mathsf{z} + \alpha_\mathsf{z}$. Since β_z takes the following form $\beta_\mathsf{z} = \mathsf{z} + \alpha_\mathsf{z} = \mathsf{xy} + \alpha_\mathsf{z} = (\beta_\mathsf{x} - \alpha_\mathsf{x})(\beta_\mathsf{y} - \alpha_\mathsf{y}) + \alpha_\mathsf{z} = \beta_\mathsf{x}\beta_\mathsf{y} - \beta_\mathsf{x}\alpha_\mathsf{y} - \beta_\mathsf{y}\alpha_\mathsf{x} + \alpha_\mathsf{x}\alpha_\mathsf{y} + \alpha_\mathsf{z}$ and the parties hold $\langle\alpha_\mathsf{x}\rangle$, $\langle\alpha_\mathsf{y}\rangle$, $\langle\alpha_\mathsf{x}\alpha_\mathsf{y}\rangle$, $\langle\alpha_\mathsf{z}\rangle$, and $\beta_\mathsf{x}, \beta_\mathsf{y}$ in clear, the parties hold $\langle\beta_\mathsf{z}\rangle$. Parties thus need to reconstruct β_z. To leverage the amortised cost of $\langle\cdot\rangle$-Rec, we batch many multiplications together. While for simplicity, we present the

protocol in Fig. 4 for a single multiplication, our complexity analysis accounts for amortization.

Protocol mult

- **Input and Output:** The input is $[\![\mathsf{x}]\!], [\![\mathsf{y}]\!]$ and the output is $[\![\mathsf{xy}]\!]$.
- **Primitives:** tripGen (§5.2; Fig. 6) and $\langle\cdot\rangle$-Rec (§4.2; Fig. 3).

Preprocessing:

– Each P_i, P_j where $1 \leq i < j \leq 4$ sample random $\langle\alpha_{\mathsf{z}}\rangle_{ij} \in \mathbb{Z}_{2^\lambda}$.

– Parties invoke tripGen with inputs $\langle\alpha_{\mathsf{x}}\rangle, \langle\alpha_{\mathsf{y}}\rangle$ to obtain $\langle\alpha_{\mathsf{x}}\alpha_{\mathsf{y}}\rangle$.

Online:

– Each P_i, P_j for $1 \leq i < j \leq 4$ and $(i,j) \neq (1,2)$ compute $\langle\beta_{\mathsf{z}}\rangle_{ij}$ such that $\langle\beta_{\mathsf{z}}\rangle_{ij} = -\beta_{\mathsf{x}}\langle\alpha_{\mathsf{y}}\rangle_{ij} - \beta_{\mathsf{y}}\langle\alpha_{\mathsf{x}}\rangle_{ij} + \langle\alpha_{\mathsf{x}}\alpha_{\mathsf{y}}\rangle_{ij} + \langle\alpha_{\mathsf{z}}\rangle_{ij}$.

– P_1, P_2 compute $\langle\beta_{\mathsf{z}}\rangle_{12} = \beta_{\mathsf{x}}\beta_{\mathsf{y}} - \beta_{\mathsf{x}}\langle\alpha_{\mathsf{y}}\rangle_{12} - \beta_{\mathsf{y}}\langle\alpha_{\mathsf{x}}\rangle_{12} + \langle\alpha_{\mathsf{x}}\alpha_{\mathsf{y}}\rangle_{12} + \langle\alpha_{\mathsf{z}}\rangle_{12}$.

– Parties invoke $\langle\cdot\rangle$-Rec to obtain β_{z}.

Fig. 4: Multiplication Protocol

5.2 Triple Generation Protocol

As a building block to our triple generation protocol, we first present a distributed multiplication protocol, where two distinct pairs of parties hold inputs to the multiplication and the goal is to additively share the product between the pairs. We build on this protocol to complete our triple generation.

Distributed Multiplication Protocol Let P_i, P_j hold a and P_k, P_m hold b. The goal of a distributed multiplication is to allow P_i, P_j compute c^1 and P_k, P_m to compute c^2 such that $c^1 + c^2 = ab$. To achieve this, P_k and P_m locally sample c^2 (using one-time key setup) then parties engage in an instance of OPE (§2) where P_i, P_j and respectively P_k, P_m enact the receiver's and sender's role.

– P_i, P_j as the receivers input a and output either c^1 or DP.

– P_k, P_m as the senders input b, $-c^2$ and output either $\perp$ or DP.

Since the pair of receivers $\{P_i, P_j\}$ hold identical inputs and use a shared source of randomness, their corresponding messages in the underlying protocol for OPE realisation will be identical. They send their messages to the senders via an instance of jmp4. Recall that the jmp4 primitive ensures that a message commonly known to two sender parties is either communicated correctly to both the receiving parties, or a dispute pair DP is identified. In the former case, the pair of senders $\{P_k, P_m\}$, having the same input and receiver's message, will prepare identical sender messages as a part of OPE and communicate to the receivers via another instance of jmp4 primitive, resulting in either a successful communication of the sender message to the receivers $\{P_i, P_j\}$ or identification of DP. In the former case, OPE is concluded successfully. Note that the verification of

jmp4 tackles any malicious behaviour, thus relying on semi-honest OPE suffices. Otherwise, DP is identified and the pair is guaranteed to include the malicious party. If fairness is the end goal, the protocol can terminate at this stage. Otherwise, it switches to an execution of a semi-honest 2PC (such as ABY2.0 [66]) with the parties outside DP to achieve the stronger guarantee of GOD.

Protocol disMult

- **Input and Output:** P_i, P_j hold a. P_k, P_m hold b. The first pair outputs c^1, the second pair c^2 such that $c^1 + c^2 = ab$. Otherwise the parties output DP.
- **Primitives:** OPE and jmp4 (§2).

– P_k and P_m locally sample a value c^2, using their shared key.
– P_i, P_j execute OPE with input a using jmp4 to send messages to P_k, P_m.
– P_k, P_m execute OPE with inputs $(b, -c^2)$ using jmp4 to send messages to P_i, P_j.

Fig. 5: Distributed Multiplication Protocol

Triple Generation Protocol The triple generation protocol allows parties holding $\langle \alpha_x \rangle, \langle \alpha_y \rangle$ to generate $\langle \alpha_x \alpha_y \rangle$. We write the product $\alpha_x \alpha_y$ as below, consisting of 36 summands, categorizing them into three types as below and as shown in Table 3.

For the summands in type S_0, no single party holds the two constituent shares of α_x, α_y. For the summands in S_1, exactly one party holds the two constituent shares, and lastly for the summands in S_2, exactly two parties hold the the two constituent shares. Note that there are 6 summands each, of the types S_0 and S_2 and 24 summands of type S_1. To generate $\langle \alpha_x \alpha_y \rangle$, we generate $\langle \cdot \rangle$-sharing of each summand of $\alpha_x \alpha_y$ and then sum them up to obtain $\langle \alpha_x \alpha_y \rangle$. The task of generating $\langle \cdot \rangle$-sharing for an individual summand differs based on the class it belongs to.

$$\begin{aligned} \alpha_x \cdot \alpha_y &= \sum_{\substack{(i,j) \\ 1 \le i < j \le 4}} \langle \alpha_x \rangle_{ij} \cdot \sum_{\substack{(k,m) \\ 1 \le k < m \le 4}} \langle \alpha_y \rangle_{km} \\ &= \underbrace{\sum_{\substack{(i,j) \\ 1 \le i < j \le 4}} \langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ij}}_{S_2} + \underbrace{\sum_{\substack{(i,j,k) \\ i,j,k \in [4]}} \langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ik}}_{S_1} + \underbrace{\sum_{\substack{(i,j),(k,m) \\ 1 \le i,k < j,m \le 4}} \langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{km}}_{S_0} \end{aligned} \tag{1}$$

Summands of S_2. Each summand in this type can be computed locally by 2 parties. For instance, $\langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ij}$ can be computed by P_i and P_j. Denoting $\langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ij}$ as τ_{ij}, $\langle \tau_{ij} \rangle$ is computed as follows:

$$\begin{aligned} &P_i, P_j \text{ set } \langle \tau_{ij} \rangle_{ij} = \langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ij} \text{ and} \\ &P_u, P_v \text{ set } \langle \tau_{ij} \rangle_{uv} = 0, \forall (u,v) \neq (i,j) \end{aligned} \tag{2}$$

Summands of S_1. Each summand here can be computed locally by a single party. For instance, $\langle \alpha_x \rangle_{ij} \langle \alpha_y \rangle_{ik}$ can be computed by P_i alone. Then P_i's goal is

to share this amongst the four parties so that one share is held by both P_i, P_k and the other by P_j, P_m. That is, for $\delta_i, \delta_i^1, \delta_i^2$ with $\delta_i = \delta_i^1 + \delta_i^2 = \langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ik}$, P_i, P_k intend to obtain δ_i^1 and P_j, P_m intend to obtain δ_i^2. The pairings $\{P_i, P_k\}$ and $\{P_j, P_m\}$ for various parties are done to balance the share count across the parties. We say that $\{P_i, P_k\}$ and respectively $\{P_j, P_m\}$ pair up for P_i's instance. Given this, $\langle \delta_i \rangle$ can be computed as (we set $k = i + 3$):

$$\begin{aligned} &P_i, P_k \text{ set } \langle \delta_i \rangle_{ik} = \delta_i^1, \ P_j, P_m \text{ set } \langle \delta_i \rangle_{jm} = \delta_i^2 \\ &P_u, P_v \text{ set } \langle \delta_i \rangle_{uv} = 0, \text{ for all } (u, v) \neq (i, k), (j, m) \end{aligned} \tag{3}$$

Now to achieve the above distribution of additive shares (δ_i^1, δ_i^2), P_i, P_j, P_m first locally sample δ_i^2 (using the shared key setup) and further, P_i computes and sends δ_i^1 to P_k. To keep P_i's misbehaviour in check, P_i is made to prove in zero-knowledge the correctness of its computation. With this high-level idea, we introduce two cost-cutting techniques.

	$\langle \alpha_{\mathsf{x}} \rangle_{12}$	$\langle \alpha_{\mathsf{x}} \rangle_{13}$	$\langle \alpha_{\mathsf{x}} \rangle_{14}$	$\langle \alpha_{\mathsf{x}} \rangle_{23}$	$\langle \alpha_{\mathsf{x}} \rangle_{24}$	$\langle \alpha_{\mathsf{x}} \rangle_{34}$
$\langle \alpha_{\mathsf{y}} \rangle_{12}$	S_2	S_1	S_1	S_1	S_1	S_0
$\langle \alpha_{\mathsf{y}} \rangle_{13}$	S_1	S_2	S_1	S_1	S_0	S_1
$\langle \alpha_{\mathsf{y}} \rangle_{14}$	S_1	S_1	S_2	S_0	S_1	S_1
$\langle \alpha_{\mathsf{y}} \rangle_{23}$	S_1	S_1	S_0	S_2	S_1	S_1
$\langle \alpha_{\mathsf{y}} \rangle_{24}$	S_1	S_0	S_1	S_1	S_2	S_1
$\langle \alpha_{\mathsf{y}} \rangle_{34}$	S_0	S_1	S_1	S_1	S_1	S_2

Table 3: The summands of $\alpha_{\mathsf{x}} \cdot \alpha_{\mathsf{y}}$ with category $\{S_0, S_1, S_2\}$

First, recall that there are 24 summands in S_1 and every P_i is capable of locally computing 6 of them. We combine the above procedure for 6 summands together. That is, δ_i^1, δ_i^2 are additive shares of $\delta_i = \sum_{(j,k)} \langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ik}$. This cuts our cost by 1/6th. Next, leveraging the malicious-minority and non-collusion of the malicious and semi-honest adversaries (implied by FaF model), we customise disZK (§2) of [20] to prove that $\sum_{(j,k)} \langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ik} - \delta_i^1 - \delta_i^2 = 0$. As per the need of such ZK, each term in the statement is additively shared amongst P_j, P_k, P_m and is possessed in entirety by the prover P_i. For instance, $\langle \alpha_{\mathsf{x}} \rangle_{ij}$ is additively shared amongst P_j, P_k, P_m with P_j's share as $\langle \alpha_{\mathsf{x}} \rangle_{ij}$ and the shares of the rest set to 0. Similarly for other shares of α_{x} and α_{y}. δ_i^2 is additively shared amongst P_j, P_k, P_m with P_j's share as δ_i^2 and the shares of the rest set to 0. Lastly, δ_i^1 is additively shared amongst P_j, P_k, P_m with P_k's share as δ_i^1 and the shares of the rest set to 0. If the disZK is successful, then P_i, P_k output δ_i^1 and P_j, P_m output δ_i^2, using which $\langle \delta_i \rangle$ an be computed as above. Otherwise, the disZK returns a dispute pair. This is executed for every party's collection of S_1 summands.

Summands of S_0. No single party can compute the summands in this category. For instance, $\langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{km}$ cannot be computed by any of the parties locally. We invoke the distributed multiplication protocol disMult (Fig. 5) for each such term, where the common input of $\{P_i, P_j\}$ and $\{P_k, P_m\}$ are $\langle \alpha_{\mathsf{x}} \rangle_{ij}$ and $\langle \alpha_{\mathsf{y}} \rangle_{km}$ respectively and their respective outputs are $\gamma_{ij,km}^1, \gamma_{ij,km}^2$, in case of success, or a dispute pair. Denoting $\gamma_{ij,km} = \gamma_{ij,km}^1 + \gamma_{ij,km}^2 = \langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{km}$, the parties can now generate $\langle \gamma_{ij,km} \rangle$ as:

$$\begin{aligned} &P_i, P_j \text{ set } \langle \gamma_{ij,km} \rangle_{ij} = \gamma_{ij,km}^1, \ P_k, P_m \text{ set } \langle \gamma_{ij,km} \rangle_{km} = \gamma_{ij,km}^2 \\ &P_u, P_v \text{ set } \langle \gamma_{ij,km} \rangle_{uv} = 0, \text{ for all } (u, v) \neq (i, j), (k, m) \end{aligned} \tag{4}$$

Protocol tripGen

- **Input and Output:** The parties input $\langle \alpha_{\mathsf{x}} \rangle, \langle \alpha_{\mathsf{y}} \rangle$. The output is $\langle \alpha_{\mathsf{x}} \alpha_{\mathsf{y}} \rangle$.
- **Primitives:** Protocol disMult (§5.2) and Protocol disZK (§2).

– For each of the 6 summands of the form $\langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{km}$ for unordered pairs $\{P_i, P_j\}$ and $\{P_k, P_m\}$ in S_0, the parties execute disMult with the inputs of $\{P_i, P_j\}, \{P_k, P_m\}$ as $\langle \alpha_{\mathsf{x}} \rangle_{ij}$ and $\langle \alpha_{\mathsf{y}} \rangle_{km}$ respectively. The parties either output DP or $\{P_i, P_j\}, \{P_k, P_m\}$ output $\gamma^1_{ij,km}$ and $\gamma^2_{ij,km}$ respectively. In the latter case, parties compute $\langle \gamma_{ij,km} \rangle$ as shown in Equation 4.

– For every i, consider *all* the 6 summands of the form $\langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ik}$ for unordered pairs $\{P_i, P_j\}$ and $\{P_i, P_k\}$ in S_1.

1. The parties P_i, P_j, P_m locally sample δ_i^2 (using the shared key setup).
2. P_i computes and sends $\delta_i^1 = \sum_{(j,k)} \langle \alpha_{\mathsf{x}} \rangle_{ij} \cdot \langle \alpha_{\mathsf{y}} \rangle_{ik} - \delta_i^2$ to P_k.
3. Parties invoke disZK to verify if $\sum_{(j,k)} \langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ik} - \delta_i^1 - \delta_i^2 = 0$. If disZK returns success, then P_i, P_j, P_k, P_m output $\langle \delta_i \rangle$ as shown in Equation 3. Otherwise, output the DP returned by disZK.

– For each of the 6 summands of S_2, of the form $\langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{ij}$, parties compute $\langle \tau_{ij} \rangle$-sharing as shown in Equation 2.

– Every P_r for every $s \neq r$ computes

$$\langle \alpha_{\mathsf{x}} \alpha_{\mathsf{y}} \rangle_{rs} = \sum_{u,v:u \neq v} \langle \tau_{u,v} \rangle_{rs} + \sum_{1 \leq \ell \leq 4} \langle \delta_\ell \rangle_{rs} + \sum_{\substack{u,v:u \neq v \\ p,q:p \neq q}} \langle \gamma_{uv,pq} \rangle_{rs}$$

Fig. 6: Triple Generation Protocol

5.3 Summary

Amortizations We summarise the various layers of amortization we use to get the best efficiency of our protocols. First, given a circuit with ℓ multiplication gates, the triple generation protocol creates $\langle \cdot \rangle$-sharing of ℓ triples at one go. All the summands of the form $\langle \alpha_{\mathsf{x}} \rangle_{ij} \langle \alpha_{\mathsf{y}} \rangle_{km}$ from S_0 category across all the ℓ instances use jmp4 for communication, whose verification is inherently batched for amortization. Next, the distributed ZK used for tackling the summands in S_1 can be used in an amortized sense as well. Recall that corresponding to a single triple generation, every P_i runs a single instance of distributed ZK to tackle 6 summands in its possession. However, we can extend this to accommodate 6ℓ summands across all the ℓ triples to achieve 40 bits of statistical security while working over a ring, by performing verification on the extended ring [20,1]. This means that we need to run overall 4 distributed ZK, one for every party. These cover all the amortizations done in the triple sharing protocol which constitutes the preprocessing of the multiplication protocol. The online phase of the multiplication protocol too exploits amortization of the batch $\langle \cdot \rangle$-reconstruction protocol. In the MPC protocol, we thus proceed level by level and execute all the multiplications placed in a level at one go.

Achieving Fairness. To obtain fairness, we can stop immediately after sensing a dispute. This means, in some cases, the effort needed for identifying a dispute pair, beyond sensing a dispute (which only says something is wrong and nothing beyond), can be slashed. For instance, in jmp4 parties can terminate immediately upon detecting conflict without identifying a dispute pair.

6 $(1,1)$-FaF Secure 4PC Protocol

Our complete protocol (4PC) realising the 4PC functionality ($\mathcal{F}_{\text{4PC-FaF}}$) for evaluating a circuit in the $(1,1)$-FaF security model with fairness and GOD is described here as a composition of the protocols discussed so far. Formal details appear in the full version of the paper. Recall that our protocol is cast in the preprocessing paradigm. In the preprocessing phase, for each input gate u, parties execute the preprocessing of $[\![\cdot]\!]$-Sh to precompute $\langle \alpha_{\mathsf{u}} \rangle$. Further, for each multiplication gate with input wires u, v and output wire w, parties obtain $\langle \alpha_{\mathsf{w}} \rangle$ and $\langle \alpha_{\mathsf{u}} \alpha_{\mathsf{v}} \rangle$ by running the preprocessing of mult. This computation is done in parallel for all the multiplication gates. Finally, for each output gate of the circuit, parties execute the preprocessing phase of $[\![\cdot]\!]$-Rec. This completes the preprocessing.

In the online phase, parties evaluate the circuit gate-by-gate in a predetermined topological order. For each input gate u, they execute the online phase of $[\![\cdot]\!]$-Sh to obtain β_{u}. Addition gates are handled locally. For each multiplication gate with input wires u, v and output wire w, parties perform the online phase of mult to compute β_{w}. Finally, they reconstruct the value of an output wire w, using the online phase of $[\![\cdot]\!]$-Rec. As mentioned in §2, we batch the verification of all the parallel instances of jmp3 and jmp4 respectively for every pair of parties, and perform it with the send in the same round. In case of malicious behaviour in these instances, additionally at most 2 rounds are required to identify a dispute pair. The above protocol either succeeds or a dispute pair is identified, which includes the malicious party. This construction achieves fairness.

To attain GOD without incurring additional overhead in the online phase, we follow the approach of segmented evaluation described in [31]. Specifically, we divide the circuit into segments, and the protocol proceeds as described in a segment-by-segment manner with topological order. As in the case of our fair protocol, either the execution of a segment completes successfully, or a dispute pair is identified. In the latter case, the segment where the fault occurs and all the segments following it are evaluated using a semi-honest 2PC, which is executed by the parties outside the dispute pair. Using this approach, only the segment where the fault occurs incurs the cost of 2PC in addition to the cost of our fair protocol. Hence, this overhead which is limited to a single segment is insignificant. The cost of evaluating the subsequent segments is solely that of the semi-honest 2PC which we instantiate with [66]. Note that in segmented evaluation of the circuit, the output of a segment acts as the input to the following segment. Hence, rerunning the segment where malicious behaviour was detected requires the outputs from the prior segment with 4PC sharing semantics to be translated to 2PC sharing semantics. However, due to a threshold of 2

in the 4PC, no pair of parties hold all the components of sharing corresponding to any secret. This necessitates interaction among parties. Suppose S_m is the segment where malicious activity is detected and w.l.o.g. $\{P_3, P_4\}$ is identified as the dispute pair, which means the evaluation till segment S_{m-1} happened correctly. W.l.o.g let z be the output of the segment S_{m-1} which is also an input to the segment S_m. Since the evaluation of S_{m-1} was correct, all 4 parties have the correct $[\![\cdot]\!]$ sharing of z, which comprises of β_z and $\langle \alpha_\mathsf{z} \rangle$. But to rerun S_m with $\{P_1, P_2\}$, they need the 2PC sharing of z. However, $\{P_1, P_2\}$ miss the $\langle \alpha_\mathsf{z} \rangle_{34}$ component which is common to P_3, P_4 and hence cannot obtain the 2PC sharing of z locally. Making P_3, P_4 send this value to P_1 or P_2 or both does not suffice. Since either P_3 or P_4 is malicious, the malicious party can send a wrong value which will lead to an inconclusive state for $\{P_1, P_2\}$, failing to achieve the end goal of 2PC sharing. To address this problem, we resort to the same idea as that of $[\![\cdot]\!]$-Rec. That is, for each output wire z of all the segments, all pairs of parties P_i, P_j commit to their common share $\langle \alpha_\mathsf{z} \rangle_{ij}$ in the preprocessing phase and jmp4-send the commitment to the other two parties. Now with the commitments established, parties in the dispute pair can send the opening corresponding to their respective commitments to the remaining two parties. In the above example, this corresponds to P_3, P_4 sending the opening of their commitments which contains $\langle \alpha_\mathsf{z} \rangle_{34}$ to P_1, P_2. Following this, P_1, P_2 can decide the correct value of $\langle \alpha_\mathsf{z} \rangle_{34}$ based on a valid opening, which is guaranteed to exist since one of P_3, P_4 is honest. Note that sending $\langle \alpha_\mathsf{z} \rangle_{34}$ does not breach privacy since the malicious party can anyway send this value to other parties as a part of view-leakage, which is handled by our sharing semantics. Now P_1 sets its 2PC additive share $[\alpha_\mathsf{z}]_1 = \langle \alpha_\mathsf{z} \rangle_{12} + \langle \alpha_\mathsf{z} \rangle_{13} + \langle \alpha_\mathsf{z} \rangle_{14}$ and P_2 sets $[\alpha_\mathsf{z}]_2 = \langle \alpha_\mathsf{z} \rangle_{23} + \langle \alpha_\mathsf{z} \rangle_{24} + \langle \alpha_\mathsf{z} \rangle_{34}$, where $\alpha_\mathsf{z} = [\alpha_\mathsf{z}]_1 + [\alpha_\mathsf{z}]_2$. Note that $(\beta_\mathsf{z}, [\alpha_\mathsf{z}]_1)$ and $(\beta_\mathsf{z}, [\alpha_\mathsf{z}]_2)$ is a valid 2PC sharing of z as per the semantics of [66]. However, as we describe below, this does not suffice.

Observe that the preprocessing of 2PC is performed along with the preprocessing of our 4PC protocol. Therefore, the value of mask corresponding to a wire z may differ in these two scenarios. To perform the 2PC execution of the circuit, we need to use the mask values selected during preprocessing for the 2PC. Let α'_z be the mask corresponding to wire z in the 2PC and $[\alpha'_\mathsf{z}]_1$ and $[\alpha'_\mathsf{z}]_2$ be the shares corresponding to P_1, P_2 respectively. Thus, the sharing of z requires to be updated according to α'_z, which essentially means updating the corresponding masked value, say β'_z such that $\beta'_\mathsf{z} = \mathsf{z} + \alpha'_\mathsf{z} = (\beta_\mathsf{z} - \alpha_\mathsf{z}) + \alpha'_\mathsf{z}$. Towards this, P_1 computes $v_1 = \beta_\mathsf{z} - [\alpha_\mathsf{z}]_1 + [\alpha'_\mathsf{z}]_1$ and sends it to P_2. Similarly, P_2 computes $v_2 = [\alpha'_\mathsf{z}]_2 - [\alpha_\mathsf{z}]_2$ and sends it to P_1. Then P_1, P_2 locally obtain $\beta'_\mathsf{z} = v_1 + v_2$ to complete the required 2PC sharing of z. Note that since both P_1, P_2 are (semi) honest, they send the correct values. Further, sending v_1 or v_2 does not breach privacy since they can learn these values from their own shares (for example, P_1 can compute v_2 given its shares $\beta_\mathsf{z}, \beta'_\mathsf{z}, [\alpha_\mathsf{z}]_1, [\alpha'_\mathsf{z}]_1$). The security of protocol 4PC as per the functionality $\mathcal{F}_{\text{4PC-FaF}}$ is stated below.

Theorem 4. *Assuming collision resistant hash functions and semi-honest OT exists, protocol* 4PC *realizes* $\mathcal{F}_{\text{4PC-FaF}}$ *with computational* $(1,1)$-FaF *security.*

Security against a mixed adversary. A closely related notion of security is that of a mixed adversary [26,37,39,8,41,47] which can simultaneously corrupt a subset of t parties maliciously and a disjoint subset of h^* parties in a semi-honest manner. In contrast to the FaF model, the adversary here is centralized. Consequently, the mixed security model allows the view of semi-honest parties to be available to the adversary while determining a strategy for the malicious parties. Although the mixed adversarial model might seem to subsume FaF, Alon et al. [3] showed that (t, h^*) mixed security does not necessarily imply (t, h^*)-FaF security. Given this, we constructed a 4PC protocol which is secure in the FaF model. However, we go a step beyond and show that our protocol is also secure against a $(1, 1)$-mixed adversary. For this, the crucial observation is that our protocol can withstand the scenario where the malicious adversary is provided with the view of semi-honest parties, which essentially captures the mixed adversarial model. Refer to the full version for details.

7 Applications and Benchmarks

This section focuses on evaluating the performance of QuadSquad. We first evaluate the performance of MPC and draw comparisons to concretely efficient traditional MPC protocols that come closest to our setting. We then establish the practicality of QuadSquad via the application of secure liquidity matching and PPML for neural network inference. We refer the readers to the full version for a detailed discussion on the benchmarking environment, secure protocols for the applications considered and analysis of performance bottlenecks. The source code of our implementation is available at quadsquad.

Environment. Benchmarks are performed over WAN using n1-standard-32 instances of Google Cloud, with machines located in East Australia (M_0), South Asia (M_1), South East Asia (M_2), and West Europe (M_3). The machines are equipped with 2.2GHz Intel Xeon processors supporting hyper-threading and 128GB RAM. Average bandwidth and round-trip time (rtt) between pair of machines was observed to be 180 Mbps and 158.31 ms respectively; though these values vary depending on the regions where the machines are located.

Software. We implement our protocol in C++17 using EMP toolkit [71]. Since we use OT as a black-box, it can be instantiated with any state-of-the-art OT protocol such as [29]. Since the public implementation of [29] is not available, we use EMP toolkit's Ferret OT [72]. We use the NTL library [68] for computation over ring extensions for disZK protocol. [53] and [31] are benchmarked in the MP-SPDZ [51] framework. Due to the unavailability of implementation of [56], we estimate its performance from microbenchmarks. We instantiate the collision resistant hash function with SHA256 and the PRF with AES-128 in counter mode. Computation is performed over $\mathbb{Z}_{2^{64}}$ for [31,56] and QuadSquad, and over $\mathbb{Z}_p$ for [53] where p is a 64-bit prime. We set the computational security parameter to $\kappa = 128$ and ensure statistical security of at least 2^{-40} for all the protocols. In particular, we set the degree of the polynomial modulus of the extended ring $\eta = 47$. We report the average value over 20 runs for each experiment.

Benchmarking Parameters. As a measure of performance, we report the online and overall (preprocessing + online) communication per party and latency for a single execution. To capture the combined effect of communication and round complexity, we additionally use *throughput* (tp) as a benchmark parameter, following prior works [56,60,67]. Here, tp denotes the number of operations (triples for 4PC preprocessing and multiplications for 4PC online protocol) that can be performed in one second.

7.1 Performance of 4PC QuadSquad

We compare the performance of our 4PC to Fantastic Four [31], Tetrad [56] and MASCOT [53]. We evaluate a circuit comprising 10^6 multiplication gates distributed over different depths. Recall that the online communication cost of our GOD protocol is similar to the fair protocol due to segment-wise evaluation. Hence, we only report the cost of the fair protocol for online comparison.

The performance of the online phase appears in Table 4. The latency of our protocol (fair and GOD) is up to 3.5× higher compared to honest majority protocol of [56] and the abort variant of [31]. This captures the overhead required to achieve the stronger notion of FaF-security. On the other hand, the dishonest majority protocol of [53] bears an overhead of 4.5× to 1.01× compared to ours.

Depth	Ref.	Online		
		Latency(s)	Comm. (MB)	tp
1	Fantastic Four	2.86	12.00	350066.51
	Tetrad	1.44	6.00	692947.87
	MASCOT	13.88	24.00	72023.80
	QS	2.94	14.00	340506.67
20	Fantastic Four	4.04	12.00	247286.04
	Tetrad	2.95	6.00	339321.22
	MASCOT	25.94	24.00	38554.22
	QS	7.42	14.00	134752.73
100	Fantastic Four	11.26	12.00	88771.32
	Tetrad	9.28	6.00	107764.43
	MASCOT	74.48	24.00	13425.63
	QS	30.92	14.00	32337.66
1000	Fantastic Four	87.82	12.00	11387.21
	Tetrad	80.52	6.00	12419.36
	MASCOT	289.69	24.00	3451.94
	QS	287.71	14.06	3475.69

Table 4: Online costs for evaluating circuits with 10^6 mult gates over various depths. (**QS** denotes QuadSquad.)

The performance of the preprocessing depends only on the number of multiplication gates, not on the circuit depth. Hence, only the communication cost and throughput are reported in Table 5. [31] does not have preprocessing and is thus, not included. Further, unlike the online phase, Table 5 reports results for both fair and GOD variants independently since their performance in the preprocessing phase is different. The *communication* bottleneck in the preprocessing of QuadSquad is due to computing summands of S_0 which involves running six instances of disMult, while the *computational* bottleneck is due to computing the summands of S_1 which involves running four instances of disZK. We implement disZK using recursion as in [20] which ensures lower communication and computation costs at the expense of higher round complexity. Our benchmarks show that disMult always tends to have a higher latency than disZK and constitutes the performance bottleneck. Detailed discussion is provided in the full version.

The GOD variant requires running the preprocessing of [66] for every pair of parties which has an overhead of around 3 KB per multiplication gate per party. This approximately halves the throughput in the preprocessing phase when compared to the fair variant since the combined preprocessing across all [66] instances is akin to running six instances of disMult which in turn is the main bottleneck in fair preprocessing. With respect to throughput, [56] has the highest tp owing to its low communication costs while the tp of QuadSquad Fair is around 1.8× that of [53]. The tp of QuadSquad GOD is comparable to that of [53] despite a significantly lower communication cost because the implementation of [53] distributes the evaluation of OT instances across the available threads while our implementation runs it in a single thread to allow running the disZK protocol in parallel.

Ref.	Comm. (KB)	tp
Tetrad	0.004	958918.39
MASCOT	67.6	4548.64
QS (Fair)	3.115	8051.27
QS (GOD)	6.22	3934.01

Table 5: Preprocessing phase cost for generating a triple.

7.2 Applications

We consider applications of secure liquidity matching and PPML inference. Before describing these and evaluating their performance via QuadSquad, we describe the building blocks designed for the same.

Building blocks Each of these applications requires designing new building blocks, as described in Table 2. Specifically, we develop the following building blocks: sharing and reconstruction for SOC setting, dot product (DotP), dot product with truncation (DotPTr), conversion to arithmetic sharing from a Boolean shared bit (Bit2A), bit extraction to obtain Boolean sharing of the most significant bit (msb) from an arithmetic shared value (BitExt), bit injection to obtain arithmetic sharing of $\mathsf{b} \cdot \mathsf{v}$ from a Boolean sharing of a bit b and the arithmetic sharing of v (BitInj). Inclusion of these blocks makes QuadSquad a comprehensive framework. The details of the constructions and the complexity analysis are discussed in the full version.

Liquidity matching Secure liquidity matching involves executing a privacy-preserving variant of the **gridlock** algorithm. This algorithm identifies a set of transactions among banks which can be executed while ensuring that all the banks possess sufficient liquidity to process them. The gridlock algorithm can be considered for the following scenarios (i) the source and the destination banks of the transactions are open (non-private) (sodoGR), (ii) the source is open, but the destination is hidden (secret) (sodsGR), and (iii) the source and the destination are hidden (ssdsGR). A secure realization for liquidity matching was provided in [7], albeit via traditionally secure MPC. Given the sensitive nature of financial data involved in liquidity matching, clearly, FaF-security is more apt. Hence, we focus on designing FaF-secure protocols for the same. Further, with respect to

#banks	#transactions	Online		Fair Total*		GOD Total*	
		Latency(s)	Comm. (KB)	Latency(s)	Comm. (MB)	Latency(s)	Comm. (MB)
256	50	5.23	21.28	9.46	4.75	10.35	14.56
	100	5.46	23.71	10.22	5.53	10.64	16.11
	250	5.70	32.04	10.56	7.87	11.06	20.77
	500	5.94	47.97	10.95	11.77	11.61	28.53
	1000	6.18	81.76	11.49	19.56	12.45	44.07
1024	50	5.70	74.41	10.72	7.98	12.17	44.91
	100	5.94	76.59	10.99	8.76	12.47	46.46
	250	6.18	83.36	11.32	11.10	12.89	51.13
	500	6.42	96.13	11.71	15.0	13.43	58.88
	1000	6.66	124.36	12.26	22.79	14.28	74.41

Table 6: Liquidity matching

the three scenarios described above, note that in most practical cases hiding the transaction amount is sufficient. Hence, we consider only the sodoGR instance (details in the full version). However, we note that extending our techniques to the other two scenarios is also possible.

At a high level, the protocol is iterative where each iteration checks the feasibility of clearing a subset of transactions. The protocol terminates with a feasible set or reports a deadlock when no transactions can be cleared. Since the communication and computation costs are identical across all iterations, we benchmark the performance for a single iteration and report the costs in Table 6. We see similar trends as observed while evaluating the performance of MPC, where the GOD variant is on par with the fair variant with respect to the overall latency. Further, we observe that the latency of an iteration for both variants is within 15s even for a large number of banks and set of transactions.

PPML For the application of PPML inference, we consider the popularly used [56,55,70,67] Neural Network (NN) architectures, given below.

- *FCNN*: Fully-Connected NN consists of two hidden layers, each with 128 nodes followed by an output layer of 10 nodes. ReLU is applied after each layer.
- *LeNet*: This NN consists of 2 convolutional layers and 2 fully connected layers, each followed by ReLU activation function. Moreover, the convolutional layers are followed by an average-pooling layer.

The inference task is performed over the publicly available MNIST [57] dataset which is a collection of 28×28 pixel, handwritten digit images with a label between 0 and 9 for each. We note that our techniques easily extend to securely evaluating other NN architectures such as convolutional neural network (CNN) and VGG16 [69] used in other MPC-based PPML frameworks of [55,56,70].

We compare the performance of PPML inference via QuadSquad for the above mentioned NN with the honest majority protocols of [56] and [31]. PPML in the 4PC dishonest majority (malicious) setting has not been explored so far. The results of our experiments are summarised in Table 7. Note that the latency reported is obtained via a single instance of circuit evaluation, whereas the throughput is computed by running the inference on larger batches. Here, tp is the number of queries evaluated in a minute since inference over WAN requires

Network	Ref.	Online			Total*	
		Latency (s)	Comm. (MB)	tp (queries/min)	Latency (s)	Comm. (MB)
FCN	Fantastic Four	48.06	27.71	43.75	48.06	27.71
FCN	Tetrad	1.66	0.006	47099.05	2.38	0.02
FCN	**QS Fair**	6.00	0.022	3176.65	29.77	371.15
FCN	**QS GOD**	6.00	0.022	3176.65	44.49	746.46
LeNet	Fantastic Four	220.17	134.28	84.22	220.17	134.28
LeNet	Tetrad	2.45	0.36	787.09	3.25	0.91
LeNet	**QS Fair**	10.36	1.27	64.24	308.89	7251.73
LeNet	**QS GOD**	10.36	1.27	64.24	607.53	14868.07

Table 7: NN inference where **QS** denotes QuadSquad.

more than a second to complete. Our fair and GOD variants have an overhead of 3x–4x in performance respectively. However we provide a stronger adversarial model compared to [56]. The numbers in Table 7 for [31] from MP-SPDZ [51] are unexpectedly high. We suspect that this anomaly is due to the preprocessing cost of [31]. However, the benchmarks seem consistent with those reported in [31] and pinpointing the exact cause is challenging due to the vast MP-SPDZ codebase. It is worth noting that the communication cost of [31] per query for larger batch sizes decreases to 0.93 MB per party for FCN and 0.46 MB per party for LeNet. The QuadSquad protocols have higher cost in the preprocessing phase from using more expensive primitives like OT and the feature dependent preprocessing phase for dot-product. However, the comparable online performance to [56] and [31] and the stronger security model make it a viable practical option despite the overhead in preprocessing.

Acknowledgements

Arpita Patra would like to acknowledge financial support from DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-CPS) 2020-2025 and SERB MATRICS (Theoretical Sciences) Grant 2020-2023. Varsha Bhat Kukkala would like to acknowledge financial support from National Security Council, India. Nishat Koti would like to acknowledge support from Centre for Networked Intelligence (a Cisco CSR initiative) at the Indian Institute of Science, Bengaluru. Shravani Patil would like to acknowledge financial support from DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020-2025. The authors would also like to acknowledge the support from Google Cloud for benchmarking.

References

1. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In: TCC (2019)

2. Abspoel, M., Dalskov, A.P.K., Escudero, D., Nof, A.: An efficient passive-to-active compiler for honest-majority MPC over rings. In: ACNS (2021)
3. Alon, B., Omri, E., Paskin-Cherniavsky, A.: Mpc with friends and foes. In: CRYPTO (2020)
4. Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., Weinstein, O.: Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In: IEEE S&P (2017)
5. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: ACM CCS (2016)
6. Archer, D.W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J.I., Smart, N.P., Wright, R.N.: From keys to databases—real-world applications of secure multi-party computation. The Computer Journal (2018)
7. Atapoor, S., Smart, N.P., Alaoui, Y.T.: Private liquidity matching using mpc. IACR Cryptol. ePrint Arch. (2021)
8. Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: Secure mpc: laziness leads to god. In: ASIACRYPT (2020)
9. Baum, C., Damgård, I., Toft, T., Zakarias, R.W.: Better preprocessing for secure multiparty computation. In: ACNS (2016)
10. Baum, C., Orsini, E., Scholl, P.: Efficient secure multiparty computation with identifiable abort. In: Theory of Cryptography Conference (2016)
11. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO (1991)
12. Beaver, D.: Precomputing oblivious transfer. In: CRYPTO (1995)
13. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: Double your online spdz! improving spdz using function dependent preprocessing. In: ACNS (2019)
14. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC (1988)
15. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and taxes: a privacy-preserving social study using secure computation. IACR Cryptology ePrint Archive (2015)
16. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: ESORICS (2008)
17. Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis - (short paper). In: FC (2012)
18. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: CRYPTO (2019)
19. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent ot extension and more. In: CRYPTO (2019)
20. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: ACM CCS (2019)
21. Byali, M., Chaudhari, H., Patra, A., Suresh, A.: FLASH: fast and robust framework for privacy-preserving machine learning. PETS (2020)
22. Byali, M., Hazay, C., Patra, A., Singla, S.: Fast actively secure five-party computation with security beyond abort. In: ACM CCS (2019)
23. Byali, M., Joseph, A., Patra, A., Ravi, D.: Fast secure computation for small population over the internet. In: ACM CCS (2018)
24. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In: ACM CCSW@CCS (2019)

25. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. NDSS (2020)
26. Chaum, D.: The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities; crypto'89, lncs 435 (1990)
27. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: CRYPTO (2018)
28. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: ACM STOC (1986)
29. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In: Annual International Cryptology Conference. pp. 502–534. Springer (2021)
30. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority. In: CRYPTO (2018)
31. Dalskov, A., Escudero, D., Keller, M.: Fantastic four: Honest-majority four-party secure computation with malicious security. In: USENIX Security (2021)
32. Damgård, I., Escudero, D., Frederiksen, T.K., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure MPC over rings with applications to private machine learning. IEEE S&P (2019)
33. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: CRYPTO (2007)
34. Damgård, I., Orlandi, C., Simkin, M.: Yet another compiler for active security or: Efficient MPC over arbitrary rings. In: CRYPTO (2018)
35. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO (2012)
36. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
37. Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. Journal of the ACM (JACM) (1993)
38. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM Journal on Computing (1983)
39. Fitzi, M., Hirt, M., Maurer, U.: Trading correctness for privacy in unconditional multi-party computation. In: CRYPTO (1998)
40. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: EUROCRYPT (2017)
41. Ghodosi, H., Pieprzyk, J.: Multi-party computation with omnipresent adversary. In: PKC (2009)
42. Gilboa, N.: Two party rsa key generation. In: CRYPTO (1999)
43. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC (1987)
44. Gordon, S.D., Ranellucci, S., Wang, X.: Secure computation with low communication from cross-checking. In: ASIACRYPT (2018)
45. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority mpc. In: CRYPTO (2020)
46. Hazay, C., Lindell, Y.: A note on the relation between the definitions of security for semi-honest and malicious adversaries. IACR Cryptol. ePrint Arch. (2010)
47. Hirt, M., Mularczyk, M.: Efficient mpc with a mixed adversary. LIPIcs (2020)
48. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO (2003)

49. Ishai, Y., Kumaresan, R., Kushilevitz, E., Paskin-Cherniavsky, A.: Secure computation with minimal interaction, revisited. In: CRYPTO (2015)
50. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer–efficiently. In: CRYPTO (2008)
51. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: ACM CCS (2020)
52. Keller, M., Orsini, E., Scholl, P.: Actively secure ot extension with optimal overhead. In: CRYPTO (2015)
53. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: ACM CCS (2016)
54. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: EUROCRYPT (2018)
55. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In: USENIX Security (2021)
56. Koti, N., Patra, A., Rachuri, R., Suresh, A.: Tetrad: Actively secure 4pc for secure training and inference. arXiv preprint arXiv:2106.02850 (2021)
57. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010), http://yann.lecun.com/exdb/mnist/
58. Lindell, Y., Nof, A.: A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In: ACM CCS (2017)
59. Mazloom, S., Le, P.H., Ranellucci, S., Gordon, S.D.: Secure parallel computation on national scale volumes of data. In: USENIX Security (2020)
60. Mohassel, P., Rindal, P.: ABY^3: A mixed protocol framework for machine learning. In: ACM CCS (2018)
61. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: ACM CCS (2015)
62. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: IEEE S&P (2017)
63. Nordholt, P.S., Veeningen, M.: Minimising communication in honest-majority MPC by batchwise multiplication verification. In: ACNS (2018)
64. Orlandi, C.: Is multiparty computation any good in practice? In: IEEE ICASSP (2011)
65. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure mpc over $\mathbb{Z}/2^k$ from somewhat homomorphic encryption. In: CT-RSA (2020)
66. Patra, A., Schneider, T., Suresh, A., Yalame, H.: Aby2. 0: Improved mixed-protocol secure two-party computation. In: USENIX Security (2021)
67. Patra, A., Suresh, A.: BLAZE: Blazing Fast Privacy-Preserving Machine Learning. NDSS (2020)
68. Shoup, V.: NTL: A Library for doing Number Theory. https://libntl.org/ (2021)
69. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
70. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. arXiv preprint (2020)
71. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit (2016)
72. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: ACM CCS (2020)