The More The Merrier: Reducing the Cost of Large Scale MPC

S. Dov Gordon[1], Daniel Starin[2], and Arkady Yerukhimovich[3]

[1] George Mason University
gordon@gmu.edu
[2] Peraton Labs
dstarin@peratonlabs.com
[3] George Washington University
arkady@gwu.edu

Abstract. Secure multi-party computation (MPC) allows multiple parties to perform secure joint computations on their private inputs. Today, applications for MPC are growing with thousands of parties wishing to build federated machine learning models or trusted setups for blockchains. To address such scenarios we propose a suite of novel MPC protocols that maximize throughput when run with large numbers of parties. In particular, our protocols have both communication and computation complexity that decrease with the number of parties. Our protocols build on prior protocols based on packed secret-sharing, introducing new techniques to build more efficient computation for general circuits. Specifically, we introduce a new approach for handling *linear attacks* that arise in protocols using packed secret-sharing and we propose a method for unpacking shared multiplication triples without increasing the asymptotic costs. Compared with prior work, we avoid the $\log |C|$ overhead required when generically compiling circuits of size $|C|$ for use in a SIMD computation, and we improve over folklore "committee-based" solutions by a factor of $O(s)$, the statistical security parameter. In practice, our protocol is up to $10X$ faster than any known construction, under a reasonable set of parameters.

Acknowledgments

The authors would like to thank the anonymous reviewers for many helpful comments. Arkady Yerukhimovich and Dov Gordon are supported by NSF grant 1955264. Arkady Yerukhimovich is also supported by a Facebook Research Award.

1 Introduction

A major goal in secure multi-party computation (MPC) is to reduce the necessary communication and computation as a function of the number of parties participating in the protocol. Today, most protocols have costs growing linearly and even quadratically as the number of parties increases. This makes MPC prohibitively expensive for applications with very large numbers of parties, and all real-world applications of MPC have focused on use-cases with only a handful of parties. Academic experiments have pushed the limit to a few hundreds of participants [37]. However, today we have use-cases for MPC that naturally have thousands if not millions of participating parties. For example, in secure federated learning, thousands of low-resource devices wish to train a single machine learning model on their collective data [30]. Another example is that of distributed trusted setup for blockchains where (possibly) millions of miners may wish to participate in the protocol. Finally, a particularly well fitting application for such large-scale MPC is to generate offline material for smaller MPC computation. Such offline material is usually expensive to generate, but enables much faster *online* MPC computations once it is available.

To address such applications we construct a new MPC protocol that scales practically to hundreds of thousands of parties. The amortized cost of our protocol decreases as the number of parties increases, resulting in reduced cost for all parties as more parties join the computation – the more the merrier. This is especially true for applications such as blockchain setup, where the number of inputs does not grow with the number of computing parties. Specifically, assuming that at most $t \leq n(1/2 - \epsilon)$ parties are actively malicious, for any $0 < \epsilon < 1/2$, our protocol requires each party to send $O(|C|/n)$ field elements, where $|C|$ is the size of the circuit we wish to evaluate, and n is the number of parties. It requires each party to perform $O(\log n|C|/n)$ field operations.[4]

While ours is not the only protocol with communication and computation that diminishes with n, our communication complexity is better, asymptotically and concretely, than every construction that we know of. For example, one naive solution in this setting of a strong honest majority is to elect a small committee of size $O(s)$, where s is a statistical security parameter, independent of n, and to run any arbitrary MPC protocol among that committee alone. Technically, the average communication and computation cost reduces with n, since $n - O(s)$ parties that remain idle still reduce the average. However, the committee members carry the worst-case cost, $O(|C|)$. As we will see, even when we consider natural improvements to this approach that spread the worst-case cost among the parties, we out-perform such approaches by a factor of $O(s)$, which, concretely, could be as large as $40X$. More interestingly, several solutions using packed secret sharing are also known [13,23]: we outperform these constructions by a factor of $O(\log |C|)$ by avoiding the use of a general compiler from standard circuits to SIMD circuits. We provide a brief asymptotic comparison with specific prior constructions at the end of this introduction, in Subsection 1.2, and we provide a thorough analysis of these comparisons in Section 5.

We have implemented our protocol and executed it for small numbers of parties. To our knowledge, this is the first implementation with sub-linear costs: among existing implementations, the one with lowest concrete costs that we know of is by Furukawa and Lindell [20], which requires $O(|C|)$ communication per party, and does not benefit from an increase in the number of participants.

[4] We are ignoring terms that do not depend on $|C|$ or n.

Missing from our implementation is any large-scale deployment, which would introduce some major engineering challenges (not to mention social and/or financial challenges of finding participants). Most obviously, it would be very difficult to convincingly simulate the network environment.[5] Additionally, while a single Linux server can efficiently handle tens of millions of TCP connections [39], this is not currently supported by the OS, and requires extensive effort to implement. Nevertheless, our implementation allows us to precisely measure the computational cost for a million participants, which is the bottleneck in our construction. It also allows us to provide an exact extrapolation on the amount of data communicated. What is missing from this extrapolation is any handling of network variance or participant failure. Recognizing these caveats and extrapolating our performance, we estimate that we can achieve a throughput of 500 million multiplication triples per second[6] when 1 million parties participate, even over a relatively slow 10 megabit per second network.

One particularly appealing application for our protocol is as a service for generating offline material for smaller scale computations. The offline phase is the bottleneck for the majority of MPC protocols. If tens of thousands of users can be incentivized[7] to generate billions of computation triples over night, which might require less than 10 minutes of their time, these triples can be transferred to a small online committee for arbitrary computation the next day, enabling malicious-secure MPC, even with a dishonest majority, at the cost of an efficient online phase.

1.1 Technical overview

We now give a brief overview of the key ideas and building blocks behind our main protocol.

Damgård-Nielsen: In the honest-majority setting, most modern approaches to MPC build upon the multiplication protocol of Damgård-Nielsen (DN) [15]. This construction begins with an input-independent pre-processing phase in which the parties compute threshold double sharings of random values, $([r]_d, [r]_{2d})$, with thresholds d and $2d$, respectively. During the online phase of the computation, these double sharings can be used to perform multiplication on shared field elements at a communication cost of 2 field elements per party.

To generate these random sharings in the semi-honest setting, the DN protocol proceeds as follows. Each party P_i samples a random r_i and sends two

[5] Sometimes such experiments are run in cloud environments, which is useful for tens or hundreds of participants. However, we are interested in deployments involving tens or hundreds of *thousands* of participants. AWS has only 64 data centers, so testing with more parties than this would provide an inaccurate simulation of the network environment.

[6] This estimate is for a malicious-secure protocol that generates *unauthenticated* triples, which suffice for semi-honest computation in the online phase. In Section 4.1, we present a known result for converting these to authenticated triples. The throughput in that setting is closer to 70 million triples per second

[7] Or commanded by Google.

threshold sharings to every other party: $[r_i]_d, [r_i]_{2d}$. This costs n^2 total communication. Naively, each party could locally sum their received shares to recover a single double sharing of a random r, but the cost per sharing would be $O(n^2)$. Instead, each party assembles an n dimensional vector from the shares that they received, and multiplies this with a Vandermonde matrix, M. Assuming $h = O(n)$ honest parties, this allows them to extract h random double sharings, instead of a single one. This reduces the total communication cost to $O(n)$ per multiplication gate, requiring each party to send only a constant number of field elements per circuit gate.

Moreover, recent work by Genkin et al. [24] showed that this semi-honest DN protocol actually offers a stronger notion of security, called *security up to additive attacks*. This means that the protocol offers privacy against a malicious adversary and only allows an additive attack, wherein an adversary can add an error δ to the result of each multiplication. Several works have since shown how to leverage this property to efficiently achieve full malicious security (with abort) [33,10,20].

Packed secret sharing: To further reduce the cost by a factor of $O(n)$, we use packed secret sharing when we construct our double sharings [18,13,23]. In a standard Shamir sharing, the secret is encoded in the evaluation of a random polynomial at 0. However, if $t = (1/2 - \epsilon)n$, then even after fixing all t adversarial shares, there are still ϵn degrees of freedom in the polynomial (while maintaining the degree $d < n/2$). These can be used for encoding additional secret values. In a packed secret sharing scheme, a vector of $\ell = \epsilon n$ elements are encoded together in a single polynomial by interpolating a random polynomial through those ℓ points, and, as in Shamir sharing, providing point evaluations of the polynomial as shares. By packing $[\mathbf{r}] = [r_1], \ldots, [r_\ell]$ into a single polynomial, and performing the Vandermonde matrix multiplication on the packed shares, we can further reduce the communication and computational costs by a factor of $O(n)$, as was done by Damgård et al., and Genkin et al. [13,23]. When performing ℓ independent computations in parallel, this directly reduces both the communication and the computation by a factor of ℓ, without any cost. When performing only a single evaluation of the circuit (which is the setting we focus on), Damgård et al. [13] show how to compile a circuit C into one of size at most $O(|C| \log |C|)$, that can leverage packed sharing even in a single computation by parallelizing the multiplication gates in groups of size ℓ.

Unpacking the secret shares: Our first contribution is a new way to avoid this $\log |C|$ multiplicative overhead, even in the single execution setting. To do this, we avoid computing on packed values, and instead unpack the random values into fresh secret shares for later use by an "online committee".[8] That committee then uses these values to perform the online phase (i.e., the input de-

[8] There are advantages and disadvantages to varying the size of this committee, which we will discuss in depth in what follows. For now, we can assume that the online committee is in fact the entire network of n parties. In the "standard" approach to executing the online phase with n parties, the communication complexity is $O(|C|)$ per party. We will address this as well.

pendent portion of the computation). Unpacking the secrets allows us to directly compute an arbitrary circuit. However, if we unpack the ℓ secrets before sending them to the online committee, we ruin our sub-linear communication complexity: for every gate in the circuit, each of the n parties would need to send (at least) one share to the online committee, resulting in a per-party communication complexity of (at least) $O(|C|)$. Instead, we observe that unpacking requires only linear operations (i.e., we need to perform polynomial interpolation, which is a linear operation), and thus can be performed on secret shares of the packed secret sharing. So, the n parties *re-share* each of their packed shares with the online committee. Because they are still packed, the per-party communication of re-sharing is $O(|C|/n)$ instead of $O(|C|)$. The online committee then locally unpacks the "inner" threshold sharing by interpolating the polynomial that is "underneath" the outside sharing. This requires no interaction inside the online committee, and maintains the desired complexity.

Now consider the question of how to re-share the packed sharing. The naive choice here is to stick with a threshold secret sharing scheme, ensuring that after they unpack into the outer scheme, the parties maintain a threshold sharing of the DN double sharings, and can proceed exactly as in the DN protocol. Unfortunately, starting with $([\mathbf{r}]_d, [\mathbf{r}]_{2d})$, we do not know how to efficiently transfer $([\mathbf{r}]_s, [\mathbf{r}]_{2s})$ to an online committee of size s without incurring a communication overhead of $O(s)$: the cost of sending a Shamir share to a committee of size s is, seemingly, $O(s)$. In the case where the "committee" is of size n, this again ruins the claim of sub-linear complexity, even though there are only $O(|C|/n)$ values to be re-shared. Instead, we use an additive secret-sharing for our outer scheme. After establishing pairwise seeds between the senders and the receivers, we can compress the cost of re-sharing: to re-share some value x, party i computes $[x]_s = x + G(r_{i,1}) + \cdots + G(r_{i,s-1})$ and sends this to party s. All other parties fix their shares deterministically using their shared seed. This saves a factor of $O(s)$ in the communication cost, and when the committee size is n, it allows us to maintain the desired complexity. However, with an additive secret sharing, we can no longer use DN for multiplication. Instead, prior to transferring the packed values, we use our double sharings to create packed multiplication triples $([\mathbf{a}]_d, [\mathbf{b}]_d, [\mathbf{ab}]_d)$, and send additive shares of these packed, threshold shares to the online committee. The online committee unpacks these values into additive shares of multiplication triples by interpolating inside the additive sharing.

So far, we have ignored two important points: the cost of the online phase, and computational complexity of the protocol. Using n-out-of-n additive shares of multiplication triples, there is no clear way to achieve sub-linear communication complexity in the online phase. Furthermore, even if we ignore the cost of the online phase, unpacking a single multiplication triple requires $O(n \log n)$ field operations, which, having successfully reduced the communication complexity, becomes the new bottleneck in our protocol. The solution to both problems is one and the same: we parallelize the work of unpacking the triples, using many small committees, each of size $O(s)$, with the guarantee that each has at least one honest participant[9]. The computational cost of unpacking is now reduced to $O(s \log n|C|/n)$ field operations per party. Rather than transfer these shares to a single evaluation committee, we then split $|C|$ among the $O(n/s)$ committees, charging each with evaluating $O(s|C|/n)$ gates.[10] For the online phase, we use the online phase of the SPDZ protocol [16]. In order to use the generated triples in this protocol, each committee first "authenticates" their triples. This is very similar to the protocols found in prior work ([16,14,32,3]), and appears in Section 4.1. We stress that the authentication and the SPDZ online protocol have bottleneck communication linear in the number of gates to be evaluated.

Security against linear attacks: Genkin et al. [23] (building on Damgård et al. [13]) present a protocol that begins as ours does, with the construction of packed double sharings. (They do not construct packed triples for use in a malicious majority online committee, but instead use the packed values in a SIMD computation, incurring the $O(\log|C|)$ overhead described above.) In their security analysis, Genkin et al. observed that when computing with packed double sharings, one has to prevent a "linear attack" in which an adversary adds to the output of a multiplication gate some arbitrary linear combination of the values that are packed into the input shares. As the authors note, this is not something that is allowed by an additive attack, since the adversary does not know the packed secrets. Briefly, the linear attack works as follows (See Appendix B of [22]). Consider shares $[\mathbf{a}]_d = a_1, \dots, a_n$, and $[\mathbf{b}]_d = b_1, \dots, b_n$ for $n = 2d+1$. Let $\delta_1, \dots, \delta_{d+1}$ denote the Lagrange coefficients that recover the first element in the packed polynomial: $b^{(1)} = \sum_{i=1}^{d+1} \delta_i b_i$. Let $\gamma_1, \dots, \gamma_n$ denote the coefficients that recover the second element in the product polynomial: $a^{(2)}b^{(2)} = \sum_{i=1}^{n} \gamma_i a_i b_i$. If an adversary modifies the first $d+1$ shares of $\mathbf{a}$, $[\hat{\mathbf{a}}]^i = [\mathbf{a}]^i + \delta_i/\gamma_i$, then the reconstruction of $a^{(2)}b^{(2)}$ will equal $a^{(2)}b^{(2)} + b^{(1)}$. Such a dependency on $b^{(1)}$ is not allowed in an additive attack.

We avoid linear attacks using two critical properties of our construction. Observe that linear attacks work by modifying at least $d+1$ shares of an input wire to a multiplication gate. Thus, we ensure that this can never happen in our construction. First, as described earlier, we only use packed multiplication to produce triples, thus we only need to deal with a depth-1 circuit of multiplication gates. In particular, the output of any (packed) multiplication gate is never used as an input into another (packed) multiplication gate. Thus, even though an adversary can introduce an arbitrary additive attack to modify all the packed values in the resulting product, these modified values will never be used as an input to a (packed) multiplication gate. There is still one place where the adver-

[9] Technically, since we are selecting many such committees, to guarantee that they all have at least one honest party requires a union bound over the number of committees, resulting in committees of size $O(s + \log n)$. However, since $s > \log n$, we drop this $\log n$ term in our asymptotic notation. However, we point out that our experimental results in Section 6 do account for this union bound.

[10] This can be done regardless of the circuit structure, and does not require a wide circuit.

sary can introduce a linear attack: the original inputs to the triple generation (i.e., $\mathcal{A}$'s shares of a and b). However, we observe that this requires the adversary to change at least $d+1$ shares of a and thus requires changing at least one share held by an honest party. However, this is detectable by the honest parties if they (collectively) check the degree of the polynomial on which their shares lie. We, therefore, avoid linear attacks on the inputs by performing a degree check on the input shares of $[\mathbf{a}]$ and $[\mathbf{b}]$ using a standard procedure. Moreover, since we can batch all of these checks, doing so is (almost) for free.

1.2 Performance Comparisons

We give here a brief overview of how our protocol compares, asymptotically, to various other protocols. We will give a more detailed description of how we arrive at these numbers in Section 5, after presenting our protocol in full; here we only present the alternative protocols used in the comparisons, and present the results. Throughout, we assume we begin the computation with n parties, and $t < n/3$ corruptions by an actively malicious adversary. We use s to denote a statistical security parameter.

Performance metrics: When comparing the communication and computation costs of various protocols, we consider the *bottleneck complexities* [7]. This refers to the maximum communication sent or received[11] by any party taking part in the protocol, and the maximum computation performed by any one party. We believe that this is a more meaningful metric than average or total communication when analyzing protocols that use small committees, since most parties might do nothing after sharing their inputs, and therefore reduce the average artificially. In Section 5 we will give a more careful analysis, and, for completeness, we include there the total complexities.

Protocol Variants and Results: In what follows, we consider three variants of our protocol, as each provides insight into various aspects of our design. We begin by considering the simplest variant in which the full network of n participants unpack all $|C|$ triples, and all n parties participate in the online phase. We compare this with two folklore solutions in which a single committee is elected to perform the computation. We then describe two changes that strengthen our protocol, and compare the impact of each improvement to a similar improvement to these folklore solutions.

Baseline comparisons: In Figure 1, we compare the asymptotic behavior of our protocol to that of Furukawa and Lindell, which is also designed for a corruption threshold of $t < n/3$ [20]. Additionally, with $t < (1/2 - \epsilon)n$, an

[11] When analyzing total or average communication, there is no need to consider receiving complexity as the number of bits sent by all parties equals the number of bits received. But, when considering bottleneck complexity, one must make a distinction between the two. For example, if many parties send messages to one party, that party's receiving bandwidth becomes the bottleneck. In fact, there are MPC protocols such as [37] that are bottlenecked by the receiving bandwidth of some of the parties.

Single Online, Single Unpack				
	Ours	$t < n/3$ [20]	$t < n/2$ [10]	$t < n$ [16]
Comm	$O(\lvert C\rvert/n)$	$O(\lvert C\rvert)$	$O(\lvert C\rvert)$	$O(s\lvert C\rvert)$
Comp	$O(\log n\lvert C\rvert)$	$O(\log n\lvert C\rvert)$	$O(\log s\lvert C\rvert)$	$O(s\lvert C\rvert)$

Fig. 1. Asymptotic complexities for the offline phase of four protocols. In our variant, and in that of [20], all parties participate throughout. In the other two protocols, the state-of-the-art is run by a small committee: with an honest majority in column 4, and a malicious majority in column 5. All values measure bottleneck costs, rather than total costs. The online communication and computation cost for all four protocols is $O(C)$, and is not included.

old folklore approach is to choose a random committee of size $O(s)$ that is guaranteed, with probability $1 - \mathsf{negl}(s)$, to contain an honest majority. We consider performing the entire computation within that committee, using the state-of-the-art protocol for the $t < n/2$ setting by Chida et al. [10]. Finally, a similar folklore solution is to select a much smaller committee, with the weaker guarantee that at least one honest party is chosen, and then run the entire protocol using the state-of-the-art construction for the $t < n$ setting, such as the protocol by Damgård et al. [16], or any of the follow-up work. Furukawa and Lindell requires $O(|C|)$ communication per party, and the folklore solution using an honest majority committee requires $O(|C|)$ communication for each party on the committee, yielding the same bottleneck complexity. The folklore solution with a malicious majority has bottleneck complexity of $O(s|C|)$. In comparison, our offline phase has bottleneck complexity of $O(|C|/n)$. However, as mentioned previously, our online phase has bottleneck complexity of $O(|C|)$ as well, and our computational cost is higher than that of the first folklore solution. More importantly, computation is the major bottleneck in our performance. In the next two tables, we address these two issues, strengthening the folklore solutions in analogous ways.

Single Online, Distributed Unpack			
	Ours	$t < n/2$	$t < n$
Comm	$O(s\lvert C\rvert/n)$	$O(s\lvert C\rvert)$	$O(s^2\lvert C\rvert/n)$
Comp	$O(s\log n\lvert C\rvert/n)$	$O(s\log s\lvert C\rvert/n)$	$O(s^2\lvert C\rvert/n)$

Fig. 2. In our protocol variant, triples are unpacked in many parallel committees, and then transferred back to the full set of size n. Costs are measured through the transfer step, and exclude the cost of the online phase. In the column labeled $t < n/2$, many parallel committees, each with an honest majority, prepare double-sharings that will be used by a single online committee of size $O(s)$. In the column labeled $t < n$, many parallel committees, each with at least one honest participant, generate multiplication triples that will be used by the full network of size n. The online phase still requires $O(|C|)$ communication and computation for all three protocols, and is not included in the Table.

Distributed triple generation: Instead of unpacking all triples in a single committee, we improve our computational complexity by a factor of $O(n/s)$ by unpacking the triples in $O(n/s)$ small committees, each large enough to guarantee at least one honest member. The resulting triples are then transferred to a single online committee, still of size n.

Making an analogous change to the two folklore solutions,[12] we consider constructing triples in parallel, using $O(n/s)$ committees – honest majority committees in the first variant, and malicious majority committees in the second – each responsible for an $O(s/n)$ fraction of the pre-processing. When using honest majority committees, we analyze the case where the double sharings are transferred to a single online committee of size $O(s)$, which is guaranteed to have an honest majority with all but negligible probability. When using malicious majority committees, we distribute the multiplication triples to the entire network of size n. (In both cases, these choices minimize the bottleneck complexity. We consider using a small online committee, both for ourselves, and for the $t < n$ setting, in Section 5. This reduces total communication complexity, but introduces the bottleneck of having a small receiving committee.) We summarize these comparisons in Figure 2.

Asymptotically, all three protocols improve equally in computational cost, by a factor of $O(s/n)$. However, because honest majority committees are about 18X larger than malicious majority committees, in concrete terms, our computational cost is quite similar to protocol using honest majority committees. We discuss in more detail in Section 5.

Distributed online computation: To claim an end-to-end protocol that has sub-linear communication, we present a final set of protocol variants in which we distribute the online computation. These three protocol variants begin as in the previous set of protocols, but they stop short of transferring the multiplication triples to an online committee. Instead, each of the $O(n/s)$ committees is

Distributed Online						
	Ours		$t < n/2$		$t < n$	
	Offline	Online	Offline	Online	Offline	Online
Comm	$O(\|C\|/n)$	$O(s\|C\|/n)$	$O(s\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s\|C\|/n)$
Comp	$O(s \log n\|C\|/n)$	$O(s\|C\|/n)$	$O(s \log s\|C\|/n)$	$O(s\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s\|C\|/n)$

Fig. 3. Here we distribute the work done in the online phase, assigning $O(s|C|/n)$ gates to each of the $O(n/s)$ committees. In all three protocols, the material generated during pre-processing remains with the same committee for the online phase. The state of the online phase is transfered from one committee to the next.

[12] Note that when we assume $t < n/3$, we cannot construct committees of size $O(s)$ that have the same corruption threshold. We therefore do not consider running Furukawa and Lindell in parallel. We could do so with larger, committees, or we could consider a smaller threshold, but we feel the current set of comparisons suffices for demonstrating the value of our protocol.

responsible for $O(s|C|/n)$ triples (or double sharings, when $t < n/2$), and holds them until the online phase. Each committee is then responsible for a proportionate "chunk" of the circuit during the online evaluation. All three protocols benefit similarly from the reduced cost of the online evaluation.

In Figure 3, we have separated the online cost in this protocol. In some settings, it might make sense to stop the protocol after the offline phase, leaving the triples in their committees until they are needed by some other group for an online computation. For example, we can imagine using such a protocol in a service that sells computation triples. The cost of producing the triples diminishes with n, and the small unpacking committees can then transfer these triples, as needed, to paying customers. The receiving customers might have a malicious majority, and the cost of receiving the transfer would be minimal in comparison to the cost of securely generating the triples on their own. In this setting, the cost of receiving these triples (which show up in Figure 3), and the cost of using them in an online evaluation, can both be reasonably ignored by the triple service provider.

This last protocol variant combining distributed unpacking and a distributed online phase gives us the following main theorem.

Theorem 1 (Informal). *Assuming the existence of a PRG, our distributed online protocol generates $|C|$ multiplication triples with $O(\frac{|C|}{n})$ bottleneck complexity and $O(\frac{s \log n|C|}{n})$ bottleneck computation for a statistical security s, achieving security against a static, malicious adversary corrupting $t < n/3$ parties.*

1.3 Related Work

A full survey of the MPC literature is out of scope for this work, so we only discuss the results that are most directly relevant. Damgård and Nielsen [15] introduced the technique of using double sharings for realizing multiplication with $O(|C|)$ per-party communication in the honest majority setting. The technique has been used in many follow-up results [5,13,33,10,20,24,23] for a variety of efficient MPC protocols with honest majority. The technique of using multiplication triples was first introduced by Beaver [4]. Since then this technique has been extremely fruitful in the setting of malicious majority with a number of works proposing improved constructions of multiplication triples based on oblivious transfer (OT) [19,27,26] and based on somewhat-homomorphic encryption [16,32].

Several works have looked into achieving sub-linear communication for MPC in the honest majority setting. In particular, packed secret sharing was originally introduced by Franklin and Yung [18]. We use the packed version of the protocol due to Damgård and Nielsen [15] that was first presented in [13]. Other recent works such as Leviosa [25] and Ligero [2] have also used packed secret sharing to achieve efficiency for MPC-in-the-head [28] and zero-knowledge proofs. Damgård and Ishai [12] present a protocol for MPC in a client / server model that leverages packed secret sharing. Their construction has total communication complexity of $O(n|C| \log |C|)$, but when the number of clients is constant, they can remove a factor of n, achieving performance that improves as they introduce more servers. Very recently, Garay et al [21] study the feasibility of constructing sub-linear communication MPC and demonstrate some challenges to achieving sub-linear communication in MPC, especially when $|C|$ is small.

Using committees to speed up performance of MPC has been studied starting with the work of Bracha [8]. Since then several works [17,29,38,36,11] have looked into using committees to reduce communication in MPC over large numbers of parties. Additionally, another line of work has leveraged committees to improve the communication locality (i.e., how many parties a party must talk to as part of the protocol) of MPC protocols [6,9]. Finally, similar to our work, Scholl et al. [35] also consider how one can outsource triple generation when an online committee does not want to do the work on its own. They propose several approaches for outsourcing triple generation for MPC.

MPC protocols secure against additive attacks were introduced by Genkin et al. [24]. Then, Genkin, Ishai, and Polychroniadou [23] introduced the notion of a linear attack and showed that the semi-honest variant of packed Damgård-Nielsen given by Damgård et al. [13] is secure up to linear attack. For details see Genkin's thesis [22].

Finally, the notion of bottleneck complexity for MPC communication was originally introduced by Boyle et al. [7].

2 Preliminaries

2.1 Secret sharing

We use two types of secret sharing schemes. Packed secret sharing, in which a secret share is a single evaluation of a polynomial that encodes ℓ secrets, and additive secret sharing, in which a secret share is a random field element, and the shares sum to the secret. We only define notation here, and do not bother to define security or correctness of secret sharing.

Packed secret sharing:
We let $[\mathbf{r}]_d$ represent a secret sharing of $\mathbf{r}$ using a degree d polynomial, and we denote party P_i's share with $[\mathbf{r}]_d^i$.

$\mathsf{share}(d, \mathbf{r})$: outputs n shares of a degree d polynomial $(d, s^1, \ldots, s^n)$.

$\mathsf{reconstruct}(d, s^1, \ldots, s^j)$: given j shares, with $j > d$, output $\mathbf{r} \in \mathbb{F}^\ell$.

$\mathsf{reconstruct}(d, i, s^1, \ldots, s^j)$: given j shares, with $j > d$, output $r_i \in \mathbb{F}$, which is the value stored in the ith packing slot. In practice, we never extract a single value, as we can unpack all values using a pair of FFT / IFFT operations. However, notationaly, it is convenient to refer to the value recovered from a single slot.

Additive secret sharing:
We fix the size of our additive secret sharing to that of the online committee, Com. Our additive secret shares will always be a sum of $|\mathsf{Com}|$ field elements.

$\mathsf{aShare}(x)$: outputs $x_1, \ldots, x_{\mathsf{Com}}$ such that $x = \sum_i x_i$. In practice, this can be done by generating the first $\mathsf{Com} - 1$ shares pseudorandomly, and then choosing the final share as $x_{\mathsf{Com}} = x - \sum_{i=1}^{\mathsf{Com}-1} x_i$.

$\mathsf{reconstruct}(x_1, \ldots, x_n)$: outputs $\sum_i x_i$.

3 Multiplication Triple Generation

Overview: We describe a maliciously secure protocol with $t = \epsilon n$ corrupted parties, for $\epsilon \in [0, .5)$. We let $h = n - t$ denote the number of honest parties, and we let $\ell = n/2 - \epsilon n$ denote the packing parameter used in our secret sharing scheme. We will use polynomials of degree $d = \lfloor (n-1)/2 \rfloor$, and $d' = n - 1$. However, to simplify the notation, we will assume n is odd, and use d and $2d$ as the degrees of these polynomials. For simplicity, we describe only one protocol variant. We describe the protocol as having a single unpacking committee, denoted Com, and, we allow the committee size to be flexible. The extension to multiple, parallel unpacking committees is straightforward. If $|\mathsf{Com}| < n$, we elect a random subset of the parties to the committee. To ensure an honest member with probability 2^{-s}, it suffices to elect a committee of size $|\mathsf{Com}| \leq -s/\log \epsilon$. This is done by performing a secure coin flip among the n parties, and using the result to select parties at random, with replacement, $-s/\log \epsilon$ times. (If the committee happens to be smaller than Com because there are collisions in the sampling, this improves performance without impacting security.)

In steps 1 and 2 of the protocol, the parties exchange secret shares of random values, and use the public Vandermonde matrix M to extract $O(h)$ packed secret values. In step 3, they perform a degree check on all the shares sent in the 1st step, ensuring that the shares of all honest parties lie on a degree d polynomial, as expected. This limits any future modifications by the adversary to $t < d$ shares, eliminating the linear attack described previously, and limiting the adversary to simple additive attacks.

In Step 4, the parties perform local multiplication on their packed shares, doubling the degree of the polynomial, and blind the result using their share of $[\mathbf{r}]_{2d}$. This is sent to the dealer, P_0, who extracts the blinded, packed secrets, and reshares them, reducing the degree back down to d.

The resharing can be sent only to the first $d + 1$ parties. At this point in the protocol, any further deviations by the adversary will only create additive attacks, which we do not bother to prevent. (These attacks will be caught in the online phase, through a call to $\mathsf{macCheck}$, before the outputs are revealed.) We therefore do not need the redundancy of extra shares.

In step 6, the $d + 1$ parties receiving shares of the blinded product re-share their shares using the additive secret sharing scheme. These additive shares are sent to the committee(s) of size Com, which might be of size n or of size $O(s)$. In step 7, the committee unpacks their shares of the triples, homomorphically, resulting in additive shares $[a]_{\mathsf{a}}, [b]_{\mathsf{a}}$ and $[ab]_{\mathsf{a}}$.

The ideal functionality is presented in Figure 4. We note that the adversary is allowed to specify what output shares they would like to receive (but learns nothing about the shared value, which is still random). This is because we do not try to prevent a rushing attack in which the adversary sees the messages sent by the honest parties in the first step of the protocol, prior to fixing its own first message. This attack is benign, and is commonly allowed in prior work. More importantly, the adversary is allowed to specify an additive attack for every triple produced. Instead of receiving $[a]_\mathsf{a}, [b]_\mathsf{a}$ and $[ab]_\mathsf{a}$, the output committee instead receives $[a]_\mathsf{a}, [b]_\mathsf{a}$ and $[ab + \delta]_\mathsf{a}$

Ideal functionality for triple generation secure up to additive attack: $\mathcal{F}_{\text{triple}}$

Adversarial Behavior: The adversary gets to specify the share values they receive from the functionality. Additionally, the adversary inputs a vector $\boldsymbol{\delta} \in \mathbb{F}^{h\ell}$.

Input: The functionality takes no input.

Computation: For $i \in \{1, \ldots, h\ell\}$

1. Sample random a_i and b_i and compute $c_i = a_i \cdot b_i + \delta_i$.
2. For each $i \in [h\ell]$, additively secret-share a_i, b_i, c_i to Com.

Output: Each party in Com receives a share of a_i, b_i, and c_i for $i \in \{1, \ldots, h\ell\}$.

Fig. 4. This is a randomized functionality that outputs additive shares of $h\ell = O(n^2)$ multiplication triples to a designated committee, Com. Note that although no parties have input, and only $|\mathsf{Com}| \leq n$ parties have output, the protocol for realizing the functionality is always an n-party protocol. The produced triples are secure up to additive attack.

Theorem 2. *Let $t = \epsilon n$ for some $\epsilon \in [0, .5)$, let $h = n - t$, and let $\ell = n/2 - \epsilon n$. Then the protocol for unauthenticated triple generation in Figure 5 securely realizes the functionality of Figure 4 in the presence of an active, computationally unbounded adversary corrupting t parties.*

Proof. We define $\mathcal{A}$ to be the set of corrupted parties, and $\mathcal{H}$ to be the set of honest parties. The simulator begins by selecting randomness on behalf of all honest parties, and executes the protocol on their behalf, exchanging messages with the adversary as needed.

In the first step of the protocol, for each $P_i \in \mathcal{A}$, when P_i calls $\mathsf{share}(d, \tilde{\mathbf{r}}_i)$, $\mathsf{share}(d, \tilde{\mathbf{a}}_i)$ and $\mathsf{share}(d, \tilde{\mathbf{b}}_i)$, the simulator extracts the values $\tilde{\mathbf{r}}_i, \tilde{\mathbf{a}}_i, \tilde{\mathbf{b}}_i$ from the shares sent to the honest parties. If the values are not consistently defined (because the degree of the polynomial is too high), $\mathcal{S}$ sets the shares to $\perp$, and continues the simulation until the degree check in Step 3, at which point $\mathcal{S}$ always aborts. Otherwise, $\mathcal{S}$ computes $(\mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(h)}), (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(h)}), (\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(h)})$ using the extracted values, the honest randomness, and the local multiplication with the Vandermonde matrix, M.

Triple generation (Π_{triple})

Inputs: No parties have any input.

1. P_i samples random $\tilde{\mathbf{r}}_i$, $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_i$, and calls $\mathsf{share}(d, \tilde{\mathbf{r}}_i)$, $\mathsf{share}(2d, \tilde{\mathbf{r}}_i)$, $\mathsf{share}(d, \tilde{\mathbf{a}}_i)$ and $\mathsf{share}(d, \tilde{\mathbf{b}}_i)$.
2. Party P_i receives:
   $([\tilde{\mathbf{r}}_1]^i_d, \ldots, [\tilde{\mathbf{r}}_n]^i_d)$, $([\tilde{\mathbf{r}}_1]^i_{2d}, \ldots, [\tilde{\mathbf{r}}_n]^i_{2d})$, $([\tilde{\mathbf{a}}_1]^i_d, \ldots, [\tilde{\mathbf{a}}_n]^i_d)$ and $([\tilde{\mathbf{b}}_1]^i_d, \ldots, [\tilde{\mathbf{b}}_n]^i_d)$.
   and computes:
   $\left([\mathbf{r}^{(1)}]^i_d, \ldots, [\mathbf{r}^{(h)}]^i_d\right) = M \cdot ([\tilde{\mathbf{r}}_1]^i_d, \ldots, [\tilde{\mathbf{r}}_n]^i_d)$,
   $\left([\mathbf{r}^{(1)}]^i_{2d}, \ldots, [\mathbf{r}^{(h)}]^i_{2d}\right) = M \cdot ([\tilde{\mathbf{r}}_1]^i_{2d}, \ldots, [\tilde{\mathbf{r}}_n]^i_{2d})$,
   $\left([\mathbf{a}^{(1)}]^i_d, \ldots, [\mathbf{a}^{(h)}]^i_d\right) = M \cdot ([\tilde{\mathbf{a}}_1]^i_d, \ldots, [\tilde{\mathbf{a}}_n]^i_d)$, and
   $\left([\mathbf{b}^{(1)}]^i_d, \ldots, [\mathbf{b}^{(h)}]^i_d\right) = M \cdot ([\tilde{\mathbf{b}}_1]^i_d, \ldots, [\tilde{\mathbf{b}}_n]^i_d)$
3. The parties call $[\boldsymbol{\gamma}]_d \leftarrow \mathcal{F}_{\mathsf{rand}}$.
   The parties call $\mathcal{F}_{\mathsf{coin}}$ to get random coefficients $c_1, \ldots, c_{3h} \leftarrow \mathbb{F}$.
   They compute $[\boldsymbol{\zeta}] = [\boldsymbol{\gamma}]_d + \sum_{i=1}^{n} c_i[\mathbf{a_i}]_d + c_{n+i}[\mathbf{b_i}]_d + c_{2n+i}[\mathbf{r_i}]_d$.
   They open $\boldsymbol{\zeta}$, and verify that it is of degree d. If not, they abort.
4. For $j \in \{1, \ldots, h\}$, P_i computes $[\mathbf{m}^{(j)}]^i_{2d} = [\mathbf{a}^{(j)}]^i_d \cdot [\mathbf{b}^{(j)}]^i_d + [\mathbf{r}^{(j)}]^i_{2d}$, and sends $[\mathbf{m}^{(j)}]^i_{2d}$ to P_0.
5. For $j \in \{1, \ldots, h\}$, P_0 receives $([\mathbf{m}^{(j)}]^1_{2d}, \ldots, [\mathbf{m}^{(j)}]^n_{2d})$. P_0 reconstructs the packed secrets: $\mathbf{m}^{(j)} = \mathsf{reconstruct}(2d, [\mathbf{m}^{(j)}]^1_{2d}, \ldots, [\mathbf{m}^{(j)}]^n_{2d})$. P_0 then calls $\mathsf{share}(d, \mathbf{m}^{(j)})$ and sends the first $d+1$ shares to $P_1, \ldots, P_{d+1}$.
6. For $j \in \{1, \ldots, h\}$, each $P_i \in \{P_1, \ldots, P_{d+1}\}$ receives $[\mathbf{m}^{(j)}]^i_d$ from P_0, and computes
   $[\mathbf{c}^{(j)}]^i_d = [\mathbf{m}^{(j)}]^i_d - [\mathbf{r}^{(j)}]^i_d = [\mathbf{a}^{(j)} \cdot \mathbf{b}^{(j)}]^i_d$.
   P_i calls $\mathsf{aShare}([\mathbf{a}^{(j)}]^i_d)$, $\mathsf{aShare}([\mathbf{b}^{(j)}]^i_d)$, and $\mathsf{aShare}([\mathbf{c}^{(j)}]^i_d)$.
7. For each $j \in \{1, \ldots, h\}$, each $P_k \in \mathsf{Com}$ receives $([[\mathbf{a}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{a}^{(j)}]^n_d]^k_{\mathsf{a}})$, $([[\mathbf{b}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{b}^{(j)}]^n_d]^k_{\mathsf{a}})$, and $([[\mathbf{c}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{c}^{(j)}]^n_d]^k_{\mathsf{a}})$.
   For $i \in \{1, \ldots, \ell\}$, $j \in \{1, \ldots, h\}$, P_k computes:
   $[a^{(i,j)}]^k_{\mathsf{a}} = \mathsf{reconstruct}(d, i, [[\mathbf{a}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{a}^{(j)}]^n_d]^k_{\mathsf{a}})$.
   $[b^{(i,j)}]^k_{\mathsf{a}} = \mathsf{reconstruct}(d, i, [[\mathbf{b}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{b}^{(j)}]^n_d]^k_{\mathsf{a}})$.
   $[c^{(i,j)}]^k_{\mathsf{a}} = \mathsf{reconstruct}(d, i, [[\mathbf{c}^{(j)}]^1_d]^k_{\mathsf{a}}, \ldots, [[\mathbf{c}^{(j)}]^n_d]^k_{\mathsf{a}})$.

Output: Each $P_k \in \mathsf{Com}$ outputs $\left\{[a^{(j)}]^k_{\mathsf{a}}, [b^{(j)}]^k_{\mathsf{a}}, [c^{(j)}]^k_{\mathsf{a}}\right\}_{j=1}^{|h\ell|}$

Fig. 5. Protocol for computing additive shares of $h\ell = O(n^2)$ triples. The shares are delivered to a designated committee, Com. For $|C|$ triples, this protocol must be repeated $|C|/h\ell$ times. $M \in \mathbb{F}^{h \times n}$ is a Van Der Monde matrix.

Prior to simulating the final message in which additive shares are sent to the output committee, Com, $\mathcal{S}$ extracts $\boldsymbol{\delta}$, the value used as an additive attack by $\mathcal{A}$, and submits this to the functionality. There are three places in which the adversary might introduce an additive attack; the simulator extracts three values, $\boldsymbol{\delta_0}, \boldsymbol{\delta_1}, \boldsymbol{\delta_2}$, some of which might be 0, and sums them to recover $\boldsymbol{\delta}$. This is done for each of the h packed shares of triples. For ease of notation, we drop the superscript and only describe the simulation for one of these h values. Recall that P_0 denotes the dealer, and Com is the set of parties receiving output. We let D^+ denote the set $\{P_1, \ldots, P_{d+1}\}$.

If $P_0 \in \mathcal{H}$, $\mathcal{S}$ recovers $\mathbf{m}$ after the malicious parties send shares of $[\mathbf{m}]_{2d}$ to P_0 in Step 4. He then computes $\boldsymbol{\delta_0} = \mathbf{m} - (\mathbf{ab} + \mathbf{r})$. If $P_0 \in \mathcal{A}$, $\mathcal{S}$ sets $\boldsymbol{\delta_0} = 0^l$.

If $(P_0 \in \mathcal{A}) \wedge (D^+ \subset \mathcal{H})$:
$\mathcal{S}$ recovers $\mathbf{m}$ by interpolating the shares sent from P_0 to D^+ in Step 5. He computes $\boldsymbol{\delta_1} = \mathbf{m} - (\mathbf{ab} + \mathbf{r})$. If $P_0 \in \mathcal{H}$, or $D^+ \not\subset \mathcal{H}$, $\mathcal{S}$ sets $\boldsymbol{\delta_1} = 0^l$.

If $D^+ \not\subset \mathcal{H} \wedge \mathsf{Com} \subset \mathcal{H}$:
$\mathcal{S}$ recovers $\widetilde{\mathbf{ab}}$ by summing the shares sent from the malicious parties in D^+ to Com in Step 6 (using the stored values of any any honest parties in D^+ to fill in the gaps) and interpolating the resulting polynomial shares. He computes $\boldsymbol{\delta_2} = \widetilde{\mathbf{ab}} - \mathbf{ab}$. If $D^+ \in \mathcal{H}$, or $\mathsf{Com} \not\subset \mathcal{H}$, $\mathcal{S}$ sets $\boldsymbol{\delta_2} = 0^l$.
$\mathcal{S}$ sets $\boldsymbol{\delta} = \boldsymbol{\delta_0} + \boldsymbol{\delta_1} + \boldsymbol{\delta_2}$.

Finally, $\mathcal{S}$ calls $\mathcal{F}_{\text{triple}}(\boldsymbol{\delta})$, and returns the output from the functionality to $\mathcal{A}$.

We now argue that $\mathcal{S}$ produces a joint distribution on the view of $\mathcal{A}$ and the output of honest parties that is statistically close to $\mathcal{A}$'s view in the real world, together with the honest output in a real execution.

We first argue that if $\mathcal{S}$ does not abort, the $\boldsymbol{\delta}$ value extracted by $\mathcal{S}$ is correct. That is, that it matches the $\boldsymbol{\delta}$ imposed in the real world output under the same adversarial behavior. Because $\mathcal{S}$ does not abort, the values $\mathbf{a}, \mathbf{b}$ and $\mathbf{r}$ that are extracted in the first step are well defined. There are only 3 other messages sent in the protocol: $[\mathbf{m}]_{2d}$ sent to P_0, the response $[\mathbf{m}]_d$ sent to D^+, and additive sharing of $[\mathbf{ab}]_d$ sent from D^+ to Com. In each of these 3 cases we show that the δ values that the adversary is able to introduce are independent of the values of $\mathbf{a}$ or $\mathbf{b}$. We note that this is not true when there is a *linear* attack since the $\boldsymbol{\delta}$ value in that case is allowed to be a linear combination of the packed values. Namely, we prove the following claim.

Claim. If $\mathcal{S}$ does not abort, then for any values $\overline{\mathbf{a}}$, $\overline{\mathbf{b}}$, and $\overline{\boldsymbol{\delta}} \in \mathbb{F}^l$, we have that

$$\Pr[\mathbf{a} = \overline{\mathbf{a}} \wedge \mathbf{b} = \overline{\mathbf{b}} | \boldsymbol{\delta} = \overline{\boldsymbol{\delta}}] = \Pr[\mathbf{a} = \overline{\mathbf{a}} \wedge \mathbf{b} = \overline{\mathbf{b}}]$$

Proof. Since $\mathcal{S}$ aborts whenever the degree of the initial sharing is too high, we know that $[\mathbf{a}]$, $[\mathbf{b}]$, and $[\mathbf{r}]$ are all shared via degree t polynomials, and thus the values $\mathbf{a}, \mathbf{b}, \mathbf{r}$ extracted by $\mathcal{S}$ are well defined. We now analyze all the ways in which a malicious $\mathcal{A}$ can introduce a non-zero $\boldsymbol{\delta}$.

If $P_0 \in \mathcal{H}$, then $\boldsymbol{\delta_0}$ is extracted by P_0 after receiving shares of $[\mathbf{m}]_{2d}$ in Step 4. In this case, $P_i \in \mathcal{A}$ can modify his share $[\mathbf{m}]_d^i$ to $[\mathbf{m}']^i = ([\mathbf{a}]_d^i + \delta_a) \cdot ([\mathbf{b}]_d^i + \delta_b) + [\mathbf{r}]_{2d}^i + \delta_m$ for adversarially chosen δ_a, δ_b, and δ_m, where each of these can be arbitrary functions of the $t < d$ shares received by the malicious parties in Step 1. Rewriting this as $[\mathbf{m}']^i = ([\mathbf{a}]_d^i \cdot [\mathbf{b}]_d^i) + (\delta_a \cdot [\mathbf{b}]_d^i) + (\delta_b \cdot [\mathbf{a}]_d^i) + \delta_m$, we see that P_i can contribute a value $\delta^i = (\delta_a \cdot [\mathbf{b}]_d^i) + (\delta_b \cdot [\mathbf{a}]_d^i) + \delta_m$, which is also a function of at most t shares of $\mathbf{a}$ and $\mathbf{b}$. Thus, the jth value in $\boldsymbol{\delta_0}$, $\boldsymbol{\delta_0^j} = \sum_{P_i \in \mathcal{A}} s_i^j \delta^i$ where the s_i^j are the corresponding Lagrange coefficients. Since $\boldsymbol{\delta_0^j}$ is a linear combination of these δ^i values, it follows that it is also independent of $\mathbf{a}$ and $\mathbf{b}$.

If $(P_0 \in \mathcal{A}) \wedge (D^+ \subset \mathcal{H})$, then $\mathcal{S}$ extracts $\boldsymbol{\delta_1}$ from the values sent to D^+ by the dealer in Step 5. Because the sharing is of degree $d = |D^+| - 1$, whatever P_0 sends in this step is a consistent sharing of some value, $\mathbf{m}'$. This value might be some function of $\mathbf{ab} + \mathbf{r}$, but because the P_0 has no information about $\mathbf{r}$, it follows that δ_1 is independent of $\mathbf{ab}$.

Finally, if $D^+ \not\subset \mathcal{H} \wedge \mathsf{Com} \subset \mathcal{H}$, $\boldsymbol{\delta_2}$ is extracted from the shares sent to Com. Thus, $\boldsymbol{\delta_2^j} = \sum_{P_i \in \mathcal{A}} s_i^j \delta_i$ where δ_i is the change introduce by P_i to his Shamir sharing of $\mathbf{ab}$ (during the process of additively sharing the Shamir share). Since this is again a linear combination of at most t shares of $\mathbf{ab}$, it is independent of $\mathbf{a}$ and $\mathbf{b}$.

We can now complete the proof of Theorem 2. It is easy to see that the shares included in $\mathcal{A}$'s view are independent of the values $\mathbf{a_i}, \mathbf{b_i}, \mathbf{r_i}$ chosen by $\mathcal{S}$. Moreover, by Claim 3, we know that, as long as the degree check does not fail, for any $\mathcal{A}$, the value of $\boldsymbol{\delta}$ is also independent of these values. In the real-world, the honest parties would output these shares while in the ideal-world they output the shares chosen by $\mathcal{F}_{\text{triple}}$. Since the view of $\mathcal{A}$ is independent of these values, both sets of honest outputs are consistent with $\mathcal{A}$'s view showing that the simulation is perfect. Finally, since the degree check succeeds with all but negligible probability, we have that the simulated joint distribution is statistically indistinguishable from that of the real world, proving the theorem.

4 Protocols for Circuit Evaluation

For input sharing and the online phase we use the well-known SPDZ protocol [16]. However, as SPDZ uses authenticated triples, the online protocol first needs to "authenticate" the triples produced by our offline phase. This authentication protocol is very similar to the protocols found in prior work ([16,14,32,3]), and appears in Section 4.1. We note that while the cost of authenticating the triples is asymptotically the same as the cost of using them for circuit evaluation, this does result in approximately a factor of 7 increase in communication in the online phase, when compare with the online phase of SPDZ. We leave reducing this as an interesting open question.

4.1 Authenticating the triples

The previous protocol provides triples that are un-authenticated. We now use these unauthenticated triples to construct *authenticated* triples of the form $(a, b, c,$

$\mathcal{F}_{\mathsf{TripAuth}}$

Inputs: None.

Computation: Sample $\alpha \leftarrow \mathbb{F}$, and a random additive sharing, $[\alpha]$.

For $i \in [m]$:
- Sample $a_i, b_i \leftarrow \mathbb{F}$, and compute $c_i = a_i b_i$.
- Sample random additive sharings, $[a_i], [b_i], [c_i], [\alpha a_i], [\alpha b_i], [\alpha c_i]$

Output: $[\alpha], \left\{[a_i], [b_i], [c_i], [\alpha a_i], [\alpha b_i], [\alpha c_i]\right\}_{i=1}^{m}$

Fig. 6. An ideal functionality for constructing authenticated triples. The functionality is parameterized by an integer m indicating how many authenticated triples should be output.

$\alpha a, \alpha b, \alpha c)$, where $ab = c$, and α, which is unknown to the parties, is used to authenticate all of the triples.

For a committee to generate $m = O(\frac{s|C|}{n})$ authenticated triples, they need to use $8m$ unauthenticated triples. We describe the protocol in the $\mathcal{F}_{\text{triple}}$-hybrid model, allowing the parties on this committee to receive these unauthenticated triples from that functionality. We remind the reader that in practice, the protocol realizing $\mathcal{F}_{\mathsf{triple}}$ is executed in the full network. The remainder of the Π_{TripAuth} protocol is carried out only by the smaller committees and only requires a dishonest majority.

Our construction is almost identical to the one used in SPDZ. The parties first choose an additive secret sharing of a random authentication value, α. They take one triple, $([a], [b], [c])$, and compute $([\alpha a], [\alpha b], [\alpha c])$ using 3 other triples, one for each multiplication (as in the classic result by Beaver [4]).[13] We denote this procedure by Mult in the protocol description.

Because the triples are additively shared and unauthenticated, it is important to note that the adversary can modify the shared value at anytime. To verify that we have a valid authenticated triple after performing these multiplications, we sacrifice one authenticated triple against another to catch any malicious modifications, precisely as in prior work.

The only difference between our construction and that of SPDZ (and the follow-up work) is in the way we instantiate the Mult sub-routine. While we use unauthenticated triples, as just described, SPDZ uses somewhat homomorphic encryption to generate these authenticated triples (prior to the sacrifice step). With that approach, once the parties hold $([a], [b], [c])$, the adversary is unable to modify the result of $\mathsf{Mult}(a, \alpha)$ or $\mathsf{Mult}(b, \alpha)$. However, because they need to refresh the ciphertext encrypting c, the adversary can introduce an additive shift, resulting in $\mathsf{Mult}(c + \delta_c, \alpha)$. In our realization of Mult, the adversary can

[13] For example, to compute $[\alpha a]$ from $[\alpha]$ and $[a]$ using triple (x, y, z), the parties open $a + x$ and $\alpha + y$. Each locally fixes its share by computing $(a + x)[\alpha] + (\alpha + y)[a] - (a + x)(\alpha + y) + [z]$.

Π_{TripAuth}

Inputs: None.
Protocol:

1. The parties call $[\alpha] \leftarrow \mathcal{F}_{\mathsf{rand}}$.
2. The parties call $\mathcal{F}_{\mathrm{triple}}$ to generate $8m$ triples. We denote the first $2m$ of these triples by $\left\{[a_i], [b_i], [c_i], [a'_i], [b'_i], [c'_i]\right\}_{i=1}^{m}$.
3. The parties call $[a\alpha] = \mathsf{Mult}(a, \alpha), [b\alpha] = \mathsf{Mult}(b, \alpha), [c\alpha] = \mathsf{Mult}(c, \alpha)$ and $[a'\alpha] = \mathsf{Mult}(a', \alpha), [b'\alpha] = \mathsf{Mult}(b', \alpha), [c'\alpha] = \mathsf{Mult}(c', \alpha)$
4. The parties call $(r_1, \ldots, r_m) \leftarrow \mathcal{F}_{\mathsf{rand}}$.
5. For $i \in [m]$, the parties
   - compute and open $\bar{a}_i = r_i \cdot [a_i] - [a'_i]$, and $\bar{b}_i = [b_i] - [b'_i]$.
   - The parties compute:

$$[\gamma_i] = r_i[c_i\alpha] - [c'_i\alpha] - \bar{b}_i[a'_i\alpha] - \bar{a}_i[b'_i\alpha] - \bar{a}_i\bar{b}_i[\alpha]$$
$$[\rho_i] = r_i[a_i\alpha] - [a'_i\alpha] - \bar{a}_i[\alpha]$$
$$[\sigma_i] = [b_i\alpha] - [b'_i\alpha] - \bar{b}_i[\alpha]$$

6. The parties call $(\tau_1, \ldots, \tau_{3m}) \leftarrow \mathcal{F}_{\mathsf{rand}}$.
7. The parties compute

$$[\zeta] = \sum_{i=1}^{m} \tau_i[\gamma_i] + \sum_{i=1}^{m} \tau_{m+i}[\rho_i] + \sum_{i=1}^{m} \tau_{2m+i}[\sigma_i]$$

The parties open $[\zeta]$ using a commit-and-reveal. If $\zeta \neq 0$, abort.

Output: Each party outputs its shares of $[\alpha], ([a_i], [b_i], [c_i])$, and $([a_i\alpha], [b_i\alpha], [c_i\alpha])$.

Fig. 7. Protocol for sacrificing some triples in order to authenticate others. The authenticated triples are then used in the online phase. The protocol is in the $\mathcal{F}_{\mathrm{triple}}$-hybrid model (Figure 4), and realizes the $\mathcal{F}_{\mathsf{TripAuth}}$ functionality (Figure 8). The Mult sub-routine is the standard protocol by Beaver [4] for securely computing $[xy]$ from $[x]$ and $[y]$.

introduce this shift on any of the inputs to Mult. For this reason, we provide a complete proof of security for the authentication step. The functionality and protocol realizing this authentication are given in Figures 8 and 9 respectively.

Complexity After receiving the $8m$ unauthenticated triples, each party sends $12m$ field elements in the $6m$ executions to the Mult subroutines, and another $2m$ field elements in the sacrifice step. The rest of the communication can be amortized, as it is independent of m.

It is known that we can further reduce the number of unauthenticated triples needed [31]. In the sacrifice step, instead of using 2 independent triples, we can use $(\alpha a, \alpha b, \alpha c)$ and $(\alpha a', \alpha b, \alpha c')$, where b was used twice. In this case, we could reduce communication per triple in our offline phase by 12.5% (by sending half as many b values), and we can reduce the number of triples needed in the Π_{TripAuth} protocol by 12.5% (because we only multiply b with α and not b'.)[14]

Theorem 3. *The protocol Π_{TripAuth} securely realizes $\mathcal{F}_{\mathsf{TripAuth}}$ in the $\mathcal{F}_{\text{triple}}$-hybrid model, in the presence of an active adversary corrupting all but one party.*

Proof. The proof of Theorem 3 follows from prior work for completeness we provide it in Appendix A.

4.2 Providing Input and MACCheck

$\mathcal{F}_{\mathsf{TripAuth}}$

Setup: Sample and store $\alpha \leftarrow \mathbb{F}$.

Triples:
Inputs: None.
Computation:
For $i \in [m]$:
- Sample $a_i, b_i \leftarrow \mathbb{F}$, and compute $c_i = a_i b_i$.
- Sample random additive sharings, $[a_i], [b_i], [c_i], [\alpha a_i], [\alpha b_i], [\alpha c_i]$

Output: $[\alpha], \big\{[a_i], [b_i], [c_i], [\alpha a_i], [\alpha b_i], [\alpha c_i]\big\}_{i=1}^{m}$

Input Authentication:
Inputs: The input party provides input value $x \in \mathbb{F}$.
Computation:
- sample $[x]$ and $[\alpha x]$

Output: $[x]$ and $[\alpha x]$ are distributed to the parties on the committee.

Fig. 8. An ideal functionality for constructing authenticated triples, and authenticated inputs. The functionality is parameterized by an integer m indicating how many authenticated triples should be output.

For providing input, we note that SPDZ uses authenticated random values $([r], [\alpha r])$ to mask the inputs. We can use the shares of either a or b from an authenticated triple in place of these random values.

Finally, as in SPDZ, prior to reconstructing the output, we perform a MACCheck on all values opened during the protocol. Specifically, as in SPDZ, we do this by opening α to all parties. The parties can then perform a batched MACCheck on all values that were publicly opened during multiplication. For the parties providing input, the committee additionally opens $[\alpha r]$ to the relevant

[14] Note that our offline phase has a computational bottleneck, so reducing the communication cost per triple might not lead to large improvement in runtime, though it still may reduce the dollar cost of communicating. Reducing the number of triples needed will reduce end-to-end runtime.

Π_{TripAuth}

Triples:

Inputs: None.

Protocol:

1. The parties call $[\alpha] \leftarrow \mathcal{F}_{\mathsf{rand}}$.
2. The parties call $\mathcal{F}_{\text{triple}}$ to generate $8m$ triples. We denote the first $2m$ of these triples by $\left\{[a_i], [b_i], [c_i], [a'_i], [b'_i], [c'_i]\right\}_{i=1}^{m}$.
3. The parties call $[a\alpha] = \mathsf{Mult}(a, \alpha), [b\alpha] = \mathsf{Mult}(b, \alpha), [c\alpha] = \mathsf{Mult}(c, \alpha)$ and $[a'\alpha] = \mathsf{Mult}(a', \alpha), [b'\alpha] = \mathsf{Mult}(b', \alpha), [c'\alpha] = \mathsf{Mult}(c', \alpha)$
4. The parties call $(r_1, \ldots, r_m) \leftarrow \mathcal{F}_{\mathsf{rand}}$.
5. For $i \in [m]$, the parties
   - compute and open $\overline{a}_i = r_i \cdot [a_i] - [a'_i]$, and $\overline{b}_i = [b_i] - [b'_i]$.
   - The parties compute:

$$[\gamma_i] = r_i[c_i\alpha] - [c'_i\alpha] - \overline{b}_i[a'_i\alpha] - \overline{a}_i[b'_i\alpha] - \overline{a}_i\overline{b}_i[\alpha]$$
$$[\rho_i] = r_i[a_i\alpha] - [a'_i\alpha] - \overline{a}_i[\alpha]$$
$$[\sigma_i] = [b_i\alpha] - [b'_i\alpha] - \overline{b}_i[\alpha]$$

6. The parties call $(\tau_1, \ldots, \tau_{3m}) \leftarrow \mathcal{F}_{\mathsf{rand}}$.
7. The parties compute

$$[\zeta] = \sum_{i=1}^{m} \tau_i[\gamma_i] + \sum_{i=1}^{m} \tau_{m+i}[\rho_i] + \sum_{i=1}^{m} \tau_{2m+i}[\sigma_i]$$

The parties open $[\zeta]$ using a commit-and-reveal. If $\zeta \neq 0$, abort.

Output: Each party outputs its shares of $[\alpha], ([a_i], [b_i], [c_i])$, and $([a_i\alpha], [b_i\alpha], [c_i\alpha])$.

Input Authentication:

Inputs: The input party provides input value $x \in \mathbb{F}$.

1. The computing committee uses a triple, and opens $(a, b\alpha, c\alpha)$ to party P_i.
2. P_i sends $(x + a, [ab\alpha])$ to the committee.
3. The committee computes $[x] = x + a - [a]$, and $[x\alpha] = (x + a)[\alpha] - [a\alpha])$ for use in the online phase. They also open to P_i: $[ab\alpha] - [c\alpha]$
4. If P_i receives an opening that is non-zero, it aborts.

Fig. 9. Protocol for sacrificing some triples in order to authenticate others. The authenticated triples are then used in the online phase. The protocol is in the $\mathcal{F}_{\text{triple}}$-hybrid model (Figure 4), and realizes the $\mathcal{F}_{\mathsf{TripAuth}}$ functionality (Figure 8). The Mult sub-routine is the standard protocol by Beaver [4] for securely computing $[xy]$ from $[x]$ and $[y]$.

party, allowing them to locally MACCheck the masks r that they used to hide their inputs.

As these protocols, and the corresponding functionalities, are exactly as described in SPDZ and we omit their descriptions here.

5 Optimizing Large-Scale MPC

Having presented the components of our protocol in Sections 3 and 4, we now describe trade-offs that result from various choices of committee sizes. In the process we identify several standard, but critical optimization techniques, and we identify communication and computation bottlenecks. We also provide a more in-depth analysis of the enhanced "folklore" schemes described in Section 1.2, explaining how these compare to our work.

5.1 Protocol Optimizations

We make heavy use of the following two standard optimization techniques. While these are not new, we briefly describe them here for completeness.

Pseudorandom Share Transfer: Our first optimization technique allows us to transfer an additively shared value using very low communication by using a PRG. Suppose a party P wants to additively secret share a value s to a committee Com containing S parties. P pre-shares a PRG seed s_i with each party P_i in Com. Then, for all but the last party in Com, both P and $P_i \in \mathsf{Com}$ compute a share as $[s]_i = G(s_i)$, and P computes $s - \sum_{i=1}^{s-1}[s]_i$ and sends this value to P_S as his share. In this way, P can share a secret with a committee of any size while communicating only one field element. If we start with an additive secret sharing of s across a committee Com_1, using the same technique, we can transfer this shared value to another committee Com_2 (of arbitrary size) by having each party in Com_1 send one field element. We observe that this technique does not work for Shamir-shared values. Instead, the natural parallel (e.g., as described by [20,34]) can only save, approximately, a factor of 2 in the communication.

Amortization: A second optimization that we use extensively in all of our protocols is amortization. Specifically, we rotate the roles in our protocol to ensure that the communication and computation loads are split equally across all parties. For example, we rotate the party serving as the dealer (P_0) in Steps 4-5 of Figure 5, and we rotate the party receiving the real share in the pseudorandom share transfer described above.

5.2 Protocol Bottlenecks

To simplify the analysis of both our and related protocols, we decompose our protocol into phases, allowing us to analyze the bottlenecks of each phase separately.

1. The *Vandermonde* phase corresponds to Steps 1-2 of Figure 5 in which parties generate doubly-shared (packed) random values. This phase is always run by all n parties and produces $O(|C|/n)$ packed secrets. It requires each

party to send (and receive) $O(n)$ shares (one to each party) to produce $O(n)$ packed secrets, resulting in $O(|C|/n)$ bottleneck communication.
For computation, for every n packed secrets, each party must produce its own packed share, requiring $O(n \log n)$ field operations using an FFT, and must perform multiplication by the Vandermonde matrix M, which can be done by an IFFT in the same time. Thus, we have a bottleneck computation of $O(|C|/n^2 \cdot n \log n) = O(\log n|C|/n)$ field multiplications.
2. The *Triple Gen.* phase corresponds to Steps 3-5 in Figure 5. This phase is again run by all n parties, and we rotate the dealer to ensure that each party plays the role of the dealer in a $1/n$ fraction of the $O(|C|/n)$ executions. Each time a party is the dealer, it receives and sends $O(n)$ shares for each packed triple and also performs an iFFT and an FFT to produce the new sharing. Thus, the bottleneck communication is $O(|C|/n^2 \cdot n) = O(|C|/n)$ field elements and the bottleneck computation is $O(\log n|C|/n)$ field operations. We note that Step 3 (degree check) is only performed once per $O(n^2)$ triples, and thus will not be the bottleneck.
3. The *Transfer* phase corresponds to Step 6 in Figure 5. In this phase all n parties sub-share their (Shamir-shared) packed triples to the committee Com. The cost of this step varies a good deal in our different protocol variants, so we defer the discussion.
4. The *Unpack* phase corresponds to Step 7 in Figure 5. In this phase the parties in Com perform local computation to unpack the received triples into additively shared triples. This requires no communication, but requires an IFFT and thus $O(n \log n)$ field operations for every n shares.
5. The *Online* phase corresponds to the online protocols described in Section 4.

5.3 Protocol Variants

We now describe several different protocol variants for realizing secure committee-based MPC. For each of these variants, we analyze its asymptotic communication and computation complexity. Additionally, for each of these protocol variants, we analyze analogous committee-based protocols built on top of the existing protocols [20,10,16], as described briefly in Section 1.2, to provide an apples-to-apples comparison to our work. For ease of reference, we duplicate the tables that appeared in Section 1.2 here.

Single Committee: The first protocol variant we consider is one that directly follows the protocol specified in Figure 5. That is, we use a single committee Com for performing both the *unpack* and *online* phases. We consider both a committee of size $O(s)$, and the case where Com is the set of all n parties. In the former case, the communication bottleneck (for the offline portion) is the cost for the committee parties to receive the packed secrets. Since we have $O(|C|/n)$ packed secrets, with n parties holding a share of each, there are a total of $O(|C|)$ shares that need to be transmitted to Com. Using pseudorandom shares and amortizing receiving cost, this requires each party in Com to receive $O(|C|/s)$ field elements. If, on the other hand, we let $|\mathsf{Com}| = n$, then we can split the task of receiving

Single Online, Single Unpack				
	Ours	$t < n/3$ [20]	$t < n/2$ [10]	$t < n$ [16]
Comm	$O(\|C\|/n)$	$O(\|C\|)$	$O(\|C\|)$	$O(s\|C\|)$
Comp	$O(\log n\|C\|)$	$O(\log n\|C\|)$	$O(\log s\|C\|)$	$O(s\|C\|)$

Fig. 10. Asymptotic complexities for the offline phase of four protocols. In our variant, and in that of [20], all parties participate throughout. In the other two protocols, the state-of-the-art is run by a small committee: with an honest majority in column 4, and a malicious majority in column 5. All values measure bottleneck costs, rather than total costs. The online communication and computation cost for all four protocols is $O(C)$, and is not included.

these shares across all n parties, resulting in $O(|C|/n)$ bottleneck communication. We note that in this case, this also matches the bottleneck communication in the Vandermonde and Triple gen. phases. In both of these options, the bottleneck communication of the online phase is $O(|C|)$: given multiplication triples, evaluating a multiplication gate requires $O(1)$ communication by each party in Com. In both cases, the computation bottleneck arises from the unpacking step, where the parties in Com need to unpack $O(|C|/n)$ packed shares, each requiring $O(n \log n)$ field operations (for FFT), resulting in bottleneck computation of $O(\log n|C|)$ field operations.

It is worth exploring the trade-off here between bottleneck complexity and total complexity. We have added Figure 11 to help do so. Because receiving shares is a bottleneck, we lower the bottleneck complexity in the offline phase if we have a bigger receiving committee. Put another way, using pseudorandom shares, sending to a larger committee does not increase the total communication, but it does distribute the cost of receiving that same data. On the other hand, when we look at the cost of the online phase, computing in a smaller committee does reduce the total communication. The question of which is preferable depends on the application, and possibly on the incentive of the participants and the protocol administrator. In some rough sense, a lower bottleneck complexity implies a shorter run-time, while a lower total complexity implies a cheaper protocol, financially. We note that the same comparison can be made for the protocol using a single malicious majority committee in Figure 12, and for the two protocols with malicious majority committees in Figure 13. We avoid the redundancy and do not include the data in our figures.

For comparison purposes, we consider analogous protocol variants built from existing protocols. First, consider the protocol of Furukawa and Lindell [20] which is secure for $t < n/3$. Since we only have $t < n/3$, this protocol must use all n parties for the entire computation resulting in an $O(|C|)$ bottleneck communication and $O(|C| \log n)$ computation (from the Vandermonde step). It is easy to see that we achieve significant asymptotic savings in offline communication while matching the online communication and computation bottlenecks. Second, consider the protocol of Genkin et al. [23]. This protocol also requires $t < n/3$ and thus must use all n parties. However, this circuit works by convert-

Single Online, Single Unpack				
	Offline		Online	
	$\lvert\mathsf{Com}\rvert = n$	$\lvert\mathsf{Com}\rvert = O(s)$	$\lvert\mathsf{Com}\rvert = n$	$\lvert\mathsf{Com}\rvert = O(s)$
Bottleneck Comm	$O(\lvert C\rvert/n)$	$O(\lvert C\rvert/s)$	$O(\lvert C\rvert)$	$O(\lvert C\rvert)$
Total Comm	$O(\lvert C\rvert)$	$O(\lvert C\rvert)$	$O(n\lvert C\rvert)$	$O(s\lvert C\rvert)$
Bottleneck Comp	$O(\log n\lvert C\rvert)$	$O(\log n\lvert C\rvert)$	$O(\lvert C\rvert)$	$O(\lvert C\rvert)$
Total Comp	$O(n\log n\lvert C\rvert)$	$O(s\log n\lvert C\rvert)$	$O(n\lvert C\rvert)$	$O(s\lvert C\rvert)$

Fig. 11. We compare two variants of our protocol that was described in Figure 12. In the first column, the packed triples are re-shared with the full network, and unpacked by all n parties. In the second column, the they are re-shared with a small, malicious majority committee, and unpacked there.

ing the circuit C into a SIMD circuit at the cost of a $O(\log|C|)$ factor increase in circuit size. This results in an offline and online bottleneck communication of $O(|C|\log|C|/n)$, a $O(\log|C|)$ overhead on our protocols.

Next, we build a protocol using the honest majority (i.e., $t < n/2$) protocol from Chida et al. [10]. For this protocol we select random Com of size $O(s)$ that guarantees honest majority within the committee, and use this committee for the entire computation. Since the Chida et al. protocol cannot take advantage of pseudorandom share transfer, there is no benefit to using a committee of size n. This results in a bottleneck communication of $O(|C|)$ for both the offline (i.e, Vandermonde) and online phases and $O(|C|\log s)$ computation (from Vandermonde multiplication). We again significantly improve in the offline communication. However, we are now slightly worse in bottleneck computation.

Finally, we consider a protocol variant using the malicious majority protocol of Damgård et al. [16]. For this setting, we choose a committee of size $O(s)$ to guarantee at least one honest party and run both the offline and online phase inside this committee. Here, the higher complexity of triple generation becomes the bottleneck resulting in $O(s|C|)$ bottleneck communication and computation for the triple generation. This communication is worse than our protocol by a factor of $O(n)$. The online communication of $O(|C|)$ matches our protocol.

Single Online, Distributed Unpack			
	Ours	$t < n/2$	$t < n$
Comm	$O(s\lvert C\rvert/n)$	$O(s\lvert C\rvert)$	$O(s^2\lvert C\rvert/n)$
Comp	$O(s\log n\lvert C\rvert/n)$	$O(s\log s\lvert C\rvert/n)$	$O(s^2\lvert C\rvert/n)$

Fig. 12. In our protocol variant, triples are unpacked in many parallel committees, and then transferred back to the full set of size n. Costs are measured through the transfer step, and exclude the cost of the online phase. In the column labeled $t < n/2$, many parallel committees, each with an honest majority, prepare double-sharings that will be used by a single online committee of size $O(s)$. In the column labeled $t < n$, many parallel committees, each with at least one honest participant, generate multiplication triples that will be used by the full network of size n. The online phase still requires $O(|C|)$ for all three protocols, and is not included in the Table.

Distributed triple generation: We note that the bottleneck computation cost of our protocol grows logarithmically in the number of parties. For our protocol, this bottleneck comes from the cost of the unpacking phase. Thus, our next protocol variant addresses this problem by distributing the unpacking. Specifically, we elect many *unpacking* committees, each of size $O(s)$ to guarantee at least one honest party per committee. These committees split the triples to unpack, and then transfer the unpacked triples to the online committee (which can again be of size $O(s)$ or include all n parties). Now each unpacking committee only needs to unpack a $O(s/n)$ fraction of the triples, requiring $O(|C|/n \cdot s/n \cdot n \log n) = O(s \log n|C|/n)$ field operations. However, we now need to communicate unpacked triples to the online committee. Since there are $O(|C|)$ unpacked triples held by committees that are each of size $O(s)$, we need to receive a total of $O(s|C|)$ shares. If we use a committee of size $O(s)$, this requires $O(|C|)$ bottleneck communication, but if we use an online committee of size $O(n)$, this only requires $O(s|C|/n)$ communication. We note, however, that the total communication of the protocol (across all parties) does not decrease, and in fact, increases for the online computation. However, we still believe that presenting the bottleneck complexity is the correct metric here as it measures the end-to-end computation time for running the protocol, whereas the total (or average) communication is measuring the total monetary cost of running the protocol.

We now describe equivalent protocol variants using the protocols of Chida et al. [10] and Damgård et al. [16]. We do not provide further comparison to Furukawa and Lindell [20] or Genkin et al. [23] as their protocols require $t < n/3$ and thus cannot be used with smaller committees. For both of these base protocols, the equivalent protocol variant is to elect many "offline" committees that generate double-shared values (in the case of Chida et al.) or triples (in the case of Damgård et al.) and then transmit this material to a single online committee. For Chida et al., since this protocol cannot take advantage of the pseudorandom share transfer, the cost of transferring the offline material forms the bottleneck of $O(s|C|)$ field elements, a factor of $O(n)$ worse than our best deployment. However, the ability to use smaller committees, each generating a $O(s/n)$ fraction of the pre-processing material, results in $O(s \log s|C|/n)$ field operations bottleneck computation. While this computation is asymptotically better than what we achieve, we note that, concretely, the two are quite similar. Specifically, the $O(s)$ size committee necessary to guarantee honest majority is much larger than the committee needed to guarantee at least one honest party. Concretely, for 2^{-40} security, a committee of size 25 suffices for dishonest majority, but a committee of size 430 is needed to guarantee honest majority. In comparing computation between these solutions, we need to compare $O(s \log n)$ for dishonest majority to $O(s \log s)$ for the honest majority setting. For $n < 2^{150}$, $25 \log n < 430 \log 430$. Taking other constants into account, our protocol requires roughly 2X more computation.

The equivalent protocol variant using Damgård et al. chooses many triple generation committees of size $O(s)$ and has them all transfer the produced triples to the online committee. Since there are $O(n/s)$ such offline committees, each one is tasked with generating $O(s|C|/n)$ triples for bottleneck communication of $O(s^2|C|/n)$, which is $O(s)$ times worse than what is achieved by our protocol.

Distributed Online						
	Ours		$t < n/2$		$t < n$	
	Offline	Online	Offline	Online	Offline	Online
Comm	$O(\|C\|/n)$	$O(s\|C\|/n)$	$O(s\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s\|C\|/n)$
Comp	$O(s \log n\|C\|/n)$	$O(s\|C\|/n)$	$O(s \log s\|C\|/n)$	$O(s\|C\|/n)$	$O(s^2\|C\|/n)$	$O(s\|C\|/n)$

Fig. 13. Here we distribute the work done in the online phase, assigning $O(s|C|/n)$ gates to each of the $O(n/s)$ committees. In all three protocols, the material generated during pre-processing remains with the same committee for the online phase. The state of the online phase is transfered from one committee to the next.

Distributed online computation: Finally, we note that the online cost of all our protocols considered so far grows linearly with $|C|$, as parties in the online committee must send $O(1)$ bits for each gate. To improve on this, we design a protocol that also distributed the online computation. Specifically, we elect many *online* committees, each of size $O(s)$ to guarantee at least one honest party per committee. These committees split the gates of the circuit to evaluate, with each committees responsible for an $O(s/n)$ fraction of the gates. Of course, the intermediate gate outputs must be communicated to the committee responsible for the next gate. Using pseudorandom share transfer, this can be done with each party sending $O(1)$ field elements per gate. Thus, each party needs $O(s|C|/n)$ communication for computing the gates it is responsible for and the same amount of communication to transfer the results of these gates. We note that this deployment also reduces the communication of the offline phase. The reason for this is that we no longer need to transfer unpacked triples to a single online committee, instead each unpacking committee serves as an online committee using the triples it unpacks. Thus, the cost of transfer in the offline phase disappears, and with triple generation and Vandermonde steps as the new bottleneck, we get $O(|C|/n)$ bottleneck communication. Since the computation in the offline phase is unchanged, this remains $O(s \log n|C|/n)$.

For a comparison, we consider similar protocol variants built using the protocols of Chida et al. [10] and Damgård et al. [16]. While we can similarly partition the online phase in both these protocols, the advantages of doing so for Chida et al. are limited. Since their $t < n/2$ protocol cannot take advantage of pseudorandom share transfer, the cost to transmit the state between gates of the circuit dominates, resulting in online communication of $O(s^2|C|/n)$, a factor of $O(s)$ worse than for our protocol. The Damgård et al. protocol, on the other hand, can benefit from pseudorandom share transfer, and can thus distribute their online phase at the same communication cost as our protocol. However, in their offline phase, the cost of triple generation becomes the bottleneck requiring communication of $O(s^2|C|/n)$, which is $O(s^2)$ times worse than our offline cost.

	1 mbps		**10 mbps**		**100 mbps**		**1000 mbps**	
n	**Ours**	**DN**	**Ours**	**DN**	**Ours**	**DN**	**Ours**	**DN**
1024	8898.0	384380.2	1366.2	39286.2	613.0	4776.8	539.6	1325.9
4096	2248.3	85541.2	369.6	8769.4	181.7	1092.2	162.9	324.5
16384	571.0	20274.3	101.6	2086.5	54.7	267.7	50.0	85.8
65536	144.9	5071.6	27.5	524.3	15.8	69.6	14.6	24.1
262144	36.8	1266.5	7.5	131.5	4.6	18.0	4.3	6.7
1048576	9.4	316.6	2.0	33.1	1.3	4.8	1.2	1.9
Ratio	33.8 - 43.2		16.2 - 28.8		3.6 - 7.8		1.6 - 2.5	
Ratio Per Gate	9.7 - 12.3		4.6 - 8.2		1 - 2.2		.5 - .7	

Fig. 14. Ours vs. Damgård-Nielsen Pre-Processing Time. This describes the time, in milliseconds, for n parties to generate one million (unauthenticated) triples (for our protocol) or doubly-shared random values (for DN) as used by Chida et al. [10]. For both we use the multiple unpacking committees protocol. Times are given for four different network bandwidths from 1mbps to 1000mbps. The ratio is a range of ratios between DN and our protocol. In the final row, we consider the fact that the online phase of Chida et al. requires only 2 unauthenticated triples per gate, whereas our online phase requires 7 unauthenticated triples per gate.

The computational load for the distributed online protocol is the same as in the distributed unpacking protocol regardless of which of the three protocols is used as the building block. Thus, in the distributed online variant, our protocol is strictly better than the variants based on either existing protocol, achieving a factor of at least $O(s)$ improvement in communication in both the online and offline phases for Chida et al., and in the offline phase for Damgård et al.

Triples as a service: A related protocol variant that we wish to mention briefly is that of triples as a service. Here, our protocol would be used to produce triples, distributed across multiple unpacking committees, for use by external parties for their online MPC computation. In this case, we do not need to pay the cost for transferring the triples to the clients or for the online phase. Thus, looking at only the offline costs for our optimal deployment (the distributed online one), we see that for this variant we save a factor of $O(s)$ and $O(s^2)$ in communication costs over Chida et al. and Damgård et al. respectively.

6 Concrete Performance Estimation

In Section 5 we analyzed the asymptotic performance of our protocols and compared them to committee-based protocols. Here we look at how these protocols might perform in practice, taking into account the constants hidden in the big-O notation, and the incomparable nature of communication and computation. To better understand the concrete performance comparison, we built a prototype to estimate computation and communication time for two of the protocol variants previously described: our own protocol using distributed unpacking and distributed online evaluation, and the comparable protocol that uses parallel honest majority committees for pre-processing (labeled $t < n/2$ in Figure 13).

In both cases, we only measure the performance of the offline phase, as we view this as our main contribution. Additionally, the offline phase of the distributed online protocol is precisely what is needed for the triples-as-a-service application.

We also do not compare to the cost of parallel triple generation using malicious majority committees. Asymptotically, this protocol is worse than the variant using honest majority committees, in both communication and computation (Table 13). When considering the concrete costs reported in Overdrive [32], it seems to compare less favorably than the honest majority protocol that we have chosen for comparison. Also important is that the protocol we implemented looks similar to our own (since both rely on DN). Nevertheless, a concrete comparison with malicious majority committees would be interesting to add in the future.

6.1 Measurement Details

To estimate the performance of these protocols, we use different techniques for computation and communication. For computation, we implement and run a single party doing the full computation of our protocol to give an accurate estimate of the bottleneck complexity. This is done using the implementation of FFT from the libiop library [1]. All experiments were performed on a machine with a dual core i7 cpu at 2.80GHz.

To estimate the cost of communication, we took a somewhat different approach. Since we do not have access to millions of compute nodes, instead of building a full, networked test of our protocol, we precisely calculated the necessary communication at each step, and estimated the communication time as a function of the network bandwidth and latency. For this evaluation, we varied the available bandwidth between 1 mbps, 10 mbps, 100 mbps, and 1000 mbps.

6.2 Results

The results of our empirical evaluation are given in Figure 14 where we report the time (in milliseconds) needed to produce one million (unauthenticated) triples. Recall that our offline protocol generates unauthenticated triples, and we need 7 such triples for each gate in the online phase. In contrast, protocols using honest majority in the online phase require only 2 unauthenticated triples per gate. In the row labeled “Ratio Per Gate” we provide estimates with this distinction in mind. In the remainder of the Table, we consider the costs of generating unauthenticated triples. At the top end of our performance, with approximately $1,000,000$ parties on a 1000 mbps network, we can generate one million triples in only 1.2 milliseconds. But, our best performance improvement over honest majority committees is on lower bandwidth networks, where we can outperform their pre-processing protocol by as much as 43.2X, and 12.3X when considering authenticated triples. This is due to the fact that our biggest improvement is in communication, at a slight cost in computation. We note that when considering deployment of MPC across tens or hundreds of thousands of parties, it is quite unlikely that all parties will have access to a high-speed (e.g., 1000 mbps) network connection. For example, 4G LTE offers roughly 10 mbps. For synchronous protocols such as ours, the bandwidth and latency of the slowest party becomes that of all parties. Thus, we believe that our results for lower network speeds more closely represent the use-cases we envision.

References

1. libiop. `https://github.com/scipr-lab/libiop`.
2. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.
3. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.
4. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.
5. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.
6. Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 356–376. Springer, Heidelberg, March 2013.
7. Elette Boyle, Abhishek Jain, Manoj Prabhakaran, and Ching-Hua Yu. The bottleneck complexity of secure multiparty computation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl, July 2018.
8. Gabriel Bracha. An o(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.
9. Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In Tim Roughgarden, editor, *ITCS 2015*, pages 153–162. ACM, January 2015.
10. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 34–64. Springer, Heidelberg, August 2018.
11. Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 5:1–5:10, 2015.
12. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006.

13. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Heidelberg, May / June 2010.
14. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.
15. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.
16. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.
17. Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, pages 242–256, 2014.
18. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.
19. Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015.
20. Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1557–1571. ACM Press, November 2019.
21. Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 420–446. Springer, Heidelberg, August 2017.
22. Daniel Genkin. Secure computation in hostile environments (phd thesis). 2016.
23. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 721–741. Springer, Heidelberg, August 2015.
24. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.
25. Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. LevioSA: Lightweight secure arithmetic computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 327–344. ACM Press, November 2019.
26. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT).

In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 86–117. Springer, Heidelberg, December 2018.

27. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2018.
28. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.
29. Mercy O. Jaiyeola, Kyle Patron, Jared Saia, Maxwell Young, and Qian M. Zhou. Good things come in loglog(n)-sized packages: Robustness with small quorums. *CoRR*, abs/1705.10387, 2017.
30. Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.
31. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.
32. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.
33. Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 259–276. ACM Press, October / November 2017.
34. Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority MPC by batchwise multiplication verification. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 321–339. Springer, Heidelberg, July 2018.
35. Peter Scholl, Nigel P. Smart, and Tim Wood. When it's all just too much: Outsourcing MPC-preprocessing. In Máire O'Neill, editor, *16th IMA International Conference on Cryptography and Coding*, volume 10655 of *LNCS*, pages 77–99. Springer, Heidelberg, December 2017.
36. Ryan Wails, Aaron Johnson, Daniel Starin, Arkady Yerukhimovich, and S. Dov Gordon. Stormy: Statistics in tor by measuring securely. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 615–632. ACM Press, November 2019.

37. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017.
38. Mahdi Zamani, Mahnush Movahedi, and Jared Saia. Millions of millionaires: Multiparty computation in large networks. Cryptology ePrint Archive, Report 2014/149, 2014. `http://eprint.iacr.org/2014/149`.
39. C. Zheng, Q. Tang, Q. Lu, J. Li, Z. Zhou, and Q. Liu. Janus: A user-level tcp stack for processing 40 million concurrent tcp connections. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

A Proof of Theorem 3

Theorem 3. *The protocol Π_{TripAuth} securely realizes $\mathcal{F}_{\mathsf{TripAuth}}$ in the $\mathcal{F}_{\text{triple}}$-hybrid model, in the presence of an active adversary corrupting all but one party.*

Proof. We first provide the description of a simulator $\mathcal{S}$. $\mathcal{S}$ starts by querying $\mathcal{F}_{\mathsf{TripAuth}}$, and receives shares of $[\alpha], [a_i], [b_i], [c_i], [a_i\alpha], [b_i\alpha], [c_i\alpha]$. Then,

1. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{rand}}$ using $[\alpha]$. It generates and stores random shares for the honest parties.
2. $\mathcal{S}$ simulates $\mathcal{F}_{\text{triple}}$ by running the code of the functionality, and using $[a_i]$, $[b_i], [c_i]$ for the adversary's output. It also simulates and records the honest output. If $\mathcal{A}$ provides a $\boldsymbol{\delta}$ to $\mathcal{F}_{\text{triple}}$, $\mathcal{S}$ reflects that value in the adversary's output by modifying the shares of c_i.
3. $\mathcal{S}$ simulates Mult by running the code of the functionality, and using $[\alpha], [a_i\alpha]$, $[b_i\alpha], [c_i\alpha]$ for the adversary's output. $\mathcal{S}$ also simulates and records the honest output. If $\mathcal{A}$ modifies (the sum of) its shares of $[\alpha], [a_i], [b_i], [c_i]$ when providing input, then $\mathcal{S}$ reflects that change in the adversary's output shares.
4. $\mathcal{S}$ completes the simulation by running the code of the honest parties, and the functionality $\mathcal{F}_{\mathsf{rand}}$.

We claim that the joint distributions over the adversarial view and the honest output, are statistically close in the real and ideal worlds. First, we make a simple observation: since the simulator runs the protocol honestly, the marginal distributions defined by the views of the adversary in each world are identical. Furthermore, the simulator aborts if and only if the honest parties would abort in the real world. We therefore only consider executions that do not end in abort. In such cases, note that in the ideal world the joint distribution always constitutes a "good authenticated triple", $(a, b, c, \tilde{a}\alpha, \tilde{b}\alpha, \tilde{c}\alpha)$, where the following two properties hold. 1) $\tilde{a}\alpha\cdot\tilde{b}\alpha = \tilde{c}\alpha^2$, and 2) $\tilde{a}$ and $\tilde{b}$ are uniformly and independently chosen from $\mathbb{F}$.[15]

[15] Note that we do not insist that $a = \tilde{a}, b = \tilde{b}, c = \tilde{c}$. There is no need to insist on this, because the adversary can change his shares of (a, b, c) at anytime, including during the online computation. This is true for all schemes using authenticated triples based on additive secret sharing.

We now argue that, assuming the protocol completes without abort, the same properties hold in the real world, except with negligible probability. Since the adversary can produce an additive change to the inputs to Mult by modifying his own shares, we denote such changes by δ with subscripts to indicate which input is modified. We denote the outputs of Mult in Step 2 by:

$$
\begin{aligned}
[\tilde{a}\alpha] &= [(a+\delta_a)(\alpha+\delta_{\alpha,a})],\\
[\tilde{b}\alpha] &= [(b+\delta_b)(\alpha+\delta_{\alpha,b})],\\
[\tilde{c}\alpha] &= [(c+\delta_c)(\alpha+\delta_{\alpha,c})],\\
[\tilde{a}'\alpha] &= [(a'+\delta'_a)(\alpha+\delta'_{\alpha,a})],\\
[\tilde{b}'\alpha] &= [(b'+\delta_{b'})(\alpha+\delta_{\alpha,b'})],\\
[\tilde{c}'\alpha] &= [(c'+\delta'_c)(\alpha+\delta'_{\alpha,c})]
\end{aligned}
$$

noting that, regardless of the values of δ chosen by the adversary, there exist some $\tilde{a}, \tilde{b}, \tilde{c}, \tilde{a}', \tilde{b}'$ and $\tilde{c}'$ for which the above equalities hold. Our proof follows from the following sequence of claims.

Claim. If $\tilde{a}\tilde{b} \neq \tilde{c}$, or $\tilde{a}'\tilde{b}' \neq \tilde{c}'$, then $\Pr[\gamma = 0 \mid] \leq \frac{2}{|\mathbb{F}|}$.

Proof. Note

$$
\begin{aligned}
[\gamma] &= r[\tilde{c}\alpha] - [\tilde{c}'\alpha] - \overline{b}[\tilde{a}'\alpha] - \overline{a}[\tilde{b}'\alpha] - \overline{a}\overline{b}[\alpha]\\
&= \left[\alpha\big(r\tilde{c} - \tilde{c}' + \tilde{a}'\tilde{b}' - r\tilde{a}\tilde{b}\big)\right]\\
&= \left[\alpha\big(r\big(\tilde{c} - \tilde{a}\tilde{b}\big) - (\tilde{c}' - \tilde{a}'\tilde{b}')\big)\right]
\end{aligned}
$$

So, if $\gamma = 0$, we have $\alpha = 0$, or $r = \frac{(\tilde{c}' - \tilde{a}'\tilde{b}')}{(\tilde{c} - \tilde{a}\tilde{b})}$. Since the adversary chose $(\tilde{a}, \tilde{b}, \tilde{c}, \tilde{a}', \tilde{b}', \tilde{c}')$ before r was chosen, the probability that the latter relationship holds is $|\mathbb{F}|^{-1}$. The claim follows from a union bound on these two events.

Claim. The real world distribution over $(\tilde{a}, \tilde{b})$ is statistically close to uniform.

Proof. Note that if $\delta_a = -a$, or if $\delta_{\alpha,a} = -\alpha$, then $\tilde{a} = 0$. The probability of this occurring is at most $\frac{2}{|\mathbb{F}|}$. (The same argument holds for $\tilde{b}$ and $\tilde{c}$.) However, conditioned on $\tilde{a} \neq 0$, $\tilde{a}$ is uniformly distributed over the remaining values of $\mathbb{F}$. It follows that the statistical distance from uniform is $O(\frac{1}{|F|})$.