Generalized Pseudorandom Secret Sharing and Efficient Straggler-Resilient Secure Computation

Fabrice Benhamouda[1], Elette Boyle[2], Niv Gilboa[3], Shai Halevi[1], Yuval Ishai[4], and Ariel Nof[4]

[1] Algorand Foundation.
fabrice.benhamouda@normalesup.org, shaih@alum.mit.edu
[2] IDC Herzliya, Israel.
eboyle@alum.mit.edu
[3] Ben-Gurion University, Israel.
gilboan@bgu.ac.il
[4] Technion, Israel.
{yuvali,ariel.nof}@cs.technion.ac.il

Abstract. Secure multiparty computation (MPC) enables n parties, of which up to t may be corrupted, to perform joint computations on their private inputs while revealing only the outputs. Optimizing the asymptotic and concrete costs of MPC protocols has become an important line of research. Much of this research focuses on the setting of an honest majority, where $n \geq 2t+1$, which gives rise to concretely efficient protocols that are either information-theoretic or make a black-box use of symmetric cryptography. Efficiency can be further improved in the case of a *strong* honest majority, where $n > 2t+1$.

Motivated by the goal of minimizing the communication and latency costs of MPC with a strong honest majority, we make two related contributions.

- **Generalized pseudorandom secret sharing (PRSS).** Linear correlations serve as an important resource for MPC protocols and beyond. PRSS enables secure generation of many pseudorandom instances of such correlations without interaction, given replicated seeds of a pseudorandom function. We extend the PRSS technique of Cramer et al. (TCC 2005) for sharing degree-d polynomials to new constructions leveraging a particular class of combinatorial designs. Our constructions yield a dramatic efficiency improvement when the degree d is higher than the security threshold t, not only for standard degree-d correlations but also for several useful generalizations. In particular, correlations for locally converting between slot configurations in "share packing" enable us to avoid the concrete overhead of prior works.
- **Cheap straggler resilience.** In reality, communication is not fully synchronous: protocol executions suffer from variance in communication delays and occasional node or message-delivery failures. We explore the benefits of PRSS-based MPC with a strong honest majority toward robustness against such failures, in turn yielding improved latency delays. In doing so we develop a novel technique for defending against a subtle "double-dipping" attack, which applies to the best existing protocols, with almost no extra cost in communication or rounds.

Combining the above tools requires further work, including new methods for batch verification via distributed zero-knowledge proofs (Boneh et al., CRYPTO 2019) that apply to packed secret sharing. Overall, our work demonstrates new advantages of the strong honest majority setting, and introduces new tools—in particular, generalized PRSS—that we believe will be of independent use within other cryptographic applications.

1 Introduction

Protocols for secure multiparty computation (MPC) [52, 30, 5, 16] enable a set of parties with private inputs to compute a joint function of their inputs while revealing nothing but the output. MPC provides a general-purpose tool for distributed computation on sensitive data, as well as for eliminating single points of failure. As a result, a major research effort focused on improving the asymptotic and concrete efficiency of MPC.

Efficient honest-majority MPC. The most practical MPC protocols rely on an *honest majority* assumption, namely security is guaranteed as long as $t < n/2$ out of the n parties are corrupted, and provide "security with abort" in the presence of malicious parties. Such protocols can be either information-theoretic, or alternatively achieve better communication cost by making a black-box use of a pseudorandom function. The latter is mainly useful for non-interactive generation of pseudorandom shared secrets via a pseudorandom secret sharing (PRSS) technique [28, 18]. Moreover, the most efficient protocols in this setting follow the blueprint of Damgård and Nielsen (DN) [22], where each layer of a circuit is evaluated by having a designated "leader" party send messages to all other parties and receive a message from each party in return.

In almost all of this line of research, one assumes the weakest honest majority assumption of $n = 2t+1$ parties. However, assuming that up to half of the parties can be corrupted may sometimes be overly *pessimistic*, and small relaxations of corruption threshold can be highly preferred in favor of boosting performance. On the other hand, existing honest-majority protocols are also overly *optimistic* in that they assume all messages arrive on time and are not robust to transient delays or failures. We will revisit this issue later.

The potential for savings in the "strong honest majority" regime of $n > 2t + 1$ has been asserted within the context of *asymptotic* efficiency [24, 19, 21, 20, 4, 27, 36]. In a sense, existing MPC protocols for $n = 2t + 1$ parties are analogous to using a repetition code, which increases the total cost by a factor of n, whereas the latter protocols are analogous to asymptotically good codes that provide a constant or near-constant amortized asymptotic overhead. However, the techniques in these theory-oriented works incur large concrete overheads placing them quite far from practical efficiency, and their asymptotic efficiency benefits kick in only for large computations.

In the context of *concretely* efficient MPC, the potential gains of a strong honest majority remain relatively untapped—both in the sense that asymptotic benefits of prior works do not currently translate to concrete wins, and that potential for concrete gains outside the standard theoretical models or (asymptotic) goals have not been well explored. One exception to this is a recent line of works leveraging a larger number of honest parties for the purpose of closing the efficiency gap between security against *malicious* (or active) adversaries and security against *semi-honest* (or passive) adversaries [31, 26]. However, recent works [8, 11, 37, 12] have successfully closed this gap even given a minimal honest majority $n = 2t + 1$, in which case this advantage no longer applies.

In this work, we initiate a deeper study of *concretely efficient* MPC with *strong honest majority* $n > 2t+1$. We focus on developing general-purpose primitives and techniques to alleviate the concrete costs of existing theory-oriented solutions, as well as exploring new directions for improved latency in realistic networks. Our primary focus is on the case where the corruption threshold t is small. This enables the use of PRSS techniques that give rise to simpler and more efficient protocols, but incur (an offline) cost that scales exponentially with t. We are motivated by two main limitations of current techniques.

The overhead of packed secret sharing. A major source of concrete overhead in the aforementioned theory-oriented works is the use of a "share packing" technique [24] in which secret-shared values are arranged into blocks, and a set of shares can simultaneously encode several values at the same per-party cost. This technique natively supports computing a single circuit on many inputs in parallel (also known as a "SIMD computation"), by computing operations simultaneously on all values within a block. However, it requires a costly routing mechanism for general computations. This overhead applies even in the semi-honest setting, but introduces additional challenges in the malicious setting. While the initial $O(\log n)$ overhead of the routing-based technique from [20] was recently improved to a constant [36], this comes at the cost of poor concrete efficiency.

Extending the ideas of these works, one may observe that existence of certain useful linear correlations across parties would enable avoiding these routing overheads altogether. The desired correlations correspond to sets of packed shares of secret random values, where different sets include the same random values in different computation "slot" positions, in line with the routing of wires within the computation circuit. But, unlocking these savings demands a large number of *different* rerouting patterns, whose generation would destroy the optimization savings in existing works. Much of the effort in previous works [19, 21, 20, 4, 33, 27, 36] was spent on efficient distributed protocols for generating these linear correlations.

Tolerating stragglers. One advantage of MPC with a strong honest majority, which serves as a primary motivation for the current work, is the potential for better robustness, in turn leading to *reduced latency* in realistic network environments. Existing MPC protocols with $n = 2t+1$ parties require at least one of the parties to wait for messages from *all other parties* before proceeding to the next round. In particular, in protocols that follow the DN blueprint, the leader needs to wait until it hears back from all other parties. But in reality, communication is not fully synchronous. Even in a semi-honest setting, protocol executions suffer from variance in communication delays and occasional message-delivery failures. This is sometimes referred to as the problem of *stragglers*. To deal with this problem, practical distributed systems typically employ redundancy to allow proceeding with the computation as long as "sufficiently many" messages were received. See [42] for empirical studies of the impact of stragglers on realistic network.

Interestingly, achieving robustness to stragglers becomes more challenging when some parties can be malicious. Standard secure protocols with good concrete efficiency do not have this feature even when $n > 3t$. While such protocols are able to terminate in the face of up to t stragglers, this occurs at the cost of labeling these parties as corrupt, and their secrets are no longer protected. Alternatively, attempting to run DN-style protocols in an "optimistic mode," by simply having the leader wait for the first $2t$ messages to arrive, gives rise to a subtle "double-dipping" attack that allows a malicious leader to learn private information. Previous solutions for this attack(see [26, 35]) require significantly more interaction and are not suitable for efficiently dealing with transient faults; See Section 5.1 for more details.

1.1 Our Contributions

Motivated by the above opportunities and challenges, we present new techniques for MPC within the setting of a *strong honest majority*, $n > 2t + 1$, focusing on the case of small[5] values of t that enable efficient use of PRSS. We make the following two main contributions.

Contribution 1: Generalized pseudorandom secret sharing (PRSS). As noted above, PRSS enables a secure non-interactive generation of (pseudo)random values that are uniformly distributed over some linear vector space. It relies on a low-communication setup, where independent pseudorandom function (PRF) seeds are distributed to different subsets of the parties. The prominent cost metric of a PRSS scheme is the number of such seeds required for the parties to each compute their entry within the sampled vector. Following a general framework of Gilboa and Ishai [28], Cramer et al. [18] described PRSS techniques for sharing degree-d polynomials between n parties using $\binom{n}{d}$ seeds, $\binom{n-1}{d}$ per party, targeting the typical use-case where the security threshold t is equal to d. Motivated by the fact that in MPC with strong honest majority we have $t < d$, we present new PRSS constructions exploiting this gap.

Our constructions leverage suitable combinatorial designs, and yield a dramatic efficiency improvement when $t \ll d$, not only for standard degree-d correlations but also for several useful generalizations. This includes correlations for locally converting between slot configurations in "share packing," which enable

[5] More precisely, our protocols have storage and (offline) computation costs that grow exponentially in t but linearly in the number of parties n. Thus, when t is a small constant, they can be practical even for a large n.

us to avoid the concrete overhead of prior works on MPC based on share packing [20, 33, 27]. We remark that our PRSS results are independently motivated by other applications beyond the context of MPC, including threshold cryptography, advanced cryptographic primitives, and targeted multi-party protocols (e.g., [15, 23, 6, 13, 7]).

We provide a general transformation yielding PRSS schemes from any instance of a so-called "covering design" with appropriate parameters. An (n, m, t)-*cover* is a collection of size-m subsets $S_i \subset [n]$, such that every subset of t elements of $[n]$ is covered by some set S_i. The goal is to do so with the fewest number of such sets S_i. Construction of covering designs is a topic of combinatorial research, where bounds are known for small parameters, and several results are known in the larger parameter regime (see Section 3.3 for discussion). While it is not hard to see that the seed replication pattern of a PRSS must induce a covering design, the converse direction is less obvious. Indeed, our transformation incurs a small overhead that leaves a $(d + 1)$ multiplicative gap between the upper and lower bounds on the number of seeds for the case of degree-d polynomials.

The following theorem summarizes our general transformation from covering designs to PRSS for degree-d polynomials, as well as some corollaries obtained by plugging in existing covering designs from the literature (cf. [32]).

Theorem 1.1 (PRSS for degree-d polynomials from covering designs, informal). *Let n, d, t be positive integers such that $t < d < n$. Given an $(n, d + 1, t)$-cover of size k, one can construct a PRSS scheme for sharing random degree-d polynomials between n parties with security threshold t, requiring $k(d + 1)(n - d)/n$ PRF seeds per party. As a special case, plugging in existing covering designs for small t, we obtain the following:*

- *For $t = 1$, any n: $\left\lceil \frac{n}{d+1} \right\rceil \frac{(d+1)(n-d)}{n}$ PRF seeds per party (or just $n - d$ when $(d + 1)|n$).*
- *For $t = 2$, any $n \leq 3(d + 1)$: $13(d + 1)$ PRF seeds per party.*

We further obtain PRSS for "double Shamir sharing" (i.e. two random polynomials of degrees d and $2d$ with the same evaluations on given $d - t + 1$ points) with roughly twice as many PRF seeds.

In comparison to the parameters above, the naive baseline from [18] is $\binom{n-1}{d}$ seeds per party, which in the case that $t < d$ can be improved to $\binom{n-1}{t}$ seeds per party (we show the details in the full version of this paper). Plugging in explicit covering design constructions from the literature, the PRSS solutions obtained via Theorem 1.1 provide significant savings to even this improved baseline. For example:

- $(n, d, t) = (48, 15, 4)$ requires $2,772$ seeds per party, versus baseline $\binom{47}{4} = 178,365$.
- $(n, d, t) = (49, 23, 4)$ requires 484 seeds per party, versus baseline $\binom{48}{4} = 194,580$.

- $(n, d, t) = (49, 23, 8)$ requires $57,281$ seeds per party, versus baseline $\binom{48}{8} \approx 3.7 \cdot 10^8$.

For additional data points, see the full version. Our PRSS constructions go beyond basic Shamir or double-Shamir shares, to a generalized form of PRSS that allows local generation of packed pseudorandom secrets with an arbitrary replication pattern. We achieve this with with additional redundancy of seeds to parties. However, the resulting complexity still provides significant savings as an alternative to existing approaches within motivated regimes. We refer the reader to Section 3.6 for a detailed treatment.

Contribution 2: Cheap straggler resilience. We propose a novel technique for dealing with the "straggler" problem of delayed messages, allowing the protocol to continue the execution once sufficiently many messages are received. In doing so, we need to defend against the subtle "double-dipping" attack mentioned above. In contrast to alternative approaches for defending against this attack [26, 35], our approach has no extra cost to the round complexity of the protocol and only a sublinear additive communication overhead. Our protocol makes black-box use of our PRSS construction to produce the required randomness without interaction.

Combining the above tools to obtain efficient MPC protocols with security against malicious parties requires additional ideas. In particular, we need to adapt the distributed zero-knowledge proof techniques of Boneh et al. [8] to the setting of MPC based on packed secret sharing. See additional discussion below.

The features of our final protocol are captured by the following theorem.

Theorem 1.2 (Malicious security with straggler resilience, informal). *Let $t \geq 1$ be a security threshold, $\ell \geq 1$ a packing parameter, $n \geq 2t + 2\ell - 1$ a number of parties, and $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n + t + 2\ell$. Then, for any arithmetic circuit C over $\mathbb{F}$ with S multiplication gates and depth D, there is an n-party protocol for C with the following efficiency and security features:*

- *The protocol makes a black-box use of any pseudorandom function;*
- *Excluding $O(1)$ rounds of preprocessing and postprocessing, the protocol consists of D epochs, where in each epoch P_1 sends a message to each other party P_i and receives a message back from each P_i;*
- *It achieves security with abort in presence of t malicious parties even if $\tau = n - (2t + 2\ell - 1)$ messages, chosen by the adversary, are dropped in each epoch;*
- *If the parties follow the protocol, it terminates successfully even if τ messages, chosen by the adversary, are dropped in each epoch;*
- *Communication cost is $\left(\frac{3}{\ell} - \frac{2t+2\ell+1}{n \cdot \ell}\right) S + o(S)$ elements of $\mathbb{F}$ sent per party.*

We further discuss the communication, computation, and storage costs in the following remarks.

Remark 1.1 (Sensitivity to the topology of C.). As in other protocols based on packed secret sharing, the communication complexity bound in Theorem 1.2 assumes that the circuit C is "non-pathological" in the sense that its width is bigger than the packing parameter ℓ. (Otherwise there is an extra communication cost resulting from empty slots.) Since we typically expect ℓ to be much smaller than the circuit size, this condition is met for almost all natural instances of big circuits.

Remark 1.2 (On the cost of PRSS.). The generalized PRSS primitive influences the local storage and computational cost, which can be performed offline and are practical for small t even for large values of ℓ and n; see the full version of this paper for concrete numbers. By increasing the degree parameter d of the generalized PRSS construction beyond the minimum required by t and ℓ, we get better PRSS complexity at the cost of a lower straggler resilience threshold τ.

Remark 1.3 (On communication complexity.). When $\ell = 2$, the amortized communication cost in Theorem 1.2 is always less than 1.5 elements per party per gate, and when $\ell = 3$ it goes below 1 element per party. We present concrete efficiency analysis of our protocol in the full version of the paper, showing that as we increase n and ℓ, our protocol not only can withstand stragglers, but also achieves *lower total communication* than the best known semi-honest protocols with $n = 2t + 1$ parties. In particular, whenever $\ell = \Omega(n)$ the total communication complexity (ignoring lower order additive terms) is $O(s)$.

Technical challenges & contributions. Our final MPC protocol builds on new solutions for the following main challenges:

- *Generalized pseudorandom secret sharing (PRSS)* based on combinatorial designs that take advantage of the gap between the polynomial degree d and the security threshold t to reduce computation and storage costs.
- *Packed secret sharing beyond SIMD*, without the asymptotic or concrete overhead of previous techniques [20, 33, 27]. Our solution relies on generalized PRSS for cheaper batch generation of useful linear correlations, for "repacking" secret shared values in different orders.
- *Preventing "double-dipping" attacks*, identified by [35, 26], which exploit the redundancy of encoding across parties in a strong honest majority to obtain related secret values under the same random mask (see below; note that this attack arises even without share packing). The works of [35, 26] protect against the attack using methods that require participation from *all* parties and increase the round complexity by 2x or more; we do so while supporting resilience to stragglers, and with essentially no extra online cost.
- *Applying sublinear distributed zero knowledge [8] on packed shares*, as well as achieving batched verification with missing shares (due to stragglers). The former challenge arises again from the non-SIMD structure of general computation, here relating to the statements to be efficiently *verified*. The latter issue pertains to verifying consistency of several robustly secret shared values, when each secret has a *different* subset of shares missing, corresponding to different sets of straggling parties.

1.2 Related Work

We mention here specific recent works relating to our second contribution, of MPC in the strong honest majority setting achieving concrete efficiency and straggler resilience.

PRSS-based vs. interactive correlated randomness generation. In this work, we use non-interactive PRSS to generate the double sharing required for the DN multiplication protocol. While we improve the efficiency of PRSS dramatically (when the polynomial degree d is higher than the corruption threshold t), the computational overhead still limits the practical use of this method to a relatively small number of corrupted parties t. See the full version for concrete estimates of computational cost. An alternative to the PRSS-based approach is using an interactive protocol, but with computation that scales polynomially with the number of parties. The state-of-the-art protocol by Goyal *et al.* [34] shows how to generate the double sharing with communication of just 0.5 field element sent per party. This implies that our method requires approximately 25% less overall communication. More importantly, the method of [34] does not support straggler resilience and applies only to gate-by-gate evaluations. While it can be easily extended to SIMD circuits, it does not extend to general non-SIMD circuits with packed secrets. Finally, the correlated randomness generation procedure from [34] requires interaction between all parties, which can be prohibitive in other applications scenarios.

MPC with strong honest majority. Concretely efficient MPC in the strong honest majority setting has gained recent focus, including the works of Gordon *et al.* [33] and Beck *et al.* [27]. In comparison, their protocols scale to a larger number of parties, while our approach provides better efficiency for the regime of small corruptions t. This is due largely to our ability to generate the necessary setup correlations with minimal interaction via generalized PRSS. In addition, our protocols provide straggler resilience (yielding savings in settings with latency variance), whereas [33, 27] assume a fully synchronous network. Finally, in these works, malicious security comes with a multiplicative overhead, whereas in our protocol, the overhead is sublinear in the size of the circuit.

A very recent work of Goyal *et al.* [36] shows how to achieve asymptotic constant communication cost per party in this setting for general non-SIMD circuits with information-theoretic security, but with poor concrete efficiency and without stragglers resilience.

MPC with partial synchrony. A number of works have studied MPC with various (stronger) flavors of partial synchrony from the perspective of feasibility, without focus on concrete efficiency. For example, the work of Zikas *et al.* [53] provides unconditionally secure protocols in a model where parties can additionally be send-omission or receive-omission corrupted. Guo *et al.* [38] consider a model where parties can periodically go offline and return. In Badrinarayanan *et al.* [3] parties can turn non-adversarially "lazy." Both of the latter rely on heavy cryptographic tools, such as (multi-key) fully homomorphic encryption.

Finally, a handful of works have considered concretely efficient MPC with forms of partial synchrony, with incomparable conclusions. Hirt and Maurer [41] consider a mixed model of malicious and fail-stop adversaries, achieve perfect security, but with larger overall cost (e.g., without the savings of share packing). The "Fluid MPC" work of Choudhuri *et al.* [17] builds efficient protocols within a very different model, designed for long computations, where in each period of time, a different set of parties carry-out the computation.

2 Preliminaries

Notation. Let $P_1, \ldots, P_n$ be the set of parties and let t, ℓ, d be integers such that $d \geq t + \ell - 1$ and $n \geq 2d + 1$. The parameter t bounds the number of parties that can be corrupted, the parameter ℓ denotes the size of the block of secrets that are evaluated together, and d will be the degree of the polynomial defined below. We use $[n]$ to denote the set $\{1, \ldots, n\}$ and denote by $\mathbb{F}$ a finite field.

2.1 Threshold Secret Sharing

Definition 2.1. *A d-out-of-n secret sharing scheme is a protocol for a* dealer *holding a secret value v and n parties $P_1, \ldots, P_n$. The scheme consists of two interactive algorithms:* $\mathsf{share}(v)$*, which outputs shares $(v_1, \ldots, v_n)$ and* $\mathsf{reconstruct}(\{v_j\}_{j \in T}, i)$*, which given the shares $v_j, j \in T \subseteq [n]$ outputs v or $\bot$. The dealer runs* $\mathsf{share}(v)$ *and provides P_i with a share v_i of the secret v. A subset of users T run* $\mathsf{reconstruct}(\{v_j\}_{j \in T}, i)$ *to reveal the secret to party P_i. The scheme must ensure that no subset of d shares provide any information on v, while $v =$* $\mathsf{reconstruct}(\{v_j\}_{j \in T}, i)$ *for T only if $|T| \geq d+1$. We say that a sharing is* consistent *if* $\mathsf{reconstruct}(\{v_j\}_{j \in T}, i) = \mathsf{reconstruct}(\{v_j\}_{j \in T'}, i)$ *for any two sets of honest parties $T, T' \subseteq \{1, \ldots, n\}$, and $|T|, |T'| \geq d + 1$.*

Packed Shamir Secret Sharing In Shamir's secret sharing scheme [48], the dealer defines a random polynomial $p(x)$ of degree d over a finite field $\mathbb{F}$ such that the constant term is the secret. Each party is associated with a distinct non-zero field element $\alpha \in \mathbb{F}$ and receives $p(\alpha)$ as its share of the secret. Since the degree of the polynomial is d, any $d + 1$ points are sufficient to compute the secret. We use the notation $[\![x]\!]_d$ to denote a sharing of x via a polynomial of degree d.

Two properties of this scheme that are very useful for MPC are: (1) linear operations on secrets can be computed locally on the shares, since polynomial interpolation is a linear operation; (2) given shares of x and y, the parties can locally multiply their shares to obtain a sharing of degree $2d$ of $x \cdot y$.

In this work, we use a generalization of Shamir's sharing scheme where multiple secrets are being encoded together, introduced by Franklin and Yung [24] and known as "packed secret sharing". This is achieved by storing the secrets on multiple points. Note however that if we pack ℓ secrets together on a polynomial of degree d, then the corruption threshold is being reduced to $t = d - \ell + 1$. Throughout this paper, we will use the notation $[\![x_1 \cdots x_\ell]\!]_d$ to denote a sharing of the block $x_1, \ldots, x_\ell$ using a polynomial of degree d, and assume that

$x_1, \ldots, x_\ell$ are stored at points $0, -1, \ldots, -\ell+1$ respectively and that the share of P_i is the value at the point i. Observe that the properties mentioned above apply to packed secert sharing as well. Namely, given a constant $\alpha, \beta \in \mathbb{F}$ and two sharings $[\![x_1 \cdots x_\ell]\!]_d$, $[\![y_1 \cdots y_\ell]\!]_d$, the following are local operations over the shares: (1) $[\![(\alpha x_1 + \beta y_1) \cdots (\alpha x_\ell + \beta y_\ell)]\!]_d = \alpha\, [\![x_1 \cdots x_\ell]\!]_d + \beta\, [\![y_1 \cdots y_\ell]\!]_d$; (2) $[\![x_1 y_1 \cdots x_\ell y_\ell]\!]_{2d} = [\![x_1 \cdots x_\ell]\!]_d \cdot [\![y_1 \cdots y_\ell]\!]_d$.

We say that a sharing $[\![x]\!]_d$ or $[\![x_1 \cdots x_\ell]\!]_d$ is inconsistent if all points do not lie on the same polynomial of degree d. Given all shares, this can be easily checked by using $d+1$ points to reconstruct the polynomial and checking whether the remaining points lie on this polynomial

2.2 Computation Model: Layered Straight-Line Programs

In this work, we present a multi-party protocol for performing arithmetic computations over a finite field. In the MPC literature, arithmetic computations are usually represented by a circuit or a straight line program (SLP) with addition and multiplication gates/operations. We use the notion of SLP, but choose a slightly different representation, with one instruction, which we call "bi-linear", that captures the two operations together. This model will allow us a simple and more clearer description of our protocols, and in particular, make the trick to achieve "free-addition" easier to describe.

Definition 2.2 (Layered straight-line program (SLP)). *A* straight-line programs (SLP) *over $\mathbb{F}$ is defined by an arbitrary sequence of the following kinds of instructions:*

- *Load an input into memory: $R_j \leftarrow x_i$.*
- *Bilinear instruction: $R_j \leftarrow (\sum_{\omega=1}^{w} a_\omega \cdot R_\omega) \cdot (\sum_{\omega=1}^{w} b_\omega \cdot R_\omega)$*
- *Output value from memory, as element of $\mathbb{F}$: $O_i \leftarrow R_j$.*

Here $x_1 \ldots, x_n$ are inputs, *$R_1, \ldots, R_w$ are* registers *and $a_1, \ldots, a_w, b_1, \ldots, b_w$ are* public constants *in $\mathbb{F}$. We define the* size *of an SLP to be the number of instructions. A* layered SLP *is an SLP where the instructions are partitioned into sets called layers such that the inputs to instructions in layer j were computed in layer $k < j$. An ℓ-layered SLP is a layered SLP in which the number of instructions in each layer is a multiple of ℓ.*

For simplicity, we assume in our MPC protocols for SLP that each party holds one input and receives one output at the end. However, the protocols naturally extend to the general case of multiple inputs or outputs per party.

Remark 2.1 (Simulating arithmetic circuits by layered SLPs). Every arithmetic circuit of size S (counting only multiplication gates, inputs, and outputs) can be converted into an SLP of size S by sorting its gates in an arbitrary topological order. The "ℓ-layered" notion of SLP intuitively corresponds to a lower bound on the circuit width. In particular, an SIMD circuit computing $k \geq \ell$ copies of a size-S circuit C on k distinct inputs can be written as an ℓ-layered SLP of size kS. Any layered SLP can be converted into an ℓ-layered one by naively adding dummy gates if needed, where the latter adds $(\ell - 1)$ times the depth in the worst case. But almost all "natural" instances of big circuits can be compiled into ℓ-layered SLPs with no overhead.

3 Generalized Pseudorandom Secret Sharing

An important resource for our main protocol is a packed secret sharing of blocks of ℓ secrets that are randomly sampled from a given linear subspace. In this section, we show how the parties can securely generate arbitrarily many such blocks of secrets without any interaction, assuming a short setup step where they distribute seeds for a Pseudorandom Function (PRF). Subsequently, shares are obtained by local computation on the seeds. We refer to this problem as generalized pseudorandom secret sharing (PRSS). This primitive is useful beyond the context of this work, and our results are useful even without any share packing (i.e., when $\ell = 1$).

More abstractly, we can view the problem as that of efficiently realizing a *linear correlation*, namely an ideal functionality that picks a random vector from a public linear space and delivers one or more entries of this vector to each party. To be applicable in an MPC protocol, even with a semi-honest adversary, the linear correlations must be generated *securely.* Loosely speaking, an adversary should not get any information on the shares of honest parties beyond what follows from the public linear correlation, even given the information that the adversary holds.

The ideal functionality $\mathcal{F}_{\mathrm{LinRand}}$. We will make security arguments relative to an ideal functionality $\mathcal{F}_{\mathrm{LinRand}}$ for sharing instances of *linear correlated randomness.* More concretely, $\mathcal{F}_{\mathrm{LinRand}}$ is parametrized by some linear subspace, and in each invocation it picks a random vector from that linear subspace and distributes one or more entries to each party. Both the linear space and the assignment of which entry goes to what party are public. It is only the actual vector sampled from the linear subspace that should remain secret.

Security is defined with respect to a *static* adversary who may corrupt up to t parties. Concretely, the real world view of the adversary together with the outputs of honest parties should be indistinguishable from an ideal world in which the adversary chooses the corrupted parties' shares, and then the honest parties' shares are sampled from the target correlation conditioned on this choice. This can be formally viewed as a multiparty instance of a Pseudorandom Correlation Function (PCF), recently defined by Boyle et al. [10], applied to *linear* correlations. The notion of PCF naturally extends the notion of a Pseudorandom Correlation Generators (PCG) [9], analogously to the way a standard PRF extends a standard PRG.

We are interested in t-secure realizations of $\mathcal{F}_{\mathrm{LinRand}}$ that have the following structure: (1) During an offline setup phase, a trusted dealer picks random and independent PRF seeds, and distributes each seed to a subset of the parties.[6] (2) Next, to realize a fresh invocation of $\mathcal{F}_{\mathrm{LinRand}}$ with common identifier id, each

[6] This setup can alternatively be implemented by a secure MPC protocol.

party *locally* evaluates the PRF with each seed it owns on one or more inputs derived from id, and outputs a *fixed linear combination* of the PRF outputs. (The linear combination is fixed and does not change from one id to the next.)

3.1 Overview

Following prior work, we reduce the goal of secure realization of $\mathcal{F}_{\mathrm{LinRand}}$ to an information-theoretic problem where the PRF seeds are replaced with true randomness. Namely, we consider locally generating an instance of the target correlation with perfect t-security given independently random field elements that are replicated between the parties. In the PRF-based computational realization of $\mathcal{F}_{\mathrm{LinRand}}$, the random field elements will be pseudorandomly sampled using the PRF. Security under the above PCF-style definition reduces to information-theoretic security via a standard hybrid argument.

The PRSS problem was first implicitly studied by Gilboa and Ishai [28]. Cramer, Damgård, and Ishai [18] made this notion explicit and described a simple construction for the case of generating t-out-of-n Shamir sharing of random values. This construction is a useful building block in many cryptographic applications. Here we extend the notion and construction of PRSS to more general settings that are motivated (among other applications) by MPC with strong honest majority. We show that a gap between the degree d and the security threshold t can give rise to dramatic efficiency improvements. Concretely:

- We start by extending the standard PRSS problem to the case where the degree of the Shamir-sharing polynomial can be larger than the security threshold t, and reduce this problem to a well-studied combinatorial design problem. This construction can be used for example to implement efficient distribution of *packed Shamir sharing* [24] of random values, and can be useful for many other applications.
- We show how to use the above construction in a black-box fashion to get efficient implementation of the kind of "double sharing" needed for protocols that follow the approach of Damgård-Nielsen (DN) [22]. Specifically, we implement the target correlation of two secret-sharing of the same (possibly packed) random values, one with a degree-d polynomial and the other with a degree-$2d$ polynomial.
- We show an extension of this technique to the harder case where we have degree-$2d$ sharings of random values, and degree-d sharings of *arbitrary linear combinations* of those random values. This is used to generate random packed secrets that satisfy given replicated constraints, as needed by efficient MPC protocols for *general circuits* based on packed secret sharing [19, 20].

We note that our techniques can be used to improve the efficiency of even more general forms of linear correlation, but leave systematic study of their application to future work.

3.2 The Gilboa-Ishai Framework

The functionality that we want to implement distributes linearly correlated random variables over some field $\mathbb{F}$ to n parties. The functionality is parameterized by a matrix $C \in \mathbb{F}^{N \times K}$ whose columns span a linear code (i.e., linear subspace of $\mathbb{F}^N$), and by a mapping $\rho : [N] \to [n]$ saying which party gets what entry of the output vector. The functionality chooses a random vector $\boldsymbol{v}$ in the code (by choosing a uniformly random $\boldsymbol{u} \leftarrow \mathbb{F}^K$ and setting $\boldsymbol{v} := C\boldsymbol{u}$), then privately sends to each party i' all the entries indexed by $\rho^{-1}(i')$.

Implementing this functionality without any interaction (beyond pre-distribution of PRF seeds) was studied by Gilboa and Ishai [28], in the information-theoretic setting where the PRF seeds are replaced by true randomness. In their framework, implementation of the linear-correlation functionality consists of:

- Input distribution, where an honest dealer draws $x_1, x_2, \ldots, x_k \in \mathbb{F}$ uniformly at random, and distributes each x_j to some subset of parties $S_j \subset [n]$;
- Local output computation, where each party i locally computes and outputs its entries of $\boldsymbol{v}$ from the x_j's that it received.

The complexity measures of interest for such a solution are:

- The number of distinct subsets S_j, corresponding to the number of PRF seeds to be distributed, and
- The sum $\sum_{j=1}^{k} |S_j|$, corresponding to the total number of pseudorandom field elements to be derived from these PRF seeds, across all the parties.

All the known implementations, including the ones that we describe here, rely on "small-support codewords" and the Gilboa-Ishai security criteria: A solution is specified by a "sparse" matrix $M \in \mathbb{F}^{N \times k}$ (typically $k \gg K$), whose columns span the same code as C. The output is computed by choosing a random vector $\boldsymbol{x} = (x_1, \ldots, x_k)$ and setting $\boldsymbol{v} := M\boldsymbol{x}$, and each party gets all the x_j'es that it needs in order to carry out this computation. Specifically, for an entry $\boldsymbol{v}[i]$ that belongs to party $\rho(i)$, we give that party the random elements x_j for which $M[i,j] \neq 0$, making it possible for this party to compute the inner product between $\boldsymbol{x}$ and the i'th row of M. Hence the sets $S_1, \ldots, S_k$ are defined

$$S_j = \{i' \in [n] : \exists i \in [N],\ M[i,j] \neq 0 \text{ and } i' = \rho(i)\}, \tag{1}$$

(For example, if the mapping ρ assigns entries 1 through 10 in $\boldsymbol{v}$ to Party 1 then the only x_j values that *are not given to this party* correspond to columns of M where the top 10 entries are all zero.) Clearly, the sparser the matrix M is, the fewer x_j values that must be distributed and the smaller we can make the sets S_j.

Gilboa and Ishai proved a necessary and sufficient criterion for security within this framework. Fix a code which is generated by the columns of the matrix C, and a solution matrix M whose columns span the same code. For a subset of parties $T \subset [n]$, let I_T be all the rows that belong to parties in T, and J_T be all the indices of x_j's that members of T get to see. That is, with the S_j's defined as in Equation (1), we have

$$I_T = \bigcup_{i' \in T} \rho^{-1}(i'), \text{ and } J_T = \big\{ j \in [k] : S_j \cap T \neq \emptyset \big\}.$$

Denote by $\mathsf{C}_{\bar{T}}$ the restriction of $\mathsf{span}(C)$ to only the codewords that are zero in all the coordinates I_T. Also denote by $M'_{\bar{T}}$ the submatrix of M consisting of the rows in the complement $I_{\bar{T}} = [N] \setminus I_T$ and the columns in the complement $J_{\bar{T}} = [k] \setminus J_T$ (i.e., the ones corresponding to x_j's that *none of the parties in* T *receives*).

Lemma 3.1 ([28]). *Let $C \in \mathbb{F}^{N \times K}$ and $M \in \mathbb{F}^{N \times k}$ be two matrices with the same column space (so M describes a solution to the distribution of a codeword from $\mathsf{span}(C)$).*

For a subset of parties $T \subset [n]$, the solution specified by M is secure against a corrupted T iff the rank of $M'_{\bar{T}}$ equals the dimension of $\mathsf{C}_{\bar{T}}$.

3.3 Technical Tool: Covering Designs

The main technical tool that we use in our construction is the following notion of covering designs:

Definition 3.1 ((n, m, t)-cover). *Fix integers $n \geq m \geq t > 0$, and let $\mathcal{C} = (S_1, \ldots, S_k)$ be a collection of k different subsets $S_j \subset [n]$, all of size $|S_j| = m$. $\mathcal{C}$ is said to be an (n, m, t)-cover if for every size-t subset $T \subset [n], |T| = t$, there is a set $S_j \in \mathcal{C}$ that covers it, $T \subseteq S_j$. We will refer to an (n, m, t)-cover as a t-cover when n, m are clear from the context.*

This notion is equivalent to the notion of t-immunity of Alon et al. [2], in which for every subset T there is a set S_j in the collection such that $T \bigcap S_j = \emptyset$. The collection $\mathcal{C}$ is an (n, m, t)-cover iff the complement sets $[n] \setminus S_j$ form an $(n, n-m, t)$-immune collection. The smallest number of subsets in an (n, m, t)-cover is also known as the hypergraph Turán number $\mathcal{T}(n, n-t, n-m)$ in honor of Paul Turán who initiated the study of these objects in [50, 51].

The parameters of covering designs have been studied extensively, e.g. see [49, 25] for surveys, but the exact value is still an open problem in the general case. The best known analytical bounds for small values of t are summarized in a Handbook of Combinatorial Designs chapter by Gordon and Stinson [32]. A good online resource that collects the best known bounds for concrete values of n, m, t with $t \leq 8$, including ones found via computer search, is Gordon's covering designs web page [1].

For general values of t, Micali and Sidney [44] proposed to construct an (n, m, t)-cover by randomly choosing $\binom{n}{t}/\binom{m}{t} \ln \binom{n}{t}$ subsets of size m from $[n]$ and used a probabilistic argument to show that with good probability this collection is an (n, m, t)-cover. Pieprzyk and Wang [39] construct a deterministic, greedy algorithm that achieves the same bound on the size of the collection. Both works were motivated by variants of the PRSS problem where the seeds are stored in a replicated form, without the compressing share conversion step from [28, 18]

A range of parameters which is especially appealing for our MPC protocol is constant t, and m which is a linear fraction of n, e.g., $m = n/3$. In this case, the protocol can cope with a large number of stragglers and reduce communication by packing. When t is constant, the constructions in [44, 39] have collections of size $O(\log n)$.

We next describe a simple construction that achieves a constant-sized collection for $t = O(1)$ and $m = \Omega(n)$, when t divides m and m divides n. Denote $c = n/m$ and partition $[n]$ into ct subsets $R_1, \ldots, R_{ct}$ of equal size. Let the collection $S_1, \ldots, S_k$ be all the possible choices of t subsets $R_{i_1} \cup \cdots \cup R_{i_t}$. Obviously, each $|S_j| = t(n/ct) = m$ and for every $T \subseteq [n], |T| = t$ there exists some S_j such that $T \subseteq S_j$. The size of the collection is $\binom{ct}{t}$, which for constant t and c is constant, improving over the construction of [44, 39].

Taking for each parameter set (n, m, t) the minimal cover size between the simple construction and the construction in [39] provides a baseline construction for t-covers. This baseline achieves an upper bound for the cover size, which is bigger than the minimal possible size by a factor of at most $O(\log n)$, due to a simple lower bound of $\binom{n}{t}/\binom{m}{t}$ on this size (see, e.g., Theorem 11.19 in [32]). Both the upper bound of the baseline construction and the simple lower bound are generally not tight. Improved bounds for certain parameter ranges can be found in [1].

3.4 Generalized PRSS for Degree-d Polynomials

It is easy to see (see Theorem 3.2) that t-covers are necessary for t-secure distribution in the Gilboa-Ishai framework, since any corrupted subset must miss at least some of the x_j's. Here we observe that the other direction is also useful, establishing a close connection between the size of the best $(n, d+1, t)$-cover and the complexity of PRSS for distributing random degree-d polynomials between n parties with security against t-collusions.

Theorem 3.1 (Generalized PRSS for degree-d polynomials). *Fix integers $n \geq d > t > 0$. A size-k' $(n, d+1, t)$-cover can be used to construct a generalized PRSS solution for t-secure distribution of degree-d polynomials, with the following complexity measures:*

- *The number of distinct subsets (or PRSS seeds) is $k = k'(d+1)$, and*
- *The total subset size (storage) is $\sum_i |S_i| = k'(d+1)(n-d)$ and*
- *The total number of PRF calls is $k'(d+1)(n-d)$.*

Proof: Let $\mathcal{C}' = (S'_1, \ldots, S'_{k'})$ be a size-k' $(n, d+1, t)$-cover, i.e. it consists of k' subsets, each of size $d+1$, that cover all the t-subsets. We then consider all the subsets that are obtained by removing one element from any of the S'_j's,

$$\bar{\mathcal{C}} = \{S' \setminus \{j\} : \ S' \in \mathcal{C}', j \in S'\}.$$

Clearly, there are at most $k \leq k'(d+1)$ distinct subsets in $\bar{\mathcal{C}}$, each of size d. Let us denote the subsets in $\bar{\mathcal{C}}$ by $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_k$, and we use these subsets in the CDI construction to distribute a random degree-d polynomial. We let $P_{\bar{S}_j}$ be the unique polynomial of degree d interpolated from

$$P_{\bar{S}_j}(X) = \begin{cases} 0 & \text{if } X \in \bar{S}_j \\ 1 & \text{if } X = 0 \end{cases}$$

As before, $P_{\bar{S}}$ is a nonzero degree-d polynomial, whose zeros are exactly all the parties in $\bar{S}_j$. A random vector $\boldsymbol{x} \in \mathbb{F}^k$ therefore defines the polynomial $Q_{\boldsymbol{x}}(X) = \sum_j x_j \cdot P_{\bar{S}_j}(X)$, and each party $i \in [n]$ gets the x_j's corresponding to the $\bar{S}_j$'s *that do not include* i, and can compute $Q_{\boldsymbol{x}}(i)$ from these x_j's. Thus, there are $k'(d+1)$ distinct subsets, each of cardinality $n-d$. This implies that the total stroage is $k'(d+1)(n-d)$ as the theorem states. Since each seed is used once, the total number of PRF calls is also the same.

In the language of the Gilboa-Ishai framework, the matrix $M \in \mathbb{F}^{n \times k}$ is defined by $M[i,j] = P_{\bar{S}_j}(i)$, and the distribution sets are exactly the complementing sets $S_j = [n] \setminus \bar{S}_j$ (namely we distribute each x_j to the complement of some $S' \in \mathcal{C}'$, together with one more element). The complexity measures are obvious.

It remains therefore to show security against a collusion of t parties, which for degree-d polynomials means showing that for every t-subset T, the submatrix $M'_{\bar{T}}$ has rank at least $d+1-t$. Fix a t-subset $T \subset [n]$, so there is a subset $S' \in \mathcal{C}'$ that covers it. Consider now the sub-matrix corresponding to the subsets $\bar{S}$ that were obtained by removing from S' one element *which is not in* T (hence those sets $\bar{S}$ still all cover T). That is, we consider the sub-matrix $M_{T,S'}$ of $M[i,j] = P_{\bar{S}_j}(i)$, consisting of the rows for $[n] \setminus T$ and the columns for $S_j = ([n] \setminus S') \cup \{j'\}$ for all $j' \in S' \setminus T$. Clearly $M_{T,S'}$ is a sub-matrix of $M'_{\bar{T}}$, it has $n-t$ rows and $d+1-t$ columns (since S' covers T), and it has the form

$$M_{T,S'} = \begin{bmatrix} * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \\ * & & & \\ & & \ddots & \\ & & & * \end{bmatrix},$$

where the $*$'s are non-zero and everywhere else there are zeros. The top rows $* \cdots *$ correspond to $[n] \setminus S'$ and the bottom rows correspond to $S' \setminus T$. The last $d+1-t$ rows of this matrix are linearly independent, hence the rank of $M_{T,S'}$ is $d+1-t$, as needed for the Gilboa-Ishai condition. ■

Corollary 3.1. *Fix integers $n \geq d > 1$. Then, the following holds for generalized PRSS solutions for t-secure distribution of degree-d polynomials with $t = 1, 2$:*

1. *There exists a solution for $t = 1$ with $\left\lceil \frac{n}{d+1} \right\rceil (d+1)$ total seeds, $\left\lceil \frac{n}{d+1} \right\rceil \frac{(d+1)(n-d)}{n}$ seeds stored by each party and $\left\lceil \frac{n}{d+1} \right\rceil \frac{(d+1)(n-d)}{n}$ calls to the PRF made by each party.*

2. *If $n \leq 3(d+1)$ then there exists a solution for $t = 2$ with $13(d+1)$ total seeds, $13(d+1)(n-d)/n$ seeds stored by each party and $13(d+1)(n-d)/n$ calls to the PRF made by each party.*

We can also prove a nearly-matching lower bound Theorem 3.1 on the solution complexity for t-secure distribution of degree-d polynomials, in terms of the achievable size for $(n, d+1, t)$-covers. This naturally generalizes a similar negative result for standard PRSS from [18]. The proof is in the full version.

Theorem 3.2 (Necessity of cover designs). *Any generalized PRSS solution for t-secure distribution of degree-d polynomials that has k distinct subsets implies an $(n, d+1, t)$-cover of size $k' \leq k$.*

The combination of Theorems 3.1 and 3.2 prove that the best $(n, d+1, t)$-cover implies a nearly optimal number of distinct subsets, up to a factor of at most $d+1$.

3.5 Double Shamir Sharing

A useful resource for efficient honest-majority MPC protocols is a so-called "double Shamir sharing" of a random secret, where the parties are given two random polynomials of degrees d and $2d$ that share the same random secrets. Here we consider the case of packed secret sharing. Letting $\ell = d - t + 1$ be the packing parameter, we want to generate a random degree-d polynomial P_1, and another polynomial P_2 of degree-$2d$ which is random subject to $P_1(x) = P_2(x)$ for all $x \in \{0, -1, -2, \ldots, -\ell + 1\}$. It is easy to see that this task reduces to generating two independent random polynomials $P_1(X)$ of degree d and $R(X)$ of degree $2d - \ell$, then setting $P_2(X) = P_1(X) + R(X) \cdot X(X+1)(X+2)\cdots(X+\ell-1)$.

Indeed, the polynomial on the right side is a random degree-$2d$ polynomial, under the constraint that its values at the points $\{0, -1, \ldots, \ell + 1\}$ are 0. Since $P_1(x)$ and $R(x)$ are random independent polynomials, we can use the construction from the previous section in a black-box way. Specifically, we can generate $P_1(x)$ using a $(n, d+1, t)$-cover and generate $R(x)$ using an $(n, 2d-\ell+1, t)$-cover.

Theorem 3.3 (Generalized PRSS for packed double sharing). *Fix integers $d > t > 0$ and $n > 2d$ and let $\ell = d - t + 1$. A size-k' $(n, d+1, t)$-cover and a size-k'' $(n, 2d - \ell + 1, t)$-cover can be used to construct a solution for t-secure distribution of double-Sharing of degree-d and degree-$2d$ polynomials, both packing the same ℓ elements, with the following complexity measures:*

- *The number of distinct subsets (seeds) is at most $k \leq k'(d+1) + k''(2d - \ell + 1) \leq k'(2d + t + 1)$;*
- *The total subset size (storage) is $\sum_j |S_j| \leq k'(d+1)(n-d) + k''(d+t)(n - d - t + 1)$.*
- *The total number of PRF calls is $k'(d+1)(n-d) + k''(d+t)(n-d-t+1)$.*

The proof is in the full version. This construction is already strong enough to support DN-type secure computation protocols, even while packing ℓ elements in each polynomial. (Hence it can be used to compute the same circuit on ℓ different inputs at once, in a SIMD fashion.)

As an alternative to the above, we can use an $(n, d+1, t)$-cover to construct both polynomials, by increasing the number of pseudorandom elements derived from each seed. This will reduce the number of seeds stored by the parties (by some factor smaller than two), but will increase the number of pseudorandom elements that must be derived from these seeds. We provide the construction in the full version of this paper. We use a similar idea in the construction in the next section.

3.6 Beyond Double Sharing

In some applications, including the protocol that we describe in Section 4, we must generate double-Shamir-sharing of linearly correlated packed values (rather than the same values twice). While we don't know how to use the random-polynomial construction in a black-box manner to achieve this, we show here how to modify that construction in order to distribute this more general linear correlation in a t-secure manner.

This extension, however, comes with some loss of efficiency. Specifically, we need to start from covers with smaller subsets, and moreover we no longer distribute only a single random element to each subset. Fix $n > d > t > 0$ and $\ell \leq d - t$ (allowing $\ell < d - t$ is useful to mitigate the parameter loss). The goal in this section is to share two types of polynomials:

- m polynomials $R_1, \ldots, R_m$ of degree $2d$, each packing ℓ "free variables" (i.e. unconstrained) in positions $0, -1, \ldots, -\ell + 1$.
- m' additional polynomials $U_1, \ldots, U_{m'}$ of degree d, each packing ℓ constrained variables, which are set as some fixed linear combinations of the free variables.

Denote the positions where these values are packed by $L = \{0, -1, \ldots, -\ell + 1\}$, and also denote the linear correlation above by $\mathcal{L}[n, d, \ell, m, m']$. In the full version, we show the following:

Theorem 3.4 (Generalized PRSS for replicated packed secrets). *Fix integers $n \geq d > t > 0$, $\ell \leq d - t$, $m, m' > 1$. A size-k' $(n, d - \ell + 1, t)$-cover can be used to construct a solution for t-secure distribution of the linear correlation $\mathcal{L}[n, d, \ell, m, m']$ above. The complexity is at most:*

- *The number of distinct subsets (seeds) is at most $k \leq k'(d - \ell + 1)$;*
- *The total subset size (storage) is $\sum_j |S_j| \leq k'(d - \ell + 1)(n - d + \ell)$;*
- *The total number of PRF calls is at most $k(n - d + \ell)(m(d + \ell + 1) + m')$.*

Parameters. We remark that the parameters of this construction behave differently than those of the previous constructions. For the constructions from Sections 3.4 and 3.5, increasing ℓ (and d) was a double-win, not so for the current construction. Here we need to start from a $(n, d - \ell + 1, t)$-cover, so setting $\ell = d - t$ we hardly get any slackness in the size of the sets in our t-cover (they will be of size only $t+1$). To improve parameters (the cover size in particular), it is better to choose a smaller value of ℓ, thereby working with larger subsets and hence being able to find smaller covers. It is likely that setting $\ell \approx (d-t)/2$ will be a sweet spot for this construction in terms of complexity.

4 Constructions for Semi-Honest Security

In this section, we present protocols to compute a layered straight-line program over a finite field $\mathbb{F}$, that is secure in the presence of a *semi-honest* adversary who controls t parties, and with straggler-resilience. Recall that we have $n \geq 2d+1$ parties, where $d \geq t+\ell-1$.

The starting point of our constructions is the DN protocol [22], which is the fastest protocol known to this date for $n > 3$ parties. We begin in Section 4.1 with recalling the baseline DN protocol. In Section 4.2, we introduce straggler resilience and show how to adapt the DN protocol accordingly. Then in Section 4.3 we provide our solutions for improving the communication and computation requirements of the protocol.

4.1 Baseline Protocol (with $\ell = 1$)

Recall that in the DN protocol [22], the parties compute linear operations without any interaction and compute multiplication operations with small constant communication cost per party. Given shares $[\![x]\!]_d$ and $[\![y]\!]_d$, the parties compute $[\![x \cdot y]\!]_d$ in the following way. The parties prepare random sharings $[\![r]\!]_d$ and $[\![r]\!]_{2d}$ in an offline step which are consumed as follows. First, the parties locally compute $[\![x \cdot y - r]\!]_{2d} = [\![x]\!]_d \cdot [\![y]\!]_d - [\![r]\!]_{2d}$ and send their shares to P_1. Then, party P_1 computes $x \cdot y - r$ and shares the result to the parties as $[\![xy-r]\!]_d$. Finally, the parties locally compute $[\![x \cdot y]\!]_d = [\![r]\!]_d + [\![xy-r]\!]_d$.

As the random sharings can be generated non-interactively (in the way described in Section 3), the communication cost is derived from parties sending one field element to P_1 and P_1 secret sharing $xy-r$ to the parties. Note that $2d$ shares are sufficient for P_1 to reconstruct $xy-r$ (together with its own share). Also, it is possible to reduce communication in the second round by setting the shares of d parties to be 0, and having P_1 define its own share and the remaining $n-d$ parties' shares, given the value of $xy-r$ and the d zero shares. This is possible since $xy-r$ is not secret (P_1 could send it in the clear to the parties) and since $[\![xy-r]\!]_d$ is shared via a polynomial of degree d, and so $d+1$ points are sufficient to define it. Overall, we have that the communication cost per party per bilinear gate is $\frac{2d+n-d-1}{n} = 1 + \frac{d-1}{n}$ field elements. When $n > 2d+1$, it is possible to improve this by having the parties secret sharing their inputs to $2d+1$ parties who perform the computation. In this case, the communication cost per party per bilinear gate reduces to $\frac{2d+d}{n} = \frac{3d}{n}$ elements.

We denote by $\Pi^{\mathsf{base}}_{\mathrm{SH}}$ the base protocol, which thus works as follows:

Protocol $\Pi^{\mathsf{base}}_{\mathrm{SH}}$:

The parties hold a description of a layered SLP over $\mathbb{F}$. Denote by S the set of parties $P_1, \ldots, P_{2d+1}$

- **Pre-processing**: The parties call $\mathcal{F}_{\text{LinRand}}$ to obtain a pair of random sharings $\llbracket r \rrbracket_d$ and $\llbracket r \rrbracket_{2d}$ for each bilinear instruction.
- **The protocol**:
  1. *Input sharing:* for each instruction $R_j \leftarrow x_i$, party P_i run $\llbracket x_i \rrbracket_d \leftarrow \mathsf{share}(x_i)$ and sends the resulting shares to the parties in S.
  2. *Evaluating the jth bilinear instruction* $R_j \leftarrow (\sum \alpha_\omega R_\omega) \cdot (\sum \beta_\omega R_\omega)$: Let $\llbracket r \rrbracket_d, \llbracket r \rrbracket_{2d}$ be the next unused pair of random sharings. Then:
     (a) The parties in S locally compute $\llbracket x \rrbracket_d = \sum_{\omega=1}^{w} \alpha_\omega \cdot \llbracket R_\omega \rrbracket_d$ and $\llbracket y \rrbracket_d = \sum_{\omega=1}^{w} b_\omega \cdot \llbracket R_\omega \rrbracket_d$, where $\llbracket R_\omega \rrbracket_d$ denotes sharing of the ω-index memory value R_ω (stored from previous operations).
     (b) The parties in S locally compute $\llbracket xy - r \rrbracket_{2d} = \llbracket x \rrbracket_d \cdot \llbracket y \rrbracket_d - \llbracket r \rrbracket_{2d}$ and send the result to P_1.
     (c) P_1 locally reconstructs $xy - r$ and then computes a sharing $\llbracket xy - r \rrbracket_d$ such that the shares of $P_2 \dots, P_{d+1}$ are 0. Then, it sends the non-zero shares to parties $P_{d+2}, \dots, P_{2d+1}$.
     (d) The parties in S set $\llbracket z \rrbracket_d \leftarrow \llbracket r \rrbracket_d + \llbracket xy - r \rrbracket_d$, and define $\llbracket z \rrbracket_d$ as their share of the output.
  3. *Output reconstruction:* For each instruction $O_i \leftarrow R_j$, the parties in S send their shares of the value in R_j to P_i, who uses them to reconstruct the output O_i.

Security of $\Pi_{\text{SH}}^{\mathsf{base}}$ against a semi-honest adversary $\mathcal{A}$ controlling d parties follows from the fact that $\mathcal{A}$'s view consists of d random shares in the input sharing step, and masked intermediate values when performing multiplication operations.

4.2 Straggler Resilience

The classical communication model for secure multi-party computation considers parties who advance in the same pace in a fully synchronous manner. However, in real world scenarios, it is unreasonable to assume that all messages arrive at the same time. A protocol which can proceed without having to wait for all the parties' messages to arrive in each round, has thus the potential to reduce the overall latency of the execution.

We consider a model of *straggler resilience*, to account for the fact that communication channels exhibit a distribution over latency times, each of which may incur long delays with small probability. Instead of requiring parties to block and wait in every communication round until the last messages arrive, we build into the protocol design that the computation may proceed even in the absence of a small number of messages per round, which have not yet successfully been delivered. We say that a protocol that terminates successfully even when τ messages are dropped in each round, is resilient to τ stragglers. As for privacy, following the standard definition of multi-party computation [29], we consider an adversary who controls t parties and, in addition, is allowed to choose τ messages to be dropped in each round.

Definition 4.1 (Straggler resilience, semi-honest security). *Let f be an n-party functionality. We say that protocol Π computes f with t-semi-honest-security and τ-straggler-resilience if it satisfies the following properties:*

- Straggler-Robust Correctness: *Π terminates successfully (i.e. each party receives its prescribed output $f_i(\boldsymbol{x})$), even if in each communication round, τ messages, chosen adaptively by the adversary, are not delivered.*
- Semi-Honest Security with Stragglers: *For every real-world semi-honest adversary $\mathcal{A}$ controlling a set I of parties with $|I| \leq t$ and, in addition, can choose adaptively τ messages to drop in each communication round, there exists an ideal-world simulator $\mathcal{S}$ such that for every vector of inputs $\boldsymbol{x}$ it holds: $\{\mathcal{S}(I, \boldsymbol{x}_I, f_I(\boldsymbol{x}))\} \equiv \{\mathsf{view}^{\pi}_{\mathcal{A}}(\boldsymbol{x})\}$, where $\boldsymbol{x}_I$ is the inputs of the parties in I, $f_I(\boldsymbol{x})$ is the output intended to the parties in I, and $\mathsf{view}^{\pi}_{\mathcal{A}}(\boldsymbol{x})$ is $\mathcal{A}$'s view in a real execution of π.*

Remark 4.1 (Straggler resilience).

1. *Round vs. epoch.* Our protocol constructions have a very specific structure, common to concretely efficient n-party computation protocols (à la DN [22]), where execution is divided into phases, or "epochs." In each epoch, a fixed designated party sends messages to the other parties, and then receives back messages from the parties. Within such structure, a somewhat more natural notion of straggler resilience will correspond to a given number of dropped messages per *epoch* (i.e., 2 rounds). However, our notion of τ dropped messages per round is more generally applicable, while still capturing the setting of bounded number of messages dropped per epoch (in this case 2τ, for the two rounds).
2. *Message vs. node drop.* We choose to model latency behavior as embodied by failure of delivery of individual *messages*. This captures settings where delays are caused by network channels, each exhibiting some distribution of latency. This further shares similarities to the "message omission" model, where messages sent to/from affected parties may never be delivered, as considered in, e.g., [40, 46, 45].
   An alternative approach is to consider temporary *node* failures per epoch (as considered in, e.g., [47, 43, 53]). This models settings where delays are caused centrally by the node itself. On one hand, our model can be more fine-grained; on the other hand, failure of a node corresponds to failure of potentially many incoming/outgoing communication messages. We remark that achieving straggler resilience against node failures poses a challenge within protocols following a star-topology communication structure as in DN and successors since failure of the designated "central" party prevents forward progression of the protocol. Seeing as this protocol structure lies at the core of concretely efficient n-party protocols to date, it remains an interesting open direction to explore whether such node-straggler resilience notion can additionally be achieved with good concrete efficiency.

Observe that the DN protocol $\Pi^{\mathsf{base}}_{\mathsf{SH}}$ from the previous section is not resilient to any straggler. Since it chooses a set S of $2d+1$ parties *in advance* to carry-out the computation, and then the server cannot proceed without all $2d$ messages arriving to him in each multiplication, then an adversary who chooses to drop the messages of even one party in the set S will cause the execution to get stuck. Note that choosing a different set S in each step will not solve the problem, since the adversary is allowed to adaptively choose a different set in each epoch (not to mention the communication cost incurred by resharing intermediate values to the new set of parties).

Next, consider a protocol, where we let all the parties participate in the execution and send their $2d$-degree shares of $xy - r$ to P_1, who then uses the *first* $2d$ shares it receives (together with its own share) to compute $xy - r$. Then, P_1 shares $xy - r$ to the parties, with the optimization outlined above, which allows him to send shares to $n - d - 1$ parties only (d shares are always 0).

Note that now the cost is $\frac{n-1+n-d-1}{n} = 2 - \frac{d+2}{n}$ field elements sent per party. We denote by $\Pi_{\text{SH}}^{\mathsf{single}}$ a protocol that is identical to $\Pi_{\text{SH}}^{\mathsf{base}}$, with the difference that the input is shared to *all* the parties and multiplication operations are carried-out in the way described above. While the communication cost of $\Pi_{\text{SH}}^{\mathsf{single}}$ is higher than of $\Pi_{\text{SH}}^{\mathsf{base}}$, it does allow $(n - 2d - 1)$ messages in each epoch to be dropped, since P_1 needs only $2d$ shares in order to compute its message to the parties. For the input sharing and output reconstruction steps, note that $d+1$ shares suffices to compute shared secrets, and so even if $(n - 2d - 1)$ messages are dropped, there are enough shares to proceed. We thus have:

Theorem 4.1. *Let f be a n-ary functionality over a finite field $\mathbb{F}$ represented by a layered SLP, let t be a security threshold, let d be a parameter such that $d \geq t$, $n \geq 2d + 1$ and $|\mathbb{F}| > n + d + 1$. Then, Protocol $\Pi_{\text{SH}}^{\mathsf{single}}$ computes f in the $\mathcal{F}_{\text{LinRand}}$-hybrid model, with t-semi-honest-security, $(n - 1 - 2d)$-stragglers-resilience and communication of $2 - \frac{d+2}{n}$ field elements sent per party for each bilinear instruction.*

Observe that setting the d parameter gives rise to trade-offs between communication cost, stragglers-resilience and storage cost. Specifically, increasing d reduces communication and also the amount of PRSS keys needed for producing the correlated randomness (see Section 3). In contrast, keeping d small (e.g., setting $d = t$) provides more room for stragglers.

4.3 Reducing Communication and Computation

In this section, we show how to reduce communication and computation cost while still providing resilience to stragglers. This is achieved by taking the approach of packed secret sharing: encoding ℓ secrets over the same polynomial and evaluating ℓ bilinear instructions together, at the cost of a single instruction. We begin with a construction that is designed for SIMD programs, and then show how to extend our techniques to general programs.

Computing SIMD Programs A program which evaluates the same sub-program many times in parallel is called a SIMD ("same-instruction-multiple-data") straight-line program. Note that a program P which consists of ℓ copies of the same sub-program can be viewed as a program which evaluates each time *a bundle of ℓ identical instructions.* Following works in this area, our idea is to store the ℓ inputs to each bundle on the same polynomial, reducing both communication and computation by a factor of ℓ. Details can be found in the full version.

Computing General Layered Straight-Line Programs We next show how use packing to reduce cost when computing any straight-line program. In the protocol, the parties will process in each round ℓ instructions together at the cost of evaluating a single instruction. For a general-structured program this clearly raises several difficulties. Recall that an instruction in our program consists of taking a linear combination of two sets of inputs and multiply them together. The goal is to carry-out this by packing the "left" inputs on one polynomial and the "right" inputs on a second polynomial and multiply them together, to obtain a polynomial encoding the outputs of ℓ instructions. However, it is now not clear how to proceed to the next batch of ℓ instructions. In particular, when we move from one batch of instructions to the next, the outputs should be reorganized into new blocks of inputs corresponding to the ordering of the inputs in the next ℓ instructions. Moreover, it is possible that an output is used as an input to more than one instruction in the next batch. In this case, we need to ensure that the same value appears in several blocks and possibly in different positions. We call this ordering the "repetition pattern" induced by the program. To overcome this challenge, we leverage the fact that in the semi-honest multiplication protocol, party P_1 sees all outputs in the clear, masked using random values. Thus, we can ask P_1 to reshare all values according the ordering of the next batch of instructions. Moreover, to achieve free-addition, we will ask P_1 to first compute the linear combinations over the masked outputs and only then reshare it to the other parties in blocks. The parties, who receive block of masked values, will unmask these values, using correlated randomness they hold, and proceed to the multiplication operation.

In our protocol, the parties hold a sharing of two blocks of ℓ inputs: $\llbracket x_1 \cdots x_\ell \rrbracket_d$ and $\llbracket y_1 \cdots y_\ell \rrbracket_d$. As in the DN protocol, they locally multiply their shares and add shares of a random block $\llbracket r_1 \cdots r_\ell \rrbracket_{2d}$ to obtain a sharing $\llbracket (x_1 \cdot y_1 + r_1) \cdots (x_\ell \cdot y_\ell + r_\ell) \rrbracket_{2d}$. Then, the parties send their shares to P_1 who reconstructs $x_1 \cdot y_1 + r_1, \ldots, x_\ell \cdot y_\ell + r_\ell$. However, instead of sending these back to the parties, we let P_1 proceed to the next batch of instructions and compute the linear combinations of the inputs over the masked secrets. Only then P_1 shares the left block of masked inputs and right block of masked inputs to the parties, to perform the next multiplication operation. Once the shares of the blocks of masked inputs are received from P_1, the parties unmask these by adding a block of shared random secret that correspond to the repetition pattern. That is, if we have in the kth position of, say, the left input, a linear combination $(\sum_{\omega=1}^{w} a_{k,\omega} \cdot R_\omega)$ and the value in R_ω was masked using r_ω, then the parties need here a sharing $\llbracket r`_1 \cdots r`_\ell \rrbracket_d$ where $r`_k = (\sum_{\omega=1}^{w} a_{k,\omega} \cdot r_\omega)$. Fortunately, our pre-processing protocol from Section 3 can produce these types of random blocks. As before, P_1 proceed once $2d$ shares have been received, which means that, as before, the protocol is resilient to $n-1-2d$ stragglers. We stress that our trick to let P_1 compute the linear operations over the masked inputs and only then reshare it back to parties, is crucial for achieving addition for free - a property that is not trivial to achieve for non-SIMD circuits.

We formally describe our semi-honest protocol in the full version. Note that for each batch of ℓ bilinear instruction, $n-1$ parties send an element to P_1, whereas P_1 need to share the inputs of the two inputs blocks, thus sending $2(n-1-d)$ elements. Overall, per a single instruction, each party sends $\frac{n-1+2(n-1-d)}{n\cdot\ell} = \frac{3}{\ell} - \frac{2d+3}{n\cdot\ell}$ field elements, where $d \geq t+\ell-1$.

Theorem 4.2. *Let f be a n-party functionality over a finite field $\mathbb{F}$ represented by a ℓ-layered SLP, let t be a security threshold parameter and let d be a parameter such that $d \geq t+\ell-1$, $n \geq 2d+1$ and $|\mathbb{F}| > n+d+\ell+1$. Then, our protocol computes f in the $\mathcal{F}_{\text{LinRand}}$-hybrid model with t-semi-honest-security, $(n-(2d+1))$-stragglers-resilience and communication of $\frac{3}{\ell}-\frac{2d+3}{n\cdot\ell}$ field elements sent per party for each bilinear instruction.*

The proof in the full version. Observe that when $\ell \geq 3$ (i.e., packing at least 3 secrets on each polynomial), we have $\frac{3}{\ell} - \frac{2d+3}{n\cdot\ell} < 1$, which means that each party sends *less than one field element* for each bilinear instruction. When $\ell = 2$, then the cost is less than 1.5 elements sent per party. We thus obtain a protocol which provide the best of both worlds: it achieves both minimal communication and stragglers resilience. This is in contrast to $\Pi_{\text{SH}}^{\text{base}}$ which achieves minimal communication without any resilience to stragglers, and $\Pi_{\text{SH}}^{\text{single}}$ which can handle stragglers but at the cost of (at least) doubling the communication cost. We provide exact cost analysis with concrete numbers in the full version.

5 From Semi-Honest to Malicious Security

In this section, we show how to augment our protocol from the previous section to malicious security (with abort). Our goal is to achieve malicious security without increasing the amortized communication cost per instruction, and while maintaining the resilience to stragglers.

We begin by defining the meaning of security and resilience to stragglers in the presence of malicious adversaries. Note that unlike the definition with semi-honest adversaries, we no longer guarantee a successful termination of the protocol, but rather provide security with abort. The straggler-robust correctness, however, will still require that the protocol ends successfully if the parties act honestly, even if in each round τ messages, chosen by the adversary, are dropped. In addition to this requirement, we also need the protocol to be secure in the presence of an adversary who controls t parties and, in addition, can drop any τ messages in each round of communication.

Following the standard ideal-world vs. real-world paradigm of MPC [29, 14], let $\mathcal{A}$ be an adversary who chooses a set of parties before the beginning of the execution and corrupts them. We assume that the adversary is *rushing*, meaning that it first receives the messages sent by the honest parties in each round, and only then determines the corrupted parties' messages in this round. Let $\text{REAL}^{f}_{\Pi,\mathcal{A},I}(1^{\kappa}, \boldsymbol{x})$ be a random variable that consists of the view of the adversary

$\mathcal{A}$ controlling a set of parties I, and the honest parties' outputs, following an execution of Π over a vector of inputs $\boldsymbol{x}$ to compute f with security parameter κ. Similarly, we define an ideal-world execution with an ideal-world adversary $\mathcal{S}$, where $\mathcal{S}$ and the honest parties interact with a trusted party who computes f for them. We consider secure computation *with abort*, meaning that $\mathcal{S}$ is allowed to send the trusted party computing f a special command abort. Specifically, $\mathcal{S}$ can send an abort command instead of handing the corrupted parties' inputs to the trusted party (causing all parties to abort the execution), or, hand the inputs and then, after receiving the corrupted parties' outputs from the trusted party, send the abort command, and prevent them from receiving their outputs. We denote by $\text{IDEAL}_{f,\mathcal{S},I}(1^\kappa, \boldsymbol{x})$, the random variable that consists of the output of $\mathcal{S}$ and the honest parties in an ideal execution to compute f, over a vector of inputs $\boldsymbol{x}$, where $\mathcal{S}$ controls a set of parties I. The security definition states that a protocol Π securely computes f with statistical error ε, if for every real-world adversary there exists an ideal-world adversary, such that the statistical distance between the two random variables is less than ε.

Definition 5.1 (Straggler resilience, malicious security). *Let f be an n-party functionality and let $\varepsilon = \varepsilon(\kappa)$ be a statistical error bound. We say that Π computes f with t-malicious-security-with-abort and τ-straggler-resilience with statistical error ε if it satisfies the following properties:*

- Straggler-Robust Correctness: *If all parties act honestly, then Π terminates successfully (i.e. each party receives its prescribed output $f_i(\boldsymbol{x})$) even if in each communication round, τ messages, chosen adaptively by the adversary, are not delivered.*
- Security with Stragglers: *For every real-world malicious adversary $\mathcal{A}$ who controls a set of parties I with $|I| \leq t$ and, in addition, can choose adaptively any τ messages to drop in each round of communication, there exists an ideal-world simulator $\mathcal{S}$, such that for every κ and every vector of inputs $\boldsymbol{x}$ it holds that $SD\left(\text{REAL}^{f}_{\Pi,\mathcal{A},I}(1^\kappa, \boldsymbol{x}), \text{IDEAL}_{f,\mathcal{S},I}(1^\kappa, \boldsymbol{x})\right) \leq \varepsilon$ where $SD(X,Y)$ is the statistical distance between X and Y*[7]*.*

To construct a protocol that satisfies the definition, we work in two steps. First, we present a protocol to compute the program until (and not including) the output-revealing stage, that provides privacy in the presence of malicious adversaries. As we will see, maybe somewhat contrary to intuition, our semi-honest protocol from the previous section may leak private data to a malicious adversary. We thus show how to fix this without changing the communication cost or the round complexity and whilst providing the same resilience to stragglers.

Then, we add a step, before the revealing of the output, in which the parties verify the correctness of the computation, and abort with high probability if cheating took place. The properties of this step are: (i) it has sublinear communication (in the size of the program) and so the overall amortized communication

[7] Note that we prove statistical security of our protocol in a hybrid model where parties hold correlated randomness. The resulting combined protocol provides computational security when this setup is instantiated using PRSS.

cost per instruction remains the same, (ii) it requires a small constant number of rounds and so does not increase the round complexity of our protocol.

We note that although the protocol we describe only guarantees security with *selective* abort, it can be easily augmented to *unanimous* abort as required by the definition above with small constant cost, by running a single Byzantine agreement before the end of the execution. For simplicity, we omit this step from the description.

Before proceeding, we briefly describe two building blocks required by our protocol:

The $\mathcal{F}_{\text{coin}}$ ideal functionality. In our protocol, the parties will sometimes need to produce fresh random coins. The $\mathcal{F}_{\text{coin}}$ functionality, when called by the parties, hands them such coins. To compute $\mathcal{F}_{\text{coin}}$ with abort, the parties can simply generate a random sharing $[\![r]\!]_d$ and open it. In the honest majority setting, there is nothing the adversary can do here beyond causing an abort. We note that to generate any number of coins with constant communication cost, it suffices to call $\mathcal{F}_{\text{coin}}$ once to obtain a seed, and expand it to many pseudo-random coins.

Consistency check. To check that m sharings $\{[\![x_{j,1} \cdots x_{j,\ell}]\!]_d\}_{j=1}^{m}$ are consistent, we use the well-known method of taking a random linear combination of these sharings, mask the result by adding a random sharing $[\![r_1 \cdots r_\ell]\!]_d$, and open it. For the random linear combination, the parties call $\mathcal{F}_{\text{coin}}$ to obtain the random coefficients.

5.1 Privacy in the Presence of Malicious Adversaries

In this section, we show how to compute a straight-line program with privacy in the presence of a malicious adversary. We begin by showing that DN-style semi-honest protocols which we consider in this work, may leak private information to a malicious adversary in the strong honest majority setting. Recall that in the semi-honest protocol, to carry-out a multiplication between shared inputs $[\![x]\!]_d$ and $[\![y]\!]_d$, the parties send $[\![x \cdot y - r]\!]_{2d}$ to P_1, who reconstruct $x \cdot y - r$ and shares it as $[\![x \cdot y - r]\!]_d$ to the parties. Then, the parties compute $[\![x \cdot y]\!]_d = [\![x \cdot y - r]\!]_d + [\![r]\!]_d$ and obtain a sharing of the output.

The "double-dipping" attack [35]. We now describe an attack that can be carried out by a malicious P_1, when $n > 2d + 1$. This attack was shown in [35] for the setting of $d < n/3$ and works over two multiplication gates/instructions as follows. Assume that the parties multiply $[\![x]\!]_d$ with $[\![y]\!]_d$. Thus, after receiving the masked shares from the parties, P_1 reconstructs $xy - r$ and computes a random sharing $[\![x \cdot y - r]\!]_d$. Then, P_1 sends the correct shares to all parties except for P_n, to whom it adds 1 to the intended share. Thus, all the parties, except for P_n, can compute the correct share of $x \cdot y$ by adding $[\![r]\!]_d$. Denote the share of $x \cdot y$ held by P_i by α_i. This means that P_n will hold $\alpha_n + 1$. Next, assume that the parties proceed to the next multiplication, where they need to multiply $[\![xy]\!]_d$ with $[\![z]\!]_d$, and denote the share of z held by P_i by z_i. Note that once P_1 receives $2d$ shares, it can not only reconstruct $xyz - r'$, where r' is the random masking for this multiplication, but also can compute the remaining $n-1-2d$ shares that should be sent. In particular, after receiving shares from any subset of $2d$ parties that does not contain P_n, it can compute the correct share that should be sent by P_n, i.e., $\alpha_n \cdot z_n - r'_n$, where r'_n is P_n's share of r'. However, P_n will send the share $(\alpha_n + 1) \cdot z_n - r'_n$, which means that P_1 can compute $(\alpha_n \cdot z_n - r'_n) - ((\alpha_n + 1) \cdot z_n - r'_n) = z_n$, obtaining the secret share z_n of P_n.

Previous solutions. The main reason for the above attack is that in the strong honest majority setting, there is redundancy in the masking. Indeed, the solution suggested in [35] is to use as masking the sharing $[\![r]\!]_{n-1}$, which means that $x \cdot y - r$ can be reconstructed only given the shares of all parties. A different solution was given in [26], where a consistency check was carried-out between each two layers of the program. This prevents the above attack, since by sending an incorrect share to P_n, the resulting sharing of $x \cdot y$ becomes inconsistent. Thus, a consistency check will detect this type of cheating and prevents P_1 from proceeding with the attack to the multiplication in the next layer. However, these solutions are not sufficient in our case, since either they require all parties to participate, preventing any resilience to stragglers, or, double the round complexity of the protocol.

A new solution with straggler resilience. We thus need a new solution that achieves privacy, while allowing P_1 to proceed without requiring all parties' shares of $x \cdot y - r$. Our idea is to have a *different independent masking value for each subset of* $2d+1$ *parties.* In particular, for each subset T of $2d+1$ parties, we want the parties to hold a pair $([\![r_T]\!]_d, [\![r_T]\!]_{2d})$ which can be used in the multiplication protocol. This however raises a question. If each subset of parties have a different masking, then which masking share should a party use when it sends its message to P_1? To overcome this, we add an additional constraint: the parties should hold a pair $([\![r_T]\!]_d, [\![r_T]\!]_{2d})$ for each subset T under the constraint that each P_i's share in $[\![r_T]\!]_{2d}$ will be *identical* for all subsets. If this holds, then only one possible message exists for each P_i to send to P_1 (i.e., $x_i \cdot y_i - r_i$ where r_i is the random share used by P_i as a mask). We will see later how to generate such correlated randomness in an efficient way (without requiring the parties to store $\binom{n}{2d+1}$ different polynomials). Assuming the parties have a way to generate such random sharings, our private protocol to multiply $[\![x]\!]_d$ and $[\![y]\!]_d$ is:

$\underline{\Pi^{\mathsf{priv}}_{\mathsf{mult}}:}$

- **Inputs**: Each P_i holds two inputs shares x_i, y_i and a random share r_i.
  For each subset $T \subset \{P_1, \ldots, P_n\}$ such that $|T| = 2d+1$, the parties hold a sharing $[\![r_T]\!]_d$, where $r_T = \sum_{j|P_j \in T} \lambda_j \cdot r_j$, with λ_j being the corresponding Lagrange coefficient for the $2d$-polynomial q_T defined such that $q_T(j) = r_j$, for each j for which $P_j \in T$.
- **The protocol:**
  1. Each party P_i locally computes $e_i = x_i \cdot y_i - r_i$ and sends it to P_1.
  2. Let $e_{i_1}, \ldots, e_{i_{2d}}$ be the first $2d$ messages received by P_1 and let T be a subset of parties defined as $T = \{P_1, P_{i_1}, \ldots, P_{i_{2d}}\}$. Then, P_1 view $e_1, e_{i_1}, \ldots, e_{i_{2d}}$

as points on a polynomial p of $2d$-degree such that $p(1) = e_1$ $\textsf{and}$ $\forall j \in [2d] : p(i_j) = e_{i_j}$ and uses them to compute (via Lagrange interpolation) the value $e_0 = p(0)$.

3. P_1 chooses a new random sharing $[\![e_0]\!]_d$, under the constraint that d shares equal to 0, and sends each party P_i, with a non-zero share, its share. In addition, it sends T to all parties.
4. The parties locally compute $[\![x \cdot y]\!]_d = [\![e_0]\!]_d + [\![r_T]\!]_d$.

It is easy to see that if the parties follow the protocol, then they will obtain $[\![x \cdot y]\!]_d$. Privacy is achieved since now there is no redundancy in the secret sharing of the masking random element, and each random share held by each party is independent from the other parties' random shares. We show this formally in the full version.

Efficient generation of the correlated randomness. Recall that our protocol requires that for each multiplication, each P_i will hold a random independent r_i and a sharing $[\![r_T]\!]_d$ for each subset of parties T of size $2d+1$, such that $r_T = \sum_{j|P_j \in T} \lambda_j \cdot r_j$. A simple way to achieve this, is to let each P_i choose a random r_i and share it to the other parties as $[\![r_i]\!]_d$. Upon holding $[\![r_i]\!]_d$ for each $i \in [n]$, the parties can locally compute $[\![r_T]\!]_d = \sum_{j|P_j \in T} \lambda_j \cdot [\![r_j]\!]_d$ for each subset T of size $2d+1$. We note that in order to save cost, the parties can defer the last step of computing $[\![r_T]\!]_d$ until they receive the subset T from P_1. This is significant since now the parties need to compute just a single sharing of degree d and not $\binom{n}{2d+1}$.

To generate any number of such correlated randomness without any interaction but a short setup step, each party P_i can distribute a set of seeds to the other parties. As explained in Section 3, it is possible to non-interactively generate any number of Shamir's secret sharings $[\![r_i]\!]_d$ from these seeds and then continue as above. Note that since P_i knows all seeds, it can locally compute r_i and use it as its mask in the multiplication operation as required.

In the full version of this paper we show how to extend the solution when multiple secrets are packed together.

5.2 Verifying Correctness of the Computation

In the previous section, we showed how to prevent leakage of private data during the computation of the program. However, nothing prevents a malicious adversary from cheating by sending false messages, causing the output to be incorrect. To achieve correctness, we add a step to our protocol, before the output is revealed, where the parties verify the correctness of the computation, and abort if cheating is detected. This additional step satisfies two desired properties: (i) it is a short constant-round protocol; (ii) it has *sublinear communication* in the size of the program, which means that *amortized* over the program, the communication cost remains the same.

We define the ideal functionality $\mathcal{F}_{\textsf{vrfy}}$ to verify that multiplications were carried out correctly. $\mathcal{F}_{\textsf{vrfy}}$ receives from the honest parties their shares of all inputs, the inputs to multiplications operations and all outputs of the program. Then, it reconstructs the secrets and check for each value, that it is correct *given* the values held by the parties as inputs for the multiplications that precede it. We stress that it suffices for only the honest parties to send their shares, since they fully define the secrets (as we will see, a consistency check is carried out before calling $\mathcal{F}_{\mathsf{vrfy}}$ in our main protocol and so we are guaranteed at this stage that all sharings are consistent).

The formal description appears in the full version of the paper. We show how to realize $\mathcal{F}_{\mathsf{vrfy}}$ using distributed zero-knowledge proofs from [8], adapted to our setting, in the full version.

5.3 Putting It All Together - The Main Protocol

We are now ready the present our main protocol with security against malicious adversaries. The protocol works by having the parties run the private protocol to compute the program, and then, before revealing the output, call the ideal functionality $\mathcal{F}_{\mathsf{vrfy}}$ to verify that the sharings they obtained throughout the execution, are correct. Since $\mathcal{F}_{\mathsf{vrfy}}$ requires the sharings it receives to be consistent, then the parties run a batch consistency check before calling $\mathcal{F}_{\mathsf{vrfy}}$.

Stragglers Resilience. We show what resilience our protocol guarantees:

- *Input sharing step*: In this step, we require the parties to send a masked input $\hat{x}_i = x_i + r$ to all parties and not only to P_1. Looking on an epoch that consists of parties sending their masked input to the other parties, and then sending messages to P_1 in the first layer of bi-linear instructions, it is easy to see that even if $n - (2d + 1)$ messages are lost, party P_1 will receive $2d$ messages and will be able to proceed to the next epoch.
- *Private computation of the program*: Our new protocol in Section 5.1 can handle $n - (2d + 1)$ dropped messages in each epoch.
- *Verification step*: A subtle issue that arises here is the effect of stragglers existence in the private protocol, on the consistency check and $\mathcal{F}_{\mathsf{vrfy}}$. Specifically, if different subset of parties participate in each epoch, then the sharings used in the consistency check and $\mathcal{F}_{\mathsf{vrfy}}$ are held by different subset of parties, which seems problematic. Nevertheless, we observe that the number of such subsets is bounded by the depth of the program. Hence, we have three possible solutions. If the depth of the program is low, then the parties can run these two steps for each subset separately (recall that each such subset is of size $2d + 1$ and so an honest majority required by the protocols is guaranteed). Since the cost in these final steps is anyway low and sublinear in the size of the program, we can afford running them several times. If the depth is larger than the number of possible subsets $\binom{n}{\tau}$ (with τ being the number of stragglers), then we can simply go over all possible subsets. Alternatively, if the program is very deep, then one can simply assume that all messages that were delayed during the computation, arrive by the time the parties reach the final steps. While this seems as a slight weakening of our stragglers-resilience model, note that even with this assumption, our protocol has a huge advantage over protocols with no resilience to stragglers, where the parties need to wait for all messages

to arrive when computing *each layer*, and not only at the end of the entire computation.

  Note that in the former solution we need to assume that no messages are dropped inside this step, since in each subset of $2d+1$ parties, if a message is lost, we might lose the honest majority and hence the security guarantees. Since this step is a short constant-round protocol, this seems as a mild assumption.

- *Output Reconstruction*: If $2d+1$ shares arrive to each party, then at least $d+1$ shares are sent by honest parties and so are correct. This implies that the party can either reconstruct its correct output or abort if cheating took place. Thus, this step can also withstand $n-(2d+1)$ stragglers.

The formal description appears in the full version of the paper. We thus obtain a maliciously-secured protocol, with the same (amortized) communication cost and same stragglers resilience as for semi-honest adversaries (with a small caveat for the short verification step). This is summarized in the following Theorem (the proof can be found in the full version):

Theorem 5.1. *Let $\mathbb{F}$ be a finite field, let f be a n-party functionality represented by a ℓ-layered straight-line program over $\mathbb{F}$ with S bilinear instructions, let t be a security threshold parameter and let d be a parameter such that $d \geq t+\ell-1$, $n \geq 2d+1$ and $|\mathbb{F}| > n+d+\ell+1$. Then, our protocol computes f in the ($\mathcal{F}_{\mathrm{LinRand}}$,$\mathcal{F}_{\mathrm{coin}}$,$\mathcal{F}_{\mathsf{vrfy}}$)-hybrid model with t-malicious-security-with-abort, $(n-(2d+1))$-stragglers-resilience, with statistical error $\frac{1}{|\mathbb{F}|}$, and communication cost of $\left(\frac{3}{\ell}-\frac{2d+3}{n\cdot\ell}\right)S+o(S)$ field elements sent per party.*

The protocol has statistical error of $\frac{1}{|\mathbb{F}|}$ due to the consistency check that may fail. For small fields the error can be reduced by repeating the check with independent randomness.

Acknowledgements. We thank Tuvi Etzion for helpful pointers to the literature on covering designs. E. Boyle supported by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC Project HSS (852952). N. Gilboa supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai supported by ERC Project NTSC (742754), NSF-BSF grant 2015782, BSF grant 2018393, and ISF grant 2774/20. A. Nof supported by ERC Project NTSC (742754).

References

1. Covering Designs. https://www.dmgordon.org/cover//.
2. N. Alon, M. Merritt, O. Reingold, G. Taubenfeld, and R.N. Wright. Tight bounds for shared memory systems accessed by byzantine processes. *Distributed Computing*, 2005.
3. S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. Secure MPC: laziness leads to GOD. In *ASIACRYPT*, 2020.
4. J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky. How to withstand mobile virus attacks, revisited. In *ACM PODC*, 2014.

5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *ACM STOC*, 1988.
6. R. Bendlin and I. Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, 2010.
7. K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS*, 2017.
8. D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO*, 2019. Full version: ePrint report 2019/188.
9. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.
10. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *FOCS*, 2020.
11. E. Boyle, N. Gilboa, Y. Ishai, and A. Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In *ACM CCS*, 2019.
12. E. Boyle, N. Gilboa, Y. Ishai, and A. Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In *ASIACRYPT*, 2020.
13. Z. Brakerski, N. Chandran, V. Goyal, A. Jain, A. Sahai, and G. Segev. Hierarchical functional encryption. In *ITCS*, 2017.
14. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
15. R. Canetti and S. Goldwasser. An efficient *Threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*, 1999.
16. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *ACM STOC*, 1988.
17. A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. In *CRYPTO*, 2021.
18. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, 2005.
19. I. Damgård and Y. Ishai. Scalable secure multiparty computation. In *CRYPTO*, 2006.
20. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, 2010.
21. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, 2008.
22. I. Damgård and J. Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, 2007.
23. I. Damgård and R. Thorbek. Non-interactive proofs for integer multiplication. In *EUROCRYPT*, 2007.
24. M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *ACM STOC*, 1992.
25. Z. Füredi. Turán type problems. *Surveys in combinatorics*, 166:253–300, 1991.
26. J. Furukawa and Y. Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. In *ACM CCS*, 2019.
27. G.Beck, A. Goel, A. Jain, and G. Kaptchuk. Order-c secure multiparty computation for highly repetitive circuits. In *EUROCRYPT*, 2021.

28. N. Gilboa and Y. Ishai. Compressing cryptographic resources. In *CRYPTO*, 1999.
29. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
30. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM STOC*, 1987.
31. D. Gordon, S. Ranellucci, and X. Wang. Secure computation with low communication from cross-checking. In *ASIACRYPT*, 2018.
32. D. M. Gordon and D. R. Stinson. Coverings. In *Handbook of Combinatorial Designs*, pages 391–398. 2006.
33. S. D. Gordon, D. Starin, and A. Yerukhimovich. The more the merrier: Reducing the cost of large scale MPC. In *EUROCRYPT*, 2021.
34. V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song. ATLAS: efficient and scalable MPC in the honest majority setting. In *CRYPTO*, 2021.
35. V. Goyal, Y. Liu, and Y. Song. Communication-efficient unconditional MPC with guaranteed output delivery. In *CRYPTO*, 2019.
36. V. Goyal, A. Polychroniadou, and Y. Song. Unconditional communication-efficient MPC via hall's marriage theorem. In *CRYPTO*, 2021.
37. V. Goyal, Y. Song, and C. Zhu. Guaranteed output delivery comes free in honest majority MPC. In *CRYPTO*, 2020.
38. Y. Guo, R. Pass, and E. Shi. Synchronous, with a chance of partition tolerance. In *CRYPTO*, 2019.
39. Huaxiong H. Wang and J. Pieprzyk. Shared generation of pseudo-random functions with cumulative maps. In *CT-RSA*, 2003.
40. V. Hadzilacos. Issues of fault tolerance in concurrent computations (databases, reliability, transactions, agreement protocols, distributed computing). *PhD thesis*, 1985.
41. M. Hirt and M. Mularczyk. Efficient MPC with a mixed adversary. In *Information-Theoretic Cryptography ITC*, 2020.
42. I. Keidar and A. Shraer. How to choose a timing model. *IEEE Trans. Parallel Distrib. Syst.*, 19, 2008.
43. C. Y. Koo. Secure computation with partial message loss. In *TCC*, 2006.
44. S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. In *CRYPTO*, 1995.
45. P. Raipin Parvédy and M. Raynal. Uniform agreement despite process omission failures. In *International Parallel and Distributed Processing Symposium (IPDPS*.
46. K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Software Eng.*, 12, 1986.
47. M. Raynal. Consensus in synchronous systems: A concise guided tour. In *Symposium on Dependable Computing (PRDC)*, 2002.
48. A. Shamir. How to share a secret. *Commun. ACM*, 1979.
49. A. Sidorenko. What we know and what we do not know about turán numbers. *Graphs and Combinatorics*, 11(2):179–199, 1995.
50. P. Turán. On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.
51. P. Turán. Research problems. *Közl MTA Mat. Kutató Int*, 6:417–423, 1961.
52. A. Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.
53. V. Zikas, S. Hauser, and U. Maurer. Realistic failures in secure multi-party computation. In *TCC*, 2009.