# The Price of Active Security in Cryptographic Protocols

Carmit Hazay[1], Muthuramakrishnan Venkitasubramaniam[2], and Mor Weiss[3]

[1] Bar-Ilan University
[2] University of Rochester
[3] IDC Herzliya

**Abstract.** We construct the first actively-secure Multi-Party Computation (MPC) protocols with an *arbitrary* number of parties in the dishonest majority setting, for an *arbitrary* field $\mathbb{F}$ with *constant communication overhead* over the "passive-GMW" protocol (Goldreich, Micali and Wigderson, STOC '87). Our protocols rely on passive implementations of Oblivious Transfer (OT) in the boolean setting and Oblivious Linear function Evaluation (OLE) in the arithmetic setting. Previously, such protocols were only known over sufficiently large fields (Genkin et al. STOC '14) or a constant number of parties (Ishai et al. CRYPTO '08).
Conceptually, our protocols are obtained via a new compiler from a passively-secure protocol for a distributed multiplication functionality $\mathcal{F}_{\mathrm{MULT}}$, to an actively-secure protocol for general functionalities. Roughly, $\mathcal{F}_{\mathrm{MULT}}$ is parameterized by a linear-secret sharing scheme $\mathcal{S}$, where it takes $\mathcal{S}$-shares of two secrets and returns $\mathcal{S}$-shares of their product.
We show that our compilation is concretely efficient for sufficiently large fields, resulting in an overhead of 2 when securely computing natural circuits. Our compiler has two additional benefits: (1) it can rely on *any* passive implementation of $\mathcal{F}_{\mathrm{MULT}}$, which, besides the standard implementation based on OT (for boolean) and OLE (for arithmetic) allows us to rely on implementations based on threshold cryptosystems (Cramer et al. Eurocrypt '01); and (2) it can rely on weaker-than-passive (i.e., imperfect/leaky) implementations, which in some parameter regimes yield actively-secure protocols with overhead less than 2.
Instantiating this compiler with an "honest-majority" implementations of $\mathcal{F}_{\mathrm{MULT}}$, we obtain the first honest-majority protocol with optimal corruption threshold for boolean circuits with constant communication overhead over the best passive protocol (Damgård and Nielsen, CRYPTO '07).

## 1 Introduction

The problem of Secure Multi-party Computation (MPC) considers a set of parties with private inputs that wish to jointly compute a function of their inputs while simultaneously preserving *correctness* of the outputs, and guaranteeing *privacy* of the inputs, i.e., nothing but the output is revealed. These properties are required to hold in the presence of an adversary that controls a subset of the parties, and attacks the protocol in an attempt to breach its security, e.g., learn more than it should about the honest parties' inputs.

Secure computation was first defined and explored in the mid 80s [Yao86, CCD87, GMW87, BGW88], and has been the focus of intensive study ever since. In the first two decades, research focused mainly on theoretical foundations, establishing the boundaries of feasibility and complexity. More recently, the focus has shifted to making MPC efficient and reducing its overhead over insecure implementations, both in terms of asymptotic and concrete efficiency (See [LP07, IPS08, IPS09, DPSZ12, WRK17a, WRK17b, HSS17, GLS19], and references therein.)

A basic classification in MPC considers protocols in which security is guaranteed with: (1) an *honest majority*, namely when the adversary corrupts a minority of the participants; or (2) a *dishonest majority*, where the adversary can corrupt arbitrarily many parties. The second category, which captures two-party protocols as a special case, has the advantage that any single party need not trust anyone but itself. Designing protocols from the second category is significantly more challenging, and they can only guarantee computational security, i.e., against computationally-bounded adversaries. On the other hand, the first category admits conceptually simpler solutions with statistical (or even perfect) security, namely against computationally-unbounded adversaries.

An orthogonal classification of MPC protocols is based on the adversarial behavior: (1) *passive* adversaries that follow the protocol's instructions but try to learn more than the prescribed information; and (2) *active* adversaries that may arbitrarily deviate from the protocol. A common paradigm in MPC is to design first a passively-secure protocol, and then compile it into an actively-secure one.

Hence, an important efficiency metric for MPC protocols is the *overhead* of actively-secure protocols over (the best) passively-secure ones. A primary goal in MPC today is to reduce this overhead, and specifically to design actively-secure protocols with *constant overhead* over *state-of-the-art passively-secure* protocols. That is, to design protocols whose communication and computation overheads grow only by a constant factor compared to the underlying passive protocols.

This work focuses on one of the most challenging MPC settings: *active security with an arbitrary number of parties.* Ideally, we would like the price of achieving active security to be minimal compared to the passively-secure counterparts.

The past decade has seen tremendous progress in the design of concretely-efficient actively-secure protocols for arbitrary functions, specified as boolean or arithmetic circuits, in either the two-party [ST04, LP07, KS08, NO09, LP12, NNOB12, SS13, HKK$^+$14, ZRE15, RR16, LR15, GLNP15, WMK17, WRK17a, HIV17], or the multi-party setting with an arbitrary number of parties [IPS08, DPSZ12, DKL$^+$13, LPSY15, WRK17b, HSS17, KPR18]. See Section 1.2 below for more details.

Despite this impressive progress there still remains important gaps between what is achievable with passive and active security. Indeed, no protocols for boolean computations with an arbitrary number of parties and constant communication overhead (even asymptotically) are known, both in the honest and the dishonest majority settings. For arithmetic computations with an arbitrary number of parties and over sufficiently large fields, the best concrete overhead (of 12x [GIP$^+$14]) still seems large. In the honest majority setting an overhead of 2 has been achieved only for large fields [CGH$^+$18].

Given this state of affairs, in this work we set out to answer the following fundamental open problem:

*Can actively-secure protocols over an arbitrary field match the complexity of passively-secure protocols, in the dishonest and honest majority settings, with an arbitrary number of parties?*

We resolve this open problem in terms of communication complexity in the affirmative, designing an asymptotically-efficient *actively-secure* protocol for *boolean* circuits (as well as arithmetic circuits over any field) in both the *honest majority* and *dishonest majority* settings, with *constant communication overhead* over the (best known) passively-secure counterparts.

We note that constant-overhead protocols are known based on general zero-knowledge proofs [GMW87], but these solutions rely on "heavy" tools and are *practically inefficient*. Instead, we focus on designing protocols that make *black-box* use of simpler (and lightweight) primitives such as One-Way Functions (OWFs), and parallel Oblivious-Transfer (OT) or parallel Oblivious Linear function Evaluation (OLE) in the boolean and arithmetic settings (resp.). Relying on OTs/OLEs is, in a sense, necessary since these are special cases of secure computation in their respective settings. Moreover, since our protocols make black-box use of these primitives, they will benefit from future improvements in the costs of OT/OLE implementations, which have been steadily decreasing.

Moreover, to frame a clean theoretical question, we focus on designing modular protocols in which the (relatively) computationally-expensive "cryptographic" component is separated from the rest of the protocol, and abstracted as an ideal functionality. Specifically, the "cryptographic" abstraction we consider in this work is a (constant-round) parallel protocol for computing distributed multiplication. Relying on a general multiplication functionality instead of OT/OLE allows us to simultaneously capture many settings of interest (boolean/arithmetic computations, two/multi-party, honest/dishonest majority) in a unified way. More specifically, we abstract distributed multiplication as an $\mathcal{F}_{\text{MULT}}$ functionality that is parameterized by a secret sharing scheme $\mathcal{S}$ over some field $\mathbb{F}$, takes $\mathcal{S}$-shares of two secrets, and produces $\mathcal{S}$-shares of their product. It is easy to see that one can use a general reduction from OT (resp. OLE) to a random instance $\mathcal{F}_{\text{RMULT}}$ of $\mathcal{F}_{\text{MULT}}$ (which generates additive shares of random multiplication triples in the sense of Beaver's triples [Bea91]) for boolean (resp. arithmetic) computations. In the multi-party setting, one can also realize $\mathcal{F}_{\text{MULT}}$ using more general protocols based on threshold additively-homomorphic encryption schemes [CDN01].

Given the previous discussion, we can rephrase our motivating question:

*Can actively-secure protocols over an arbitrary field match the complexity of passively-secure implementations of $\mathcal{F}_{\text{MULT}}$, in the dishonest and honest majority settings, with an arbitrary number of parties?*

## 1.1 Our Results – A New Framework

In this work we answer the open problem stated above with respect to communication complexity on the affirmative, introducing the first actively-secure protocol with constant communication overhead over passive GMW [GMW87], for any number of parties and over any field, in the $\mathcal{F}_{\text{MULT}}$-hybrid model.

We obtain our result via a new compiler which transforms a passively-secure protocol for $\mathcal{F}_{\text{MULT}}$ into an actively-secure protocol for arbitrary functionalities, while inheriting the setting of the $\mathcal{F}_{\text{MULT}}$ protocol (i.e., boolean/arithmetic, two/multi-party, and honest/dishonest majority). Specifically, the compiler is described in the $\mathcal{F}_{\text{MULT}}$-hybrid model, and using different instantiations of $\mathcal{F}_{\text{MULT}}$ we obtain actively-secure protocols with constant communication overhead in the boolean and arithmetic, two-party and multi-party, and honest and dishonest majority settings. Moreover, the overhead of our protocols is 2 for large fields and "typical" circuits (i.e., that have sufficiently many parallel multiplication gates; for our asymptotic result, it suffices for this width to be $\Omega(s)$, where $s$ is a statistical security parameter).

Working in the $\mathcal{F}_{\text{MULT}}$-hybrid model allows us to preserve a clear separation between the "passive" (alternatively, cryptographic) components of our protocol, namely the implementation of $\mathcal{F}_{\text{MULT}}$, which relies on cryptographic assumptions; and the "correctness-enforcing" (alternatively, non-cryptographic) components which involve tools from the literature of honest-majority protocols, employing consistency tests to enforce honest behavior. Besides scalability (and reduced communication complexity), we believe our approach is simple and modular.

Our compiler improves over the state-of-the-art in several settings; see Table 1 for a summary, and Section 6 for a detailed discussion.

| **Corruption Threshold** | **Number of Parties** | **Field Size** | **Hybrid Model** | **Asymptotic Overhead** | **Best Passive** | **Theorem Number** |
|---|---|---|---|---|---|---|
| $t < n$ | Arbitrary | $O(1)$ | OT | Constant | [GMW87] | Theorem 3 |
| $t < n$ | Arbitrary | Arbitrary | OLE | Constant* | [GMW87] | Theorem 5 |
| $t < n/2$ ** | Arbitrary | Arbitrary | — | Constant | [BGW88] | Theorem 6 |

Table 1: Asymptotic communication overheads of our results in both the dishonest and honest majority settings for boolean and arithmetic computations. The "best passive" column refers to the passively-secure protocol over which the overhead is computed. The "theorem number" column specifies the theorem which implies the corresponding result.
* Concretely, this constant is 2 for moderately wide circuits.
** We note that though in the honest majority setting guaranteed output delivery is achievable, our protocol only guarantees security with abort.

**New protocols in the dishonest majority setting.** Our complier exhibits the most substantial improvements in the dishonest majority setting, yielding the *first* constant-overhead actively-secure protocol with a dishonest majority over an arbitrary number of parties for boolean circuits. The concrete constants of our compiler are yet unknown since they depend on the concrete efficiency of Algebraic Geometric (AG) secret sharing schemes over constant-size fields [CC06]. The result is summarized in the following informal theorem; see Theorem 3 for the formal statement.

**Theorem 1 (Informal)** *Any $m$-party function $f$ over a constant-size field (resp., arbitrary size field) can be securely realized by an $O(d)$-round protocol in the OT-hybrid (resp., OLE-hybrid) model against an active adversary corrupting an arbitrary number of parties with total communication $O(m^2 |\mathrm{C}|) + \mathsf{poly}(\kappa, d, m)$ field elements, where $\mathrm{C}$ is a depth-$d$ circuit for $f$, and $\kappa$ is a computational security parameter.*

For arithmetic computations, we can concretely analyze the constants introduced by our compiler, and show that they can be as small as 2 for moderately wide circuits and sufficiently large fields. This improves over [GIP+14] in two aspects. First, their work requires at least 12 invocations of an active implementation of $\mathcal{F}_{\text{MULT}}$, while ours requires only two invocation of a passive implementation. This allows us to instantiate our compiler with passive implementations of $\mathcal{F}_{\text{MULT}}$ based on threshold additively homomorphic encryption schemes [CDN01, BDOZ11]. Second, their result is only useful for computations over sufficiently large fields (where the statistical error $O\left(|\mathrm{C}| / |\mathbb{F}|\right)$ is small), whereas our result applies to fields of arbitrary size.

Building on the recent result of Hazay et al. [HIMV19], we can extend our compiler to rely on a weaker-than-passive (e.g., imperfect or leaky) implementation of $\mathcal{F}_{\text{MULT}}$. Consequently $\mathcal{F}_{\text{MULT}}$ can be instantiated with lattice-based protocols with "aggressive" (weaker) parameters, yielding actively-secure compiled protocols whose communication cost almost matches that of the best passive protocols, namely, essentially achieving active security at the cost of passive!

Additionally, we achieve an interesting corollary in the constant-round regime for boolean computations. By viewing distributed garbling [BMR90] as an arithmetic functionality over $\mathbb{GF}(2^\kappa)$, we can instantiate our compiler for arithmetic circuits to achieve constant-overhead over that passive variant of [BMR90] instantiated with $\mathcal{F}_{\text{MULT}}$ over $\mathbb{GF}(2^\kappa)$. See the full version [HVW19] for details.

We believe our protocols can also be made to tolerate adaptive corruptions by instantiating the underlying cryptographic primitives (namely, $\mathcal{F}_{\text{MULT}}$ and $\mathcal{F}_{\text{COM}}$) with their adaptively-secure counterparts, and leave this to future work.

**New protocols in the honest majority setting.** In the honest majority regime for $t < n/2$, our compiler gives an actively-secure protocol for boolean circuits with constant overhead over a variant of passive-BGW [BGW88] that is instantiated using AG secret sharing schemes. This result improves over the recent protocol by Chida et al. [CGH+18], which only achieves constant overhead for large fields (introducing an extra statistical security parameter $s$ for small fields with an overhead of $s/\log_2(|\mathbb{F}|)$), and over Ishai et al. [IKP+16] who achieve constant-overhead for arbitrary fields, but only for few parties. We note that [DI06] achieves constant-rate secure protocols, but only for suboptimal corruption thresholds. For boolean computation with an arbitrary number of parties and optimal threshold, the best protocols are due to Genkin et al. [GIW16] and achieve a $\mathsf{poly}\log(|\mathrm{C}|, s)$ overhead, where $|\mathrm{C}|$ is the circuit size.

## 1.2 Related Work

We give a brief overview of recent efficient protocols, summarized in Table 2.

**The state-of-the-art: boolean multi-party setting.** For boolean circuits, secure protocol against a dishonest majority with an (asymptotic) constant overhead over passively-secure protocols, was achieved for constant number of parties by Ishai, Prabhakaran and Sahai [IPS08] (referred to as the "IPS-compiler"). Their protocol operates in the OT-hybrid model, achieving constant overhead over passive-GMW. It also achieves constant rate, namely the communication complexity of evaluating a circuit $\mathrm{C}$ is $O\left(|\mathrm{C}|\right) + \mathsf{poly}\left(\log|\mathrm{C}|, d, m, \kappa\right)$, where $d, m, \kappa$ are the depth of $\mathrm{C}$, the number of parties, and a security parameter, respectively. For an arbitrary number of parties, the protocol of Genkin et al. [GIW16] obtains $\mathsf{poly}\log\left(|\mathrm{C}|, s\right)$ overhead over passive-GMW, where $s$ is a statistical security parameter. This result is obtained by converting a boolean circuit $\mathrm{C}$ into a functionally-equivalent randomized circuit $\mathrm{C}'$ that is immune against so called "additive attacks", and evaluating $\mathrm{C}'$ using the semi-honest protocol of [GMW87]. (This technique was originally introduced by [GIP$^+$14], but was essentially only useful over large fields, see discussion below.)

**The state-of-the-art: arithmetic multi-party setting.** In the arithmetic setting in which the computation is performed over an arbitrary field $\mathbb{F}$, Genkin et al. [GIP$^+$14] designed MPC protocols in the OLE-hybrid model, with a statistical error of $O(|\mathrm{C}|/\mathbb{F})$, and constant communication overhead compared to an algebraic variant of passive-GMW [GMW87], for sufficiently large fields $\mathbb{F}$. As described above, their result is obtained by converting a circuit $\mathrm{C}$ over some field $\mathbb{F}$ into its additively-secure variant $\mathrm{C}'$, and evaluating $\mathrm{C}'$ using passive-GMW and actively secure implementation of OLE. In practice, the constant in the communication overhead of their protocol is 12, and moreover their protocol is only useful for circuits over large fields (for which $O(|\mathrm{C}|/\mathbb{F})$ is sufficiently small). For arbitrary fields, the work of Döttling et al. [DGN$^+$17] give an actively secure protocol where the overhead is 22 invocations of an actively secure implementation of $\mathcal{F}_{\text{MULT}}$ per multiplication gate of the circuit. A practical implementation for arbitrary number of parties was given in [KPR18] based on "tailor-made" zero-knowledge proofs to achieve active security.

We note that in the *honest majority* setting, the recent work by Chida et al. [CGH$^+$18] presents a new actively-secure protocol for arithmetic circuits that obtains overhead 2 over passive protocols for sufficiently large fields. Similar to our protocol, their protocol is in the $\mathcal{F}_{\text{MULT}}$-hybrid model, where $\mathcal{F}_{\text{MULT}}$ can be instantiated with any passively-secure protocol that further guarantees a notion of "security up to additive attacks" in the presence of active adversaries. It is unclear whether their paradigm extends to the dishonest majority setting, since their model of additive attacks is weaker than the standard one formulated in [GIP$^+$14], where in all natural candidates an active attack translates into an additive attack in the latter (stronger) attack model, and is therefore not protected against by the framework of [CGH$^+$18].

In an orthogonal vein, we note that Applebaum et al. [ADI$^+$17] designed the first (variant of) passively-secure OLE based on LPN-style assumptions, implying secure arithmetic computation with asymptotic constant computational overhead over an insecure evaluation of the circuit.

**The state-of-the-art: two-party setting.** In the boolean setting, the protocols of [IPS08] and [HIV17] achieve (asymptotic) constant communication overhead over the passive protocols of [GMW87] and [Yao86], respectively. The latter has the added benefit of matching the number of OT calls in [Yao86], which (unlike [GMW87]) is sublinear in the circuit size. Practical implementations of [IPS08] have been studied in [LPO11], who identified bottlenecks in obtaining concretely-efficient protocols based on the IPS protocol due to the implementation of the so-called "watchlist channels". In the arithmetic setting, a recent work by Hazay et al. [HIMV19] instantiated the framework of [IPS08] with a concretely-efficient honest majority protocol, obtaining small multiplicative overheads (between 2-8) compared to the passive protocol of [GMW87].

| **Construction** | **Number of Parties** | **Hybrid Model** | **Asymptotic Overhead** | **Concrete Overhead** |
|---|---|---|---|---|
| [IPS08] | Constant | OT (passive) | Constant$^*$ | Unexplored |
| [NNOB12] | Two | OT$^{**}$ (active) | $O(s/\log s)$ | — |
| [GIP$^+$14] | Arbitrary | OLE (active) | Constant | 12$^{***}$ |
| [DGN$^+$17] | Two | OLE (active) | Constant | 22$^\dagger$ |
| [WRK17b] | Arbitrary | OT$^{**}$ (active) | $O(s/\log \lvert \mathrm{C} \rvert)$ | — |
| **This work** | Arbitrary | $\mathcal{F}_{\mathrm{MULT}}$ (passive) | Constant | 2 |

Table 2: Asymptotic and concrete communication overheads of state-of-the-art 2PC and MPC protocols in the dishonest majority setting. The overhead is measured as the number of calls to the underlying (passively or actively seucre) OT or OLE functionality, compared to the number of calls made by the passive-GMW to the corresponding (passively secure) functionality (OT or OLE). The concrete overhead column is specified only when the overhead is constant, and holds over sufficiently large fields. $s$ denotes a statistical security parameter, and C is the circuit being evaluated.
$^*$ In terms of asymptotic complexity, we note that [IPS08] also achieves constant rate.
$^{**}$ Security is proven in the random oracle model.
$^{***}$ Based on personal communication with the authors.
$^\dagger$ This constant holds for a particular instantiation of OLE based on noisy encoding.

## 2 Our Techniques

We first recall the so-called "IPS framework" of Ishai, Prabhakaran and Sahai [IPS08], that constructs actively-secure $m$-party protocols for a function $f$ using the following two weaker ingredients as a black-box: (1) an actively-secure honest-majority protocol (the "outer protocol") for $f$ with $m$ clients and $n$ servers, tolerating active corruption of a minority $t < n/2$ of the servers and an arbitrary number of clients; and (2) a passively secure $m$-party protocol (the "inner protocol") for a "simpler" functionality, tolerating an arbitrary number of corruptions.

Using appropriate instantiations of the outer and inner protocols, this framework yields a constant-overhead (in fact, constant-rate) actively-secure protocol for boolean functionalities in the dishonest majority setting with a constant number of parties $m$. However, it does not obtain constant overhead for a super-constant $m$, as we now explain.

**To watch or not to watch?** The high-level idea of the IPS compiler it to have the $m$ parties "virtually" execute the outer protocol by emulating its $n$ servers. Specifically, the parties first obtain (through some joint computation) secret shares of the initial server states, then use the inner protocol on the shared states to generate (secret shares) of the outputs of the "next message" functions of each server. Since the outer protocol is only secure when a majority of the servers are honest, the parties must insure that most servers were correctly emulated, for which it suffices to verify that the parties behave honestly in sufficiently many of the inner protocol executions. The IPS compiler introduces a novel "watchlist" mechanism in which parties "watch" each other to enforce such honest behaviour. More precisely, every party $P_i$ picks a random subset of $t$ servers for which it learns the entire internal state throughout the computation. Consequently, $P_i$ can check that all parties honest emulated the $t$ servers, and abort if some party misbehaves. The identity of servers watched by honest parties remains hidden from the adversary, thus even a single honest party forces the adversary to honestly emulate most (specifically, a majority) of the servers. In terms of parameters, obtaining a $2^{-\Omega(s)}$ soundness error for a statistical security parameter $s$ requires $t, n = \Omega(s)$. Since each corrupted party can choose an arbitrary subset of $t$ watched servers, and there could be $m - 1$ corrupted parties, privacy is only preserved when $(m - 1)t < n/2$. Since achieving constant-overhead requires $n = O(s)$, this is only possible for $m = O(1)$.

**Compute first, check later.** To solve this problem, our first idea is to have a *single* random subset of $t$ servers which are *simultaneously* watched by *all* parties. Of course, now that the identity of the watched servers is known to all parties, it cannot be revealed before the computation has been completed. Instead, the subset is chosen using joint coin-tossing after the circuit has been evaluated, but before the output is reconstructed from the output shares. Correctness is preserved similarly to the original IPS compiler, but checking honest behavior after-the-fact might violate privacy. Indeed, unlike the IPS compiler we can no longer "catch" the adversary as soon as it deviates from the protocol, which raises two privacy concerns. First, by actively deviating from the protocol, the adversary can potentially violate the inner protocol privacy, and learn intermediate values during the circuit evaluation. Second, the adversary can potentially violate the privacy of the outer protocol, by "corrupting" a majority of the servers in the outer protocol (i.e., by not emulating them correctly). We note that even if the inner protocol has the stronger guarantee of remaining *private* even against *active* adversaries, this does not resolve the second issue because as long as the inner protocol is *not actively-secure*, active corruptions in it might violate correctness, which corresponds to corrupting servers in the outer protocol. Thus, an active adversary might still violate privacy in the outer protocol by violating correctness in the inner protocol (thus, in effect, corrupting possibly a majority of the servers).

**Our approach.** Due to these issues, we take a step back, and (instead of extending the IPS framework) focus on designing a new compiler that amplifies the security of a passively-secure inner protocol via a tailor-made outer protocol. Since we use different instantiates of the inner protocol, we model it more generally, assuming the parties have oracle access to an ideal multiplication functionality $\mathcal{F}_{\text{MULT}}$ that works over some agreed-upon secret sharing scheme $\mathcal{S}$. We note that in our compiler, we will not refer to "servers" (or an "outer" protocol), but rather think of these as "copies" of the circuit.

**The combined protocol.** To highlight the main components of our framework, we describe a basic MPC variant that will loosely rely on the passive BGW [BGW88] protocol. Though this does not yield our asymptotic results, it will serve as a good starting point, which we build on to obtain our final framework (as described towards the end of the section).

At the onset of the computation each party $P_i$ secret shares its input $x_i$ using Shamir's secret sharing scheme with privacy parameter $t$, to obtain the shares $\left(X^1, \ldots, X^n\right)$ (as in the passive-BGW protocol). Then, $P_i$ generates additive shares $\left(x_j^l\right)$ of each Shamir share $X^l$, and sends $\left(x_j^l\right)_{l \in [n]}$ to $P_j$. The protocol will evaluates the circuit gate-by-gate as in passive-BGW, where addition gates are locally computed. We will preserve the invariant that when parties evaluate a gate $G$, they collectively hold additive shares of Shamir shares of the values of its input wires. That is, if $G$'s inputs are values $a, b$ which in the passive-BGW protocol have Shamir shares $\left(A^1, \ldots, A^n\right)$, $\left(B^1, \ldots, B^n\right)$ (respectively), then for every $l \in [n]$, party $P_i$ holds values $a_i^l, b_i^l$ such that $\sum_i a_i^l = A^l$ and $\sum_i b_i^l = B^l$.

In passive-BGW, multiplications are performed by having each party locally multiply its Shamir shares $A^l, B^l$, followed by all parties jointly running a degree-reduction sub-protocol on these products. However, in our modified protocol parties can no longer locally compute the products $A^l \cdot B^l$, because no party knows $A^l, B^l$ (parties only know additive shares of these values). To solve this issue, we use an ideal distributed-multiplication functionality $\mathcal{F}_{\text{MULT}}$ which takes as input additive shares of two values $x, y$, and outputs a (fresh) additive sharing of their product $x \cdot y$. (We discuss $\mathcal{F}_{\text{MULT}}$ instantiations below.) This allows parties to learn additive shares of each product $A^l \cdot B^l$.

Once (additive shares of) the products $A^l \cdot B^l$ have been computed, degree reduction should be performed. In the classical passive-BGW protocol, degree reduction requires expensive communication, which is improved by protocols such as [DN07]. We use a new approach that significantly reduces the communication complexity, leveraging the fact that degree-reduction is a linear operation over the Shamir shares.

**Local degree-reduction.** Each party *locally* performs degree reduction over its additive shares of the Shamir shares. Across all parties, the additive shares obtained as a result of this procedure constitute a valid Shamir sharing of the "right" value, due to the linearity properties of Shamir's secret sharing scheme. Intuitively, the second secret-sharing layer allows parties to locally perform degree reduction, because it gives each party a *global "view"* of the protocol execution, as an additive share of the global view of the protocol execution.

**Enforcing correctness.** Once the computation is completed in all copies, we ensure it was performed correctly by incorporating a "correctness-enforcing" mechanism into the protocol. Specifically, before opening the output shares obtained at the outputs of all copies, we first run some correctness tests which will check that (with high probability) all parties honestly executed the computation. The output shares are revealed (and the output is reconstructed from these shares) only if all correctness tests pass.

To explain our correctness tests, we first analyze possible malicious strategies of corrupted parties. Roughly, a corrupted party can deviate from the protocol in one of four ways. First, it can incorrectly share its input (i.e., the "sharing" isn't of the right degree $t$). Second, it can incorrectly perform the degree-reduction procedure, by generating a fresh sharing that either isn't of the right degree (i.e., $t$), or doesn't share the right value (i.e., the value shared before degree reduction). Third, when evaluating a multiplication gate (i.e., computing the product of Shamir shares as described above), it can use different values than the ones provided by $\mathcal{F}_{\text{MULT}}$. Fourth, it can incorrectly perform the local linear computations.

To handle such deviations from the protocol, we introduce three tests. The first is a *degree* test, which checks that the secrets sharings used by all parties, either to share their inputs or as input to multiplication gates, have the right degree. The second is an *equality* test, which checks that the secret sharings before and after degree reduction share the same value. The degree and equality tests jointly guarantee that with overwhelming probability, the input sharings are valid, and the degree reduction procedure was executed correctly (in most copies). Similar degree and equality tests were used in [AHIV17, HIMV19] to check similar conditions. The last test is a *consistency* test, which verifies that (with high probability) parties correctly performed the local computations in (most) copies of the circuit. This checks that the values used by the parties when evaluating a multiplication gate are consistent with the values they obtained from $\mathcal{F}_{\text{MULT}}$, that the local linear operations were performed correctly, and will also guarantee the soundness of the degree and equality tests. For this test, a random subset of copies is chosen, each party reveals its local view of the computation in those copies, and all parties check that the views are consistent with each other. Similar tests were used in the context of MPC-in-the-head [IKOS07, IPS08].

We note that this high-level overview omits important details (see Section 4). For example, the order in which parties commit and reveal the correctness tests' values is crucial to preserving privacy even when the computations in most copies are incorrect. Using this combination of correctness tests, and proving the security of this approach is novel to our work, and requires subtle analysis.

**Achieving constant communication overhead.** Our basic MPC protocol does not achieve constant communication overhead since it increases the communication complexity of the underlying BGW protocol [BGW88] by $O(s)$, where $s$ is a security parameter. We reduce this overhead to constant by replacing [BGW88] with the protocol of Franklin and Yung [FY92] that uses packed secret sharing.

Loosely speaking, packed secret sharing extends Shamir's secret sharing, allowing a block of $\mathcal{B}$ secrets to be shared within a single set of shares. To exploit the advantages of packed secret sharing, we will assume the circuit is arranged in layers that contain only one type (addition/multiplication) of gates, where each phase of the protocol evaluates the gates in one layer.

Using packed secret sharing introduces two main differences from the basic protocol. First, before evaluating a specific layer the parties need to rearrange (repack) the shared secrets corresponding to the input wire values of that layer, to align the packing in blocks with the order of gates within the layer. Then, the layer can be evaluated similarly to the basic protocol (where additions are computed locally, and multiplications involve a call to $\mathcal{F}_{\text{MULT}}$, followed by a local degree-reduction step). The second difference from the basic protocol is that to insure correctness we must now check that the parties correctly rearranged the shared secrets between layers. This is checked through an additional "permutation test" [DI06, AHIV17]. See Section 5 for further details.

This protocol reduces the amortized per-gate communication overhead to constant, because in effect the packed secret sharing allows us to evaluate many gates in one "shot". In particular, the wider the circuit to be evaluated, the larger the gains from employing packed secret sharing.

**Instantiating the multiplication functionality $\mathcal{F}_{\text{MULT}}$.** We instantiate $\mathcal{F}_{\text{MULT}}$ through a reduction to a simpler functionality $\mathcal{F}_{\text{RMULT}}$ which generates (unauthenticated) random triples. All prior protocols that relied on this abstraction (apart from [IPS08]), used actively-secure multiplication protocols to instantiate $\mathcal{F}_{\text{MULT}}$. Interestingly, we can greatly weaken the security of the multiplication protocol, requiring only a passively-secure instantiation, together with a coin tossing protocol to ensure correctly-sampled randomness. Moreover, our protocol can benefit from a preprocessing stage in an offline/online setting, where the triples are generated in the offline phase, and used in the online phase. The consistency test (described for our basic MPC protocol) will ensure, at the cost of a small overhead, that the triples were correctly generated with respect to the tossed coins. We note that unlike prior works, our security analysis can tolerate a small number of ill-formed triples without violating secrecy.

**Related techniques.** Conceptually, our consistency test can be viewed as a combination of the cut-and-choose approach [LP07] and the watchlist mechanism of [IPS08]. Indeed, on the one hand we maintain multiple copies of the computed circuit, yet unlike the cut-and-choose technique the checked copies are not discarded, but rather used in the remainder of the computation to reconstruct the outputs. On the other hand, the purpose of our consistency test is similar to the watchlist channels, which add privacy and correctness to passively-secure protocols. The main difference between our tests and the watchlists of [IPS08] is that in IPS these channels are used to *constantly* enforce correct behaviour *throughout* the protocol execution (and consequently also cause a high overhead), whereas we perform a single consistency test *after the protocol execution has (essentially) ended*, right before the output is reconstructed. These correctness enforcement mechanisms are known to have limitations to achieving scalable MPC. Specifically, the asymptotic limit of cut-and-choose is $O(s/\log|\mathrm{C}|)$ [WRK17b], whereas the watchlists mechanism requires $O(s \cdot n)$ virtual servers for the outer protocol [LOP11]. In both cases, the communication grows with some statistical parameter, and is hence neither constant-overhead nor scalable.

## 3 Preliminaries

In this section we provide necessary preliminaries. Further preliminaries are deferred to the full version [HVW19].

*Basic notations.* We denote a security parameter by $\kappa$. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large $\kappa$'s it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial-time and denote by $[n]$ the set of elements $\{1, \dots, n\}$ for some $n \in \mathbb{N}$. We assume functions to be represented by an arithmetic circuit C (with addition and multiplication gates of fan-in 2), and denote the size of C by $|\mathrm{C}|$. By default we define the size of the circuit to include the total number of gates including input gates. For a random variable $X$, we use $\mathsf{Supp}(X)$ to denote the set of values which $X$ takes with positive probability.

### 3.1 Layered Arithmetic Circuits

An arithmetic circuit defined over a finite field $\mathbb{F}$ is a directed acyclic graph, where nodes (or *gates*) are labelled either as input gates, output gates or computation gates. Input gates have no incoming edges (or *wires*), while output gates have a single incoming wire and no outgoing wires. Computation gates are labelled with a field operations (either addition or multiplication),[4] and have exactly two incoming wires, which we denote as the left and right wire. A circuit with $i$ input gates and $o$ output gates over a field $\mathbb{F}$ represents a function $f : \mathbb{F}^i \to \mathbb{F}^o$ whose value on input $x = (x_1, \ldots, x_i)$ can be computed by assigning a value to each wire of the circuit. Note that this abstraction captures boolean circuits as well, by setting $\mathbb{F} = \mathbb{GF}(2)$. In this work, we will exploit an additional structure of the circuit. Specifically, the gates of an arithmetic circuit can be partitioned into ordered layers $\mathcal{L}_1, \ldots, \mathcal{L}_d$, such that i) a layer only consists of gates of the same type (i.e., addition, multiplication, input or output gates belonging to the same party), and ii) the incoming wires of all gates of layer $i$ originate from gates in layers 0 to $i-1$.

### 3.2 Multiplication Functionalities

A core building block in our protocols is a multiplication functionality $\mathcal{F}_{\textsc{mult}}$ shown in Figure 1, that takes additive shares of two secrets over some field $\mathbb{F}$ and produces additive shares of their product. In fact, we will reduce $\mathcal{F}_{\textsc{mult}}$ to a random instance $\mathcal{F}_{\textsc{rmult}}$, shown in Figure 2, where all shares are chosen uniformly at random from $\mathbb{F}$. The reduction, due to Beaver [Bea91], is as follows. Denote by $[a]$ the additive sharing of some value $a \in \mathbb{F}$, namely, the tuple $(a_1, \ldots, a_m)$. Then, given a random triple $[a], [b], [c]$ obtained as the output of $\mathcal{F}_{\textsc{rmult}}$, and inputs $[x], [y]$ for $\mathcal{F}_{\textsc{mult}}$, we can compute $[xy]$ by first reconstructing $e = [x + a]$ and $d = [y + b]$. Next, the parties compute a (trivial) secret sharing $[ed]$ of $ed$ by having $P_1$ set its share to $ed$, and the rest of the parties set their shares to 0. Finally, the parties compute the following equation (each party locally computes the equation on its own shares)

$$[xy] = [c] + e[y] + d[x] - [ed] = [ab] + (x + a)[y] + (y + b)[x] - (x + a)(y + b).$$

### 3.3 Secret-Sharing

A secret-sharing scheme allows a dealer to distribute a secret among $n$ parties, where each party receives a share (or piece) of the secret during a *sharing phase*. In its simplest

[4] Subtraction gates can be handled analogously to addition gates, and we ignore them here for simplicity.

**Functionality $\mathcal{F}_{\text{MULT}}$**

Functionality $\mathcal{F}_{\text{MULT}}$ communicates with parties $P_1, \ldots, P_m$ and adversary $\mathcal{S}$ corrupting a subset $I \subset [m]$ of parties. It is parameterized by a secret sharing scheme $\mathcal{S} = (\mathsf{Share}, \mathsf{Recon})$ (see Section 3.3 below).

1. Upon receiving the input $(\mathsf{sid}, a_j, b_j)$ from $P_j$ record $(\mathsf{sid}, (a_j, b_j))$.
2. If a tuple is recorded from all parties continue as follows:
   (a) Compute $c = \mathsf{Recon}(a_1, \ldots, a_m) \cdot \mathsf{Recon}(b_1, \ldots, b_m)$.
   (b) Receive corrupted parties' shares $\{c_j\}_{j \in I}$.
   (c) Sample a secret sharing $(c'_1, \ldots, c'_m)$ uniformly at random from $\mathsf{Supp}(\mathsf{Share}(c))$ subject to the constraint that $c'_j = c_j$ for every $j \in I$. For every $j \notin I$, set $c_j = c'_j$.
   (d) Forward $c_j$ to party $P_j$.

Fig. 1: The multiplication functionality.

**Functionality $\mathcal{F}_{\text{RMULT}}$**

Functionality $\mathcal{F}_{\text{RMULT}}$ communicates with parties $P_1, \ldots, P_m$ and adversary $\mathcal{S}$ corrupting the subset of parties in $I \subset [m]$. It is parameterized by a secret sharing scheme $\mathcal{S} = (\mathsf{Share}, \mathsf{Recon})$ (see Section 3.3 below).

1. Receive corrupted parties' shares $\{a_j, b_j, c_j\}_{j \in I}$.
2. Sample secret shares $(a'_1, \ldots, a'_m)$ and $(b'_1, \ldots, b'_m)$ uniformly at random from $\mathsf{Supp}(\mathsf{Share}(\cdot))$ subject to the constraint that $a'_j = a_j$ and $b'_j = b_j$ for every $j \in I$. For every $j \notin I$, set $a_j = a'_j$ and $b_j = b'_j$.
3. Compute $c = \mathsf{Recon}(a_1, \ldots, a_m) \cdot \mathsf{Recon}(b_1, \ldots, b_m)$.
4. Sample a secret sharing $(c'_1, \ldots, c'_m)$ uniformly at random from $\mathsf{Supp}(\mathsf{Share}(c))$ subject to the constraint that $c'_j = c_j$ for every $j \in I$. For every $j \notin I$, set $c_j = c'_j$.
5. Forward $a_j, b_j, c_j$ to party $P_j$.

Fig. 2: The random multiplication functionality.

form, the goal of (threshold) secret-sharing is to allow only subsets of players of size at least $t+1$ to reconstruct the secret. More formally a $t+1$-out-of-$n$ secret sharing scheme comes with a sharing algorithm that on input a secret $s$ outputs $n$ shares $s_1, \ldots, s_n$ and a reconstruction algorithm that takes as input $((s_i)_{i \in S}, S)$ where $|S| > t$ and outputs either a secret $s'$ or $\perp$. In this work, we will use Shamir's secret sharing scheme [Sha79] with secrets in $\mathbb{F} = \mathbb{GF}(2^\kappa)$. We present the sharing and reconstruction algorithms below:

**Sharing algorithm:** For any input $s \in \mathbb{F}$, pick a random polynomial $p(\cdot)$ of degree $t$ in the polynomial-field $\mathbb{F}[x]$ with the condition that $p(0) = s$ and output $p(1), \ldots, p(n)$.

**Reconstruction algorithm:** For any input $(s'_i)_{i \in S}$ where none of the $s'_i$ are $\perp$ and $|S| > t$, compute a polynomial $g(x)$ such that $g(i) = s'_i$ for every $i \in S$. This is possible using Lagrange interpolation where $g$ is given by

$$g(x) = \sum_{i \in S} s'_i \prod_{j \in S/\{i\}} \frac{x-j}{i-j} \ .$$

Finally the reconstruction algorithm outputs $g(0)$.

*Packed secret-sharing.* The concept of packed secret-sharing was introduced by Franking and Yung in [FY92] in order to reduce the communication complexity of secure multi-party protocols, and is an extension of standard secret-sharing. In particular, the authors considered Shamir's secret sharing with the difference that the number of secrets $s_1, \ldots, s_\ell$ is now $\ell$ instead of a single secret, where the secrets are represented as the evaluations of a polynomial $p(\cdot)$ at $\ell$ distinct points. To ensure privacy in case of $t$ colluding corrupted parties, $p(\cdot)$ must have degree at least $t + \ell$. Packed secret sharing inherits the linearity property of Shamir's secret sharing, with the added benefit that it supports batch (block-wise) multiplications. This was used to design secure computation protocols with an honest majority and constant amortized overhead [DI06]. For this reason, we use this tool in our honest majority MPC protocol embedded within our dishonest majority protocol from Section 4, leveraging its advantages to improve the overhead of the former protocol.

## 3.4 Error Correcting Codes

A crucial ingredient in our construction is the use of Reed-Solomon codes as a packed secret sharing scheme [FY92] (as defined in Section 3.3). In what follows, we provide our coding notations and related definitions.

*Coding notation.* For a code $C \subseteq \Sigma^n$ and vector $v \in \Sigma^n$, we denote by $d(v, C)$ the minimal distance of $v$ from $C$, namely the number of positions in which $v$ differs from the closest codeword in $C$, and by $\Delta(v, C)$ the set of positions in which $v$ differs from such a closest codeword (in case of a tie, take the lexicographically first closest codeword). For any $k \leq d(v, C)$, we say that $v$ is $k$-close to $C$, and for every $k > d(v, C)$, we say that $v$ is $k$-far from $C$. We further denote by $d(V, C)$ the minimal distance between a vector set $V$ and a code $C$, namely $d(V, C) = \min_{v \in V}\{d(v, C)\}$.

**Definition 1 (Reed-Solomon code)** *For positive integers $n, k$, finite field $\mathbb{F}$, and a vector $\eta = (\eta_1, \ldots, \eta_n) \in \mathbb{F}^n$ of distinct field elements, the code $\mathsf{RS}_{\mathbb{F},n,k,\eta}$ is the $[n, k, n-k+1]$-linear code*[5] *over $\mathbb{F}$ that consists of all $n$-tuples $(p(\eta_1), \ldots, p(\eta_n))$ where $p$ is a polynomial of degree $< k$ over $\mathbb{F}$.*

**Definition 2 (Encoded message)** *Let $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$ be an RS code and $\zeta = (\zeta_1, \ldots, \zeta_w)$ be a sequence of distinct elements of $\mathbb{F}$ for $w \leq k$. For $u \in L$ we define the message*

[5] We denote by $[n, k, d]$-linear code a linear code of length $n$, rank $k$ and minimum distance $d$, where the minimum distance of the code is the minimal weight of a codeword in the code.

$\mathsf{Decode}_\zeta(u)$ *to be* $(p_u(\zeta_1), \ldots, p_u(\zeta_w))$*, where* $p_u$ *is the polynomial (of degree* $< k$*) corresponding to* $u$*. For* $U \in L^m$ *with rows* $u^1, \ldots, u^m \in L$*, we let* $\mathsf{Decode}_\zeta(U)$ *be the length* $mw$ *vector* $x = (x_{11}, \ldots, x_{1w}, \ldots, x_{m1}, \ldots, x_{mw})$ *such that* $(x_{i1}, \ldots, x_{iw}) = \mathsf{Decode}_\zeta(u^i)$ *for* $i \in [m]$*. We say that* $u$ $L$*-encodes* $x$ *(or simply encodes* $x$*) if* $x = \mathsf{Decode}_\zeta(u)$*.*

Moreover, we recall that $\mathsf{Decode}_\zeta(\cdot)$ is a linear operation, i.e. for any $a, b \in \mathbb{F}^n$ (even if $a, b$ are not in $L$), $\mathsf{Decode}_\zeta(a+b) = \mathsf{Decode}_\zeta(a) + \mathsf{Decode}_\zeta(b)$.

It will be convenient to view $m$-tuples of codewords in $L$ as codewords in an interleaved code $L^m$. We formally define this notion below.

**Definition 3 (Interleaved code)** *Let* $L \subset \mathbb{F}^n$ *be an* $[n, k, d]$ *linear code over* $\mathbb{F}$*. We let* $L^m$ *denote the* $[n, mk, d]$ *(interleaved) code over* $\mathbb{F}^m$ *whose codewords are all* $m \times n$ *matrices* $U$ *such that every row* $U_i$ *of* $U$ *satisfies* $U_i \in L$*. For* $U \in L^m$ *and* $j \in [n]$*, we denote by* $U[j]$ *the* $j$*'th symbol (column) of* $U$*.*

## 4 Basic MPC Protocol

In this section we describe a simple variant of our MPC protocol, which we build on in Section 5 to achieve constant overhead.

Our starting point is a passively-secure variant of the BGW protocol [BGW88], which we amplify to the actively-secure dishonest-majority setting. Amplifying the security of this protocol requires facing three challenges: (1) high overhead due to the degree-reduction sub-protocol; (2) security holds only with a dishonest minority; and (3) security holds only against passive corruptions.

Our strategy towards addressing the first issue is to have parties *locally* perform the degree-reduction procedure which the degree-reduction sub-protocol implements, thus (almost) eliminating the interaction it requires. This is achieved by using a second layer of secret-sharing.

Concretely, our MPC protocol with $m$ parties relies on two layers of secret sharing schemes: (1) first layer sharing: Reed-Solomon codes (which can be thought of as Shamir's secret sharing), denoted by $L$-encoding, where $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$ (cf. Section 3.4); and (2) second layer sharing: additive secret sharing.[6] Throughout the execution, the parties hold additive shares of the $L$-encodings of the wires of the evaluated circuit C. We note that using this two-layer secret sharing decouples the number of parties $m$ from the length of the encoding $n$, since (unlike passive-BGW) parties no longer hold the symbols of the $L$-encodings. In fact, it will be useful to have $m \neq n$. Intuitively, this can be though of as having the parties emulate $n$ copies of C, where the wires of the $l$'th copy carry the $l$'th symbol in the $L$-encodings of the wire values of C, and these symbols are additively shared among the parties. The execution maintains the invariant that when evaluating the gates in layer $\mathcal{L}$, the parties hold for each copy $l$ additive shares of the $l$'th symbols in the $L$-encodings of the outputs of previous layers.

[6] We note that the second layer sharing is added "on top" of the secret sharing used in BGW, and differs from the resharing performed in BGW (in which Shamir shares are reshared using *Shamir*'s scheme). This additional layer of additive sharing allows us to exploit the linearity of BGW's degree reduction procedure to perform degree reduction *locally*.

Our protocol is described in the $\mathcal{F}_{\text{RMULT}}$-hybrid model (cf. Section 3.2) which generates $m$ additive shares of random triples, and is used to execute multiplications. In more detail, the parties evaluate the $n$ copies of C layer by layer, locally performing additions, substractions and multiplications by a constant (this is possible due to the linear nature of our secret sharing schemes), whereas multiplication gates require communication.

Roughly, a multiplication gate $G$ in the $l$'th copy of C is evaluated as follows. The parties hold additive shares of the $l$'th symbols $A^l, B^l$ at the inputs of $G$, and use $\mathcal{F}_{\text{RMULT}}$ (and a reduction from $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$, described in Section 3.2) to obtain additive shares of the product $A^l B^l$. Across all copies, these products form an $\tilde{L}$-encoding of the output wire of $G$, where $\tilde{L} = \mathsf{RS}_{\mathbb{F},n,2k,\eta}$. To obtain a fresh $L$-encoding of the output wire, each party interprets its additive shares of the $\tilde{L}$-encoding (across all copies) as an encoding in $\mathsf{RS}_{\mathbb{F},n,n,\eta}$, decodes it, and then generates a fresh $L$-encoding of this decoded value. The additive shares obtained through this procedure reconstruct to the correct value because degree reduction is a linear operation.

Employing a second secret-sharing layer solves the second challenge (that passive-BGW is only private in the honest majority setting) since a subset of parties learn only a strict subset of additive shares. The third challenge (passive-BGW is only secure against passive corruptions) is handled by incorporating correctness-enforcing tests into the protocol, as described in Section 2.

Our detailed protocol is given in Figures 3-5. We next state the following theorem; its proof appears in the full version [HVW19].

**Theorem 1** *Protocol $\Phi$ described in Figures 3-5 securely realizes $\mathcal{F}$ in the ($\mathcal{F}_{\text{COM}}$, $\mathcal{F}_{\text{RMULT}}$, $\mathcal{F}_{\text{COIN}}$)-hybrid model, tolerating $m-1$ active (static) corruptions, with statistical security error*

$$(1-e/n)^{\delta} + \frac{n-k+2}{|\mathbb{F}|} + 2^{-\Omega(e)}$$

*where $k > \delta + 4e$,$n > 2k + 4e$ and $e \leq (n-k+1)/3$.*

**Proof sketch.** The simulation follows by having the simulator $\mathsf{Sim}$ execute the protocol with the adversary, emulating the ideal functionalities for it, and emulating the honest parties on dummy 0-inputs. Before executing the output decommitment step, $\mathsf{Sim}$ performs several checks regarding the actions of the corrupted parties. Specifically, the simulator determines the set $E$ of copies for which, if they were chosen during the consistency test, the test would fail. It also identifies the set $E'$ of copies in which the $\mathcal{F}_{\text{RMULT}}$ values the corrupted parties committed to are inconsistent with the ones $\mathsf{Sim}$ provided to them. Then, it verifies that $|E| \leq e, |E|' \leq 3e$, and that there exist $\hat{U}, \hat{\boldsymbol{X}}_i, i \in [m]$, and $\hat{\boldsymbol{z}}$ which are valid encodings in the appropriate (interleaved) codes that agree with $\sum_{i\in[m]} U_i, \boldsymbol{X}_i, i \in [m]$, and $\sum_{i\in[m]} \boldsymbol{z}_i$ (respectively) except for the copies in $E$. It also verifies that there exists a $\hat{V}$ in the interleaved code over $\tilde{L}$ that agrees with $\sum_{i\in[m]} V_i$ except for the copies in $E \cup E'$. We note that $\mathsf{Sim}$ can perform these checks because it emulated the internal ideal functionalities for the adversary, whereas the honest parties in the protocol cannot perform these checks. If all checks pass then $\mathsf{Sim}$ can extract effective inputs for the corrupted parties, and use them to

**Protocol $\Phi$.**

- **Inputs.** $P_i$'s input is $x_i$ for all $i \in [m]$. The parties share a description of an arithmetic circuit C with fan-in 2 which contains $h$ multiplication gates and implements functionality $\mathcal{F}$.
- **Initialization.**
  The parties invoke the $\mathcal{F}_{\mathrm{RMULT}}$ functionality $hn$ times. Each invocation yields additive shares $\left(r_1^1, \ldots, r_m^1\right)$, $\left(r_1^2, \ldots, r_m^2\right)$ and $\left(r_1^3, \ldots, r_m^3\right)$, with party $P_i$ holding $(r_i^1, r_i^2, r_i^3)$, such that $r^j = \sum_{i=1}^m r_i^j$ for $j \in \{1,2,3\}$, and $r^3 = r^1 \cdot r^2$. Each party $P_i$ generates a random $L$-encoding $\boldsymbol{\gamma}_i = (\gamma_i^1, \ldots, \gamma_i^n)$ of a random value, a random $L$ encoding $\boldsymbol{\nu}_i = (\nu_i^1, \ldots, \nu_i^n)$ of 0, and a random $\tilde{L}$ encoding $\widetilde{\boldsymbol{\gamma}}_i = (\widetilde{\gamma}_i^1, \ldots, \widetilde{\gamma}_i^n)$ of 0. $P_i$ samples a tuple $(\boldsymbol{\psi}_i^1, \ldots, \boldsymbol{\psi}_i^m)$ such that $\boldsymbol{\psi}_i^j \in \mathbb{F}^n$ and $\sum_{j=1}^m \boldsymbol{\psi}_i^j$ is the all-**0** vector. $P_i$ sends $\boldsymbol{\psi}_i^j$ to party $P_j$. These "blinding" encodings are used in the degree and equality tests of Figure 4.
  Then, for every copy $l \in [n]$, $P_i$ commits using $\mathcal{F}_{\mathrm{COM}}$ to:
  - The triples obtained from the $(l-1) \cdot h + 1, \ldots, hl$'th invocations of the $\mathcal{F}_{\mathrm{RMULT}}$ oracle.
  - $\gamma_i^l, \nu_i^l, \widetilde{\gamma}_i^l$ and $\psi_i^{j,l}$ (i.e., the $l$'th element of $\boldsymbol{\psi}_i^j$) for every $j$.
- **Input sharing.** Each party $P_i$ generates a random $L$-encoding $\boldsymbol{X}_i = \left(X_i^1, \ldots, X_i^n\right)$ of its input $x_i$ (where $X_i^l$ will be used in the evaluation of the $l$'th copy of C), and commits to $X_i^1, \ldots, X_i^n$ using $\mathcal{F}_{\mathrm{COM}}$. For every $1 \le l \le n$, $P_i$ generates an additive sharing $\left(x_{i,1}^l, \ldots, x_{i,m}^l\right)$ of $X_i^l$, and sends $\left(x_{i,j}^l\right)_{l\in[n]}$ to $P_j$. Each party $P_i$ uses the shares $x_{j,i}^l$ ($j \in [n]$) as its inputs to the $l$'th copy.
- **Emulating the computation.** For every copy $l \in [n]$ of C, every layer $\mathcal{L} \in [d]$ in C, and every gate $G \in [w]$ in layer $\mathcal{L}$ (where $w$ is the width of C), do:
  1. **Additions/subtractions.** If $G$ is an addition or subtraction gate, each $P_i$ performs the gate operation by applying it locally on the additive shares maintained as the inputs of that gate in the $l$'th copy.
  2. **Multiplications.** To compute a multiplication gate, the parties invoke the following multiplication protocol, where each party uses as inputs its $l$'th-copy shares of the inputs of $G$.
     - For every $i$, let $a_i^l, b_i^l$ denote the shares of the inputs of $G$ which $P_i$ holds in the $l$'th copy of C. Then the parties compute additive shares $\left(\tilde{c}_1^l, \ldots, \tilde{c}_m^l\right)$ of $(\sum_{i=1}^m a_i^l)(\sum_{i=1}^m b_i^l)$, where $P_i$ receives $\tilde{c}_i^l$, via the reduction from $\mathcal{F}_{\mathrm{MULT}}$ to $\mathcal{F}_{\mathrm{RMULT}}$ (described in Section 3.2), using the first unused triple obtained from $\mathcal{F}_{\mathrm{RMULT}}$ in the (next unused portion of the) randomness generation phase above.
     - Then, $P_i$ locally performs degree reduction on its shares $\tilde{c}_i^1, \ldots, \tilde{c}_i^n$ as follows: it interprets $\left(\tilde{c}_i^1, \ldots, \tilde{c}_i^n\right)$ as an encoding in $\mathsf{RS}_{\mathbb{F},n,n,\eta}$, and applies the decoding procedure to obtain a value $o_i$. It then generates a fresh $L$-encoding $\left(c_i^1, \ldots, c_i^n\right)$ of $o_i$, which it uses as the additive shares of the output of $G$ across the $n$ copies. (We note that $\tilde{c}_i^1, \ldots, \tilde{c}_i^n$ are additive shares of a purported $\tilde{L}$-encoding where $\tilde{L} = \mathsf{RS}_{\mathbb{F},n,2\cdot k,\eta}$, but as a length-$n$ encoding it is always consistent with *some* valid encoding in $\mathsf{RS}_{\mathbb{F},n,n,\eta}$.)
- **Output commitments.** For the output wire $z$, let $\boldsymbol{w}_i$ be the additive shares held by party $P_i$ for the output. Then, $P_i$ computes $\boldsymbol{z}_i = \boldsymbol{w}_i + \boldsymbol{\nu}_i$ where $\boldsymbol{\nu}_i$ is the $L$-encoding of 0 committed to during the initialization step. Then, $P_i$ commits using $\mathcal{F}_{\mathrm{COM}}$ to its shares $\boldsymbol{z}_i = \left(z_i^1, \ldots, z_i^n\right)$.

Fig. 3: **Actively Secure MPC $\Phi$ – Part 1 (Circuit Emulation)**.

**Correctness tests.** The following tests are performed to verify that the parties correctly evaluated the $n$ copies of C (including the degree reduction step executed after each multiplication gate).

- **Commit to degree test.** *This test checks that the input encodings and the shares produced by all parties at the end of every degree reduction step are valid $L$-encodings. This is done by checking that a random linear combination of the sum of all these shares is a valid encoding in $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$.*
  More precisely, the parties first obtain from $\mathcal{F}_{\text{COIN}}$ random vectors $\boldsymbol{r} \in \mathbb{F}^h$, $\boldsymbol{r}' \in \mathbb{F}^m$, and $r'' \in \mathbb{F}$ (recall that $h$ is the number of multiplication gates in C, and $m$ is the number of inputs — one from each party). Next, each party $P_i$ constructs the matrix $U_i \in \mathbb{F}^{h \times n}$ that contains the $L$-encodings obtained after the degree reduction step of all multiplication gates (arranged in some arbitrary order, agreed upon by all parties). Then, $P_i$ locally computes

$$q_i = \boldsymbol{r}^T U_i + r'_i \boldsymbol{X}_i + r'' \boldsymbol{\nu}_i + \boldsymbol{\gamma}_i,$$

  where $\boldsymbol{X}^i$ is the $L$-encoding of $P_i$'s input $x_i$ committed at the input sharing step, $\boldsymbol{\nu}_i$ is the $L$-encoding of 0 committed to by $P_i$ at the initialization step and $\boldsymbol{\gamma}_i$ is the blinding $L$-encoding committed to at the initialization step. $P_i$ then commits to each element of $q_i$, and each column of $U_i$, using $\mathcal{F}_{\text{COM}}$.
- **Commit to equality test.** *This test checks that the degree reduction step was performed correctly. This is done by checking that a random linear combination of the sum of differences of shares before and after the degree reduction step (performed as part of evaluating a multiplication gate) is a valid encoding of 0 in $\widetilde{L} = \mathsf{RS}_{\mathbb{F},n,2k,\eta}$.*
  Specifically, the parties obtain from $\mathcal{F}_{\text{COIN}}$ a random vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_h) \in \mathbb{F}^h$ and random element $\beta \in \mathbb{F}$. $P_i$ sets $V_i$ to contain the additive shares which $P_i$ obtains from the $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$ reduction computed during the evaluation of multiplication gates. Next, $P_i$ locally computes:

$$\tilde{q}_i = \boldsymbol{\alpha}^T (V_i - U_i) + \beta \boldsymbol{\nu}_i + \widetilde{\boldsymbol{\gamma}}_i + \boldsymbol{b}_i$$

  where $b_i^l = \sum_{j=1}^m \psi_j^{i,l}$, $\widetilde{\boldsymbol{\gamma}}_i$ is the $\widetilde{L}$-encoding of 0 from the initialization step. Finally, $P_i$ commits to each element of $\tilde{q}_i$ using $\mathcal{F}_{\text{COM}}$.

Fig. 4: **Actively Secure MPC $\Phi$ – Part 2 (Correctness Tests Commitments).**

obtain the output from the trusted party. Finally, Sim "corrects" the output shares of the honest parties to share the correct outputs.

Next, we highlight some of the challenges we face when proving indistinguishability of the simulated and real views. Recall that unlike [IPS08] we run a single consistency test, right before output reconstruction. Thus, we essentially have one "shot" to catch the adversary, causing the test to be more involved. Another challenge is that parties are only committed to small portions of the execution, whereas in [IPS08] parties commit to *all their messages* via the watchlists channels. Consequently, Sim cannot verify correct behavior directly by checking the messages, and instead we need to show

- **Consistency test.** *This test checks that the parties correctly executed the local computations in each copy.*
  $P_1, \dots, P_m$ obtain from $\mathcal{F}_{\text{COIN}}$ a random subset $\Gamma \subset [n]$ of size $\delta$. For every $l \in \Gamma$, each $P_i$ opens its entire view of the execution of the $l$'th copy of C. Specifically, $P_i$ decommits $X_i^l$, and the randomness (including all components of the commitments generated in the initialization step) it used in the execution of the $l$'th copy. It also opens the commitments to the degree and equality tests, and the additive shares of the final outputs of the $l$'th copy. Then, $P_i$ checks (as described next) that all local computations in the copies in $\Gamma$ were performed correctly, aborting if an inconsistency is detected.
  To check the $l$'th copy, $P_i$ first checks that for every $j \in [m]$, $\sum_{j' \in [m]} \psi_j^{j',l} = 0$. Then, it obtains the $l$'th column of $U_j$ and $\boldsymbol{z}_j$ from the decommitments of $P_j$, and uses the decommitments to $\mathcal{F}_{\text{RMULT}}$ values to determine the multiplication triples used by all parties for the $l$'th copy. Using these triples, $P_i$ determines the inputs and outputs each party used in each multiplication gate of the $l$'th copy. Having determined the outputs of multiplication gates, $P_j$ can reconstruct the $l$'th column of $V_j$. Moreover, since the final output is a linear combination of outputs of multiplication gates and parties' inputs, $\sum_j \boldsymbol{w}_j^l$ can be obtained by computing this linear combination over the corresponding rows in $\sum_j U_j$'s and the $\boldsymbol{X}_j$'s.
  Since addition gates are evaluated locally, correct execution of addition gates can be verified by checking that the inputs to all multiplication gates were computed correctly. Recall that an input to a multiplication gate is a linear combination of outputs of previous multiplication gates and parties' inputs. Thus, correctness can be checked by verifying that the sum of additive shares used as inputs to multiplication gates by all parties (as inferred from the $\mathcal{F}_{\text{RMULT}}$ triples, and the transcript), and the linear combination of the corresponding rows in $\sum_j U_j$ and the $\boldsymbol{X}_j$'s, are equal.
  Parties also verify that the reduction from $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$ was computed correctly in the $l$'th copy, and that $z_i^l = w_i^l + \nu_i^l$ for every $i$.
- **Degree test check.** The parties decommit the degree test commitments for all remaining copies $l \notin \Gamma$, namely each $P_i$ opens the commitment to the value $q_i$ computed in Figure 4. (Note that the parties do not decommit the remaining columns of $U_i$.) Each party computes the vector $q = (q_1 + \ldots + q_m)$ and aborts if $q$ is not a valid $L$-encoding.
- **Equality test check.** The parties decommit their equality test commitments for all copies $l \notin \Gamma$, namely each $P_i$ opens the commitment to the value $\tilde{q}_i$ computed in Figure 4. Each party computes $\tilde{q} = (\tilde{q}_1 + \ldots + \tilde{q}_m)$, and aborts if either $\tilde{q} \notin \tilde{L}$ or $\tilde{q}$ does not decode to the value 0.
- **Output decommitments.** If the consistency, degree and equality tests pass correctly, then every party $P_i$ decommits its output commitments for all copies $l \notin \Gamma$. The parties then locally reconstruct $\boldsymbol{z} = \sum_i \boldsymbol{z}_i$, and if it is an $L$-encoding, decode the output of C from the encoding.

Fig. 5: **Actively Secure MPC $\Phi$ – Part 3 (Correctness Tests)**.

that the messages can be extracted from the partial information which parties commit to. Fortunately, we show that correctness can be defined based on the $\mathcal{F}_{\text{RMULT}}$ inputs, and the transcript of the reduction from $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$. Finally, correctness is guaranteed by the combination of local and global checks in our protocol. Specifically, the consistency test verifies *local* correctness of the computation within each copy, by inspecting a subset of copies; and the degree and equality tests verify that some *global* relation holds over all copies (i.e., all additive shares).

In the proof, we show that if all the protocol tests pass then except with negligible probability, all the conditions checked by the simulator before the output reconstruction phase hold, and moreover the output is consistent with the outputs of the honest parties, and the effective outputs that $\mathsf{Sim}$ extracts for the corrupted parties. Thus, it suffices to prove indistinguishability of the simulated distribution and a hybrid distribution which is obtained from the real execution by performing $\mathsf{Sim}$'s checks, and aborting if they are violated. The difference between the hybrid and simulated distributions is that the honest parties use their real inputs in the former, and 0-inputs in the latter. We prove indistinguishability by a case analysis based on which tests pass. Intuitively, the views revealed during the consistency tests are identically distributed due to the secrecy of Shamir's secret sharing scheme (alternatively, Reed-Solomon codes). The degree test values are indistinguishable because the honest parties' values are valid $L$-encodings, which are uniformly random due to the masking by the $\boldsymbol{\gamma}_i$'s. The equality test values are indistinguishable because the sum of honest parties' values are valid $\widetilde{L}$-encodings of $\mathbf{0}$, which are uniformly random subject to this constraint due to the masking by the $\widetilde{\boldsymbol{\gamma}}_i$'s. Since the equality test values are masked by additive shares of $\mathbf{0}$, the values themselves are identically distributed. Finally, conditioned on all tests passing, the output shares are uniformly random $L$-encodings whose sum encodes the correct output, due to the masking by the $\boldsymbol{\nu}_i$'s.

**Communication complexity of protocol $\Phi$.** Assuming the existence of a PRG, parties can commit to their $\mathcal{F}_{\text{RMULT}}$ triples by committing (during the initialization step) to a PRG seed for each copy (the other initialization-phase commitments are generated as in Figure 3). Consequently, the total communication, assuming rate-1 commitments, is:

$$\underbrace{n \cdot m \cdot (\kappa + (3+m) \cdot \log_2 |\mathbb{F}|)}_{\text{rnd/blind com.}} + \underbrace{m \cdot n \cdot \log_2 |\mathbb{F}|}_{\text{input commitments}} + \underbrace{m^2 \cdot n \cdot \log_2 |\mathbb{F}|}_{\text{input sharing}}$$
$$+ \underbrace{n \cdot h \cdot \mathrm{CC}_{\text{MULT}}}_{\text{multiplication}} + \underbrace{|\mathit{\Gamma}| \cdot m \cdot (\kappa + (4+m) \cdot \log_2 |\mathbb{F}|)}_{\text{consistency test}}$$
$$+ \underbrace{2 \cdot m \cdot n \cdot \log_2 |\mathbb{F}|}_{\text{degree test com. and dec.}} + \underbrace{2 \cdot m \cdot n \cdot \log_2 |\mathbb{F}|}_{\text{equality test com. and dec.}} + \underbrace{2 \cdot n \cdot m \cdot \log_2 |\mathbb{F}|}_{\text{output com. and dec.}}$$

where $\mathrm{CC}_{\text{MULT}}$ is the communication complexity of the $m$-party multiplication protocol (implementing $\mathcal{F}_{\text{RMULT}}$ and the $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$ reduction), and $h$ is the number of multiplication gates in the circuit. (We note that the degree and equality test commitments revealed during the consistency test are counted as part of the degree and equality test terms, resp.) In order to get $2^{-\Omega(s)}$ soundness, we need to set $n = O(s)$. Assuming $s \leq \kappa$, the overall communication complexity can be bounded by $O(s \cdot h \cdot \mathrm{CC}_{\text{MULT}}) + \mathsf{poly}(m, \kappa, \log_2 |\mathbb{F}|)$. Since $h$ represents the size of the circuit (i.e. number of multiplication gates), the best passive protocol in the $\mathcal{F}_{\text{MULT}}$-hybrid can be bounded by $O(h) \cdot \mathrm{CC}_{\textsc{mult}}$. Therefore, the communication overhead of our basic variant is $O(s)$.

## 4.1 Instantiating $\mathcal{F}_{\textsc{rmult}}$

Recall from Section 4.1 that $\mathcal{F}_{\textsc{rmult}}$ is the multiplication functionality that outputs three tuples of additive shares $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ such that the "inputs" $\boldsymbol{a}, \boldsymbol{b}$ share random values $a, b$, and the "output" $\boldsymbol{c}$ shares the product $a \cdot b$. In this section we discuss how to realize this functionality, while identifying the minimal security properties required from it.

Our first observation is that we do not need an actively-secure implementation of the $\mathcal{F}_{\textsc{rmult}}$ functionality. In fact, it suffices to consider a protocol that is only "private" against active adversaries, in the sense that throughout the protocol execution, an actively corrupted party cannot violate the privacy of the honest parties' inputs. In particular, the underlying implementation does not have to retain correctness in this case, or provide a mechanism for extracting the adversary's inputs. Extraction in our protocol is achieved by requiring the adversary to commit to its input and randomness used for the $\mathcal{F}_{\textsc{rmult}}$-functionality. Correctness, on the other hand, is enforced through our consistency test that ensures correctness of the computations in most of the copies, by checking a random subset of $\delta$ copies.

When computing a boolean circuit, the pairwise products of the shares can be computed using Oblivious Transfer (OT) [Bea91, NNOB12]. Based on the discussion above, it suffices to use a private OT protocol [HK12]. Indeed, consistency between the different OT executions will be verified during the consistency test of our protocol, as discussed above.[7] Intuitively, privacy is guaranteed because an OT sender has no output in the execution, and the security/privacy of OT ensures that even if the sender cheats it learns nothing about the receiver's input. Moreover, though an OT receiver can use inconsistent inputs in the OT executions with different honest parties, this can only violate correctness, and not privacy, since the output of each OT execution is an additive share of the cross product (e.g., $a_i \cdot b_j$), which reveals nothing about the other party's share. Similarly, when working over large fields, $\mathcal{F}_{\textsc{rmult}}$ can be realized using private OLE, where private OLE can be defined analogously to private OT, requiring that parties do not infer any additional information (except what can be inferred from their inputs and outputs).

**Relaxing to passive implementation of the $\mathcal{F}_{\textsc{rmult}}$-functionality.** We can further weaken the security requirement on the $\mathcal{F}_{\textsc{rmult}}$ implementation, by incorporating the reduction from defensible privacy to passive security. We first informally review the notion of *defensible privacy* which was introduced by Haitner in [Hai08, HIK$^{+}$11]; see [HIK$^{+}$11] for the formal definitions. Let $\pi$ be a two-party protocol between $P_1$ and $P_2$, and let $\mathsf{trans} = (q_1, a_1, \ldots, q_\ell, a_\ell)$ be a transcript of an execution of $\pi$ when $P_1$ is controlled by an adversary $\mathcal{A}$, where $q_i$ denotes the $i$'th message from $P_1$, and $a_i$ denotes the $i$'th message from $P_2$ (that is, $a_i$ is the response to $q_i$). Informally, a *defence* of $\mathcal{A}$ relative to $\mathsf{trans}$, which is provided after the protocol execution ends, is a pair $(x, r)$

[7] More specifically, we use OT between pairs of parties to compute a 2-out-of-2 additive secret sharing of the product they should compute. Then. we perform the consistency check, and reconstruct the outputs of OTs only if this test passes.

of input and random tape for $P_1$. We say that a defence $(x, r)$ of $\mathcal{A}$ relative to $\mathsf{trans}$ is *good* if the transcript generated by running the honest $P_1$ with input $x$ and random tape $r$ against $P_2$'s messages $a_1, \ldots, a_\ell$ results exactly in $\mathsf{trans}$. Intuitively, a defense $(x, r)$ is good relative to $\mathsf{trans}$ if, whenever $P_i$ uses $(x, r)$ in an honest execution of $\pi$, the messages sent by $P_i$ are identical to the messages sent by the adversary in $\mathsf{trans}$. Thus, in essence, a defense serves as a "proof" of honest behavior. Defensible privacy guarantees that when the adversary provides a good defence, then it learns nothing beyond what can be inferred from its input and prescribed output.[8]

The security of a passive protocol can be amplified to defensible privacy by adding a coin tossing phase (which, in our case, samples the inputs to $\mathcal{F}_{\text{RMULT}}$), and ensuring that these coins were indeed used in the passive execution. The latter can be checked as part of our consistency test, however to guarantee privacy we cannot postpone this check until the consistency test is performed at the end of the circuit emulation, since by that time the adversary could have already violated privacy by using badly-sampled randomness. Thus, we include in our protocol two consistency tests: the first is the consistency test described in Figure 4, and the second checks consistency of $\mathcal{F}_{\text{RMULT}}$ inputs and the tossed coins, and is executed during the initialization phase. This second consistency test ensures that with overwhelming probability, all but (possibly) a small subset of random triples are correct (namely, the aggregated parties' shares correspond to $c = a \cdot b$), and consistent with the random coins. This will suffice for our security analysis, because the number of copies will be sufficiently large such that by cheating in a small number ($< k$) of copies, the adversary learns nothing.

**Relaxing further to weaker than passive.** Following ideas from [HIMV19], our protocol can, in fact, tolerate an *imperfect* passive OLE, namely one that has a non-negligible statistical privacy or correctness error. This security feature can be turned into an efficiency advantage. For example, imperfect $\mathcal{F}_{\text{RMULT}}$ can be implemented more efficiently by aggressively setting the parameters in existing LWE-based OLE constructions, see the full version [HVW19] for details.

## 5 Actively Secure MPC with Constant Communication Overhead

In this section we present our main result, namely, an MPC protocol for an arbitrary number of parties that achieves constant communication overhead over the passive GMW protocol.

On a high-level, we will incorporate a variant of the protocol of Frankling and Yung [FY92] instead of [BGW88] in our basic MPC protocol. Recall that the main overhead in the basic MPC protocol is caused by the $n = O(s)$ copies of the circuit used to perform the computation, where $s$ is a statistical security parameter. Then, similar to [FY92] we improve the communication overhead, achieving *constant* overhead, by

[8] For instance, an OT protocol is defensibly private with respect to a corrupted sender if any adversary interacting with an honest receiver with input $u$, and providing a good defence at the end of the execution, does not learn $u$. Similarly, an OT protocol is defensibly private with respect to a corrupted receiver if for any input $u$, and any inputs $(v_0, v_1)$ for the sender, any adversary interacting with the honest sender with input $(v_0, v_1)$, that is able to provide a good defense for input $u$, does not learn $v_{1-u}$.

having all copies evaluate multiple gates in parallel using packed secret sharing. Our protocol will achieve constant-overhead for moderately wide circuits (See Section 6 for a more detailed discussion.)

In more detail, given a circuit C, and block-width parameter $\mathcal{B}$, the parties agree on a divisions of the circuit evaluation into layers, where at most $\mathcal{B}$ multiplication gates are evaluated in parallel in each layer, and arbitrary linear operations are performed between layers. During the evaluation of the protocol on a specific input, we can associate with each layer of gates $G$ a vector (block) $B_O^G$ of $\mathcal{B}$ values whose $i$'th position contains the output value assigned to the $i$'th gate in the layer (or 0 if the block has less than $\mathcal{B}$ gates). For each layer (except blocks of input gates), we will associate two additional blocks: a "left" block $B_L^G$ and "right" block $B_R^G$ whose $i$'th position contains the value of the left input wire and right input wire of the $i$'th gate, respectively. In other words, the value of the $i$'th gate of a multiplication block can be expressed as $(B_O^G)_i = (B_L^G)_i(B_R^G)_i$. In the protocol, the parties will collectively operate on an efficient (constant-rate) Reed-Solomon encoding (equivalently, packed secret shares) of each block. The protocol parameters include a description of the Reed-Solomon code $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$, and a vector of elements $\zeta = (\zeta_1, \dots, \zeta_{\mathcal{B}}) \in \mathbb{F}^{\mathcal{B}}$ which is used for decoding.

Next, we describe our protocol, simulation and proof by specifying the main differences from the description of the basic protocol from Section 4.

- INITIALIZATION. Recall that each party generates $\boldsymbol{\gamma}_i, \boldsymbol{\nu}_i, \widetilde{\boldsymbol{\gamma}}_i$ and $(\boldsymbol{\psi}_i^1, \dots, \boldsymbol{\psi}_i^m)$. The parties generate the same vectors except that $\boldsymbol{\gamma}_i$ is a random $L$-encoding of a random block of values, and $\boldsymbol{\nu}_i$ and $\widetilde{\boldsymbol{\gamma}}_i$ are random $L$ and $\tilde{L}$ encodings of the all 0's block. In addition, the parties generate a random $L'$-encoding $\boldsymbol{\gamma'}^i = (\gamma'^i_1, \dots, \gamma'^i_n)$ of a block of values that are random subject to the condition that they add up to 0, where $L' = \mathsf{RS}_{\mathbb{F},n,k+\mathcal{B},\eta}$.
- INPUT SHARING. The parties share a block rather than a single element. Namely, the parties embed their input value(s) into a block of length $\mathcal{B}$, and generates a packed secret sharing $L$-encoding for this block, distributing the outcome as in the basic protocol.
- EMULATING THE COMPUTATION. The computation proceed in layers of multiplication gates, where for each layer, we maintain the invariant that the parties hold additive shares of the inputs to the (at most) $\mathcal{B}$ multiplication gates in the layer. The difference from the basic protocol is that before evaluating a layer, the parties need to repack the inputs to the layer. (See discussion below on why repacking might be needed.)

  Concretely, to evaluate a layer, each party first rearranges the left wire values and right wire values of the multiplication gates in the layer into blocks $B_L$ and $B_R$, and generates an $L$-encoding of each block. For every $i$, let $a_i^l, b_i^l$ denote $P_i$'s shares of $B_L, B_R$ (respectively) corresponding to the $l$'th copy. Then the parties compute (via the reduction from $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$) additive shares $\left(\tilde{c}_1^l, \dots, \tilde{c}_m^l\right)$ of $(\sum_{i=1}^m a_i^l)(\sum_{i=1}^m b_i^l)$, where $P_i$ receives $\tilde{c}_i^l$, just as in the basic MPC protocol. Then, each $P_i$ locally performs the degree reduction procedure as in the basic MPC protocol, with the difference that $P_i$ decodes $(\tilde{c}_i^1, \dots, \tilde{c}_i^n)$ to obtain a *block* of values which it uses as the additive shares of the outputs of the multiplication gates in the layer.

*Why repacking is needed.* To see why rearranging the values within and between blocks might be necessary, consider a circuit that has a wire connecting the 3'rd value in the 2'nd block in layer 1 with the 5'th value in the 3'rd block in layer 2; or a wire connecting the 4'th value in the 1'st block in layer 1 with the 2'nd value in the 1'st block in layer 2.[9]

– CORRECTNESS TESTS. We will employ the equality test as before, modify the degree test to also check the repacked encodings, and add an additional *permutation test*, as described next.

THE MODIFIED DEGREE TEST. As in the basic protocol, the degree test will compute a random linear combination of the tested encodings. These encodings include the blocks $\boldsymbol{X}_i$ encoding the parties' inputs (which were committed in the input sharing step), the block of 0s encoded in $\boldsymbol{\nu}_i$ (which was committed in the initialization step), and the matrix $U_i$ which contains $L$-encodings of the blocks of additive shares that were obtained from the degree reduction step following a multiplication step ($U_i$ was committed to during the commit to degree test step). The difference from the degree test of the basic MPC protocol is that the linear combination will now also include an additional matrix $U_i''$ which contains the $L$-encodings of the repacked blocks of additive shares that were used as inputs to multiplication gates. (We note that these values are never committed to, but as explained in the proof of Corollary 2 below, can be extracted by the simulator from the transcript of the execution.) More formally, the parties will obtain from $\mathcal{F}_{\text{COIN}}$ random vectors $\boldsymbol{r}, \boldsymbol{r}', \boldsymbol{r}'''$, and a random value $r''$, and party $P_i$ will compute

$$q_i = \boldsymbol{r}^T U_i + \boldsymbol{r}'''^T U_i'' + r_i' \boldsymbol{X}_i + r'' \boldsymbol{\nu}_i + \boldsymbol{\gamma}_i.$$

**Permutation test:** this test verifies that the parties correctly permute (i.e., rearrange) the additive shares of wire values between layers. In particular, the test verifies that the encodings of the left and right input blocks of each computation layer correctly encode the values from the previous layers (and similarly for the output blocks). Note that the set of constraints that the blocks of values have to satisfy can be expressed as a set of linear equations in at most $m\mathcal{B}$ equations and $m\mathcal{B}$ variables (where $w$ is the width, $d$ is the depth of the computed circuit, and $m = dw/\mathcal{B}$), where variable $x_{i,j}$ represents the $j$'th value in the $i$'th block. (For example, if the circuit has a wire between the 3'rd value of the 2'nd block and the 5'th value in the 3'rd block, the corresponding constraint would be $x_{2,3} - x_{3,5} = 0$.) These linear equations can be represented in matrix form as $Ax = 0^{m\mathcal{B}}$, where $A \in \mathbb{F}^{m\mathcal{B} \times m\mathcal{B}}$ is a public matrix which only depends on the circuit being computed. The permutation test simply picks a random vector $\boldsymbol{r} \in \mathbb{F}^{m\mathcal{B}}$ and checks that $(\boldsymbol{r}^T A)x = 0$. To check these constraints, the parties obtain from $\mathcal{F}_{\text{COIN}}$ a random vector $\boldsymbol{r} \in \mathbb{F}^{m\mathcal{B}}$ and compute

$$\boldsymbol{r}^T A = (r'_{11}, \ldots, r'_{1\mathcal{B}}, \ldots, r'_{m1}, \ldots, r'_{m\mathcal{B}}).$$

[9] Addition gates do not require repacking: they can be computed locally (because parties hold additive shares of the wire values), then repacked for the next multiplication layer.

Now, let $r_j(\cdot)$ be the unique polynomial of degree $< \mathcal{B}$ such that $r_j(\zeta_Q) = r'_{jQ}$ for every $Q \in [\mathcal{B}]$ and $j \in [m]$. Then party $P_i$ locally computes $q'_i = (r_1(\zeta_i), \ldots, r_m(\zeta_i))^T U'_i + \gamma'_i$, where $\gamma'_i$ is the blinding encoding from the initialization step (that encode in $\mathsf{RS}_{\mathbb{F},n,k+\mathcal{B},\eta}$ random blocks of values that sum to 0), and $U'_i$ is the matrix obtained by concatenating the rows of $U_i$ and $U''_i$ from the degree test. Notice that the rows of $U'_i$ consist of the $L$-encodings which $P_i$ obtained at the output of multiplication layers (after degree reduction), and the $L$-encodings it used as inputs to multiplication layers (after repacking). Finally, $P_i$ commits to each element of $q'_i$using $\mathcal{F}_{\text{COM}}$.

- CONSISTENCY TEST CHECK. In the consistency test, we also check for all $l \in \Gamma$ that the permutation test values of copy $l$ were computed correctly. Specifically, each party checks that for every $i \in [m]$, the $l$'th element of $q'_i$ is consistent with the $l$'th element of $\gamma'_i$, the $l$'th column of $U'_i$, and $\boldsymbol{r}$ (the coins obtained from $\mathcal{F}_{\text{COIN}}$ for the permutation test).
- PERMUTATION TEST CHECK. The parties decommit their permutation test commitments for all copies $l \notin \Gamma$, namely each $P_i$ opens the commitment to the value $q'_i$ computed above. Each party computes $q'_i = (q'_1 + \ldots + q'_m)$, and aborts if $q' = (q'_1, \ldots, q'_n) \notin \mathsf{RS}_{\mathbb{F},n,k+\mathcal{B},\eta}$ or $x_1 + \cdots + x_w \neq 0$ where $x = (x_1, \ldots, x_w) = \mathsf{Decode}_{\zeta}(q')$.

The following Theorem follows from Theorem 1 and the discussion above; its proof appears in the full version [HVW19].

**Theorem 2** *The packed variant of protocol $\Phi$ of Figures 3-5 securely realizes $\mathcal{F}$ in the $(\mathcal{F}_{\text{COM}}, \mathcal{F}_{\text{RMULT}}, \mathcal{F}_{\text{COIN}})$-hybrid model, tolerating $m-1$ active (static) corruptions, with statistical security error*

$$(1 - e/n)^{\delta} + ((e + k + \mathcal{B})/n)^{\delta} + (n - k + 3)/|\mathbb{F}| + 2^{-\Omega(e)}$$

*where $k > \delta + 4e + \mathcal{B}$, $n > 2k + 4e$, and $e < (n - k + 1)/3$.*

Assuming that each layer of the circuit has at least $\mathcal{B}$ parallel multiplications, the communication complexity of this variant is given by $O(n \cdot \frac{h}{\mathcal{B}} \cdot \text{CC}_{\text{MULT}}) + \mathsf{poly}(m, \kappa, \log_2 |\mathbb{F}|)$ since we amortize over $\mathcal{B}$ multiplications. By setting $n = O(s)$, the amortized communication overhead of this protocol becomes $O(1)$ per copy. Circuits of an arbitrary structure can be easily handled at a worst-case additional cost of $\mathsf{poly}(s, d)$. The statistical error can be improved by repeating the tests. The analysis presented above works for fields of size larger than $n$, for smaller fields, we can rely on the analysis from [DI06].

## 6 Corollaries and Applications

In this section we consider several different instantiations of the $\mathcal{F}_{\text{RMULT}}$ functionality, thus obtaining our main results in the different settings as instances of the generic protocol of Section 5.

## 6.1 Constant Overhead MPC for Constant-Size Fields

**Dishonest majority.** Our main result is obtained by replacing the Reed-Solomon codes in our protocol with Algebraic Geometric (AG) secret sharing over fields of constant size [CC06], instantiating the $\mathcal{F}_{\text{RMULT}}$ functionality with pairwise calls to a passively-secure implementation of the $\mathcal{F}_{\text{OT}}$ functionality, and instantiating commitments using a pseudorandom generator. Formally:

**Theorem 3 (Theorem 1, restated)** *Let $\kappa, s$ denote computational and statistical security parameters (resp.), $m$ denote the number of parties, and $\mathbb{F}$ be a constant-size field. Then there exists a protocol compiler that, given a pseudorandom generator G with seed length $\kappa$, $s$, a constant-round implementation of the $\mathcal{F}_{\text{OT}}$ functionality with total communication complexity $\text{CC}_{\text{OT}}$, and a description of an $m$-party functionality expressed as a depth-$d$ circuit $\text{C}$ with constant fan-in, outputs a UC-secure $O(d)$-round $m$-party protocol realizing $f$ with communication complexity $O(m^2\,|\text{C}|\,\text{CC}_{\text{OT}}) + \mathsf{poly}(m, \kappa, d)$, where security holds against an active adversary corrupting an arbitrary number of parties.*

We note that the exact constants in the overhead of Theorem 3 depend on the concrete constants of the underlying AG code, which have not been studied before. The communication complexity of our protocol using a bit-OT protocol for the boolean setting asymptotically matches the communication complexity of the best known passively-secure protocol, namely [GMW87] using a passively-secure OT protocol. The best known previous result for active security is due to Genkin et al. [GIW16] who achieve $O(m^2\,|\text{C}|\,\mathsf{poly}\log(s))$ communication complexity, i.e., a multiplicative factor of $\mathsf{poly}\log(s)$ over GMW.

**Honest majority.** To obtain our main result for the honest majority setting, we need to slightly modify our protocol in two ways. First, we will rely on the passive variant of a protocol of Damgård and Nielsen [DN07], instantiated with secret-sharing based on AG codes over constant-size finite fields, to instantiate the parallel $\mathcal{F}_{\text{RMULT}}$ functionality (i.e., to generate the triples in the initialization phase). To achieve this, we replace the additive secret sharing used in our protocol with secret sharing based on AG codes for constant-size fields. We note that the passively-secure honest-majority $m$-party protocol of [DN07] can generate $T = \Omega(m)$ random triples with total communication complexity $O(mT)$. Second, we will consider $\mathcal{F}_{\text{RMULT}}$ and $\mathcal{F}_{\text{MULT}}$ whose underlying secret sharing scheme is based on the same AG secret sharing scheme. Specifically, parallel $\mathcal{F}_{\text{RMULT}}$ distributes secret-shares of many triples $a, b$ and $c$ such that $a \cdot b = c$. Then the $\mathcal{F}_{\text{MULT}}$ to $\mathcal{F}_{\text{RMULT}}$ reduction works essentially as in the basic protocol, where the only difference is that the values $e, d$ are reconstructed using the reconstruction procedure of the AG secret sharing scheme. Consequently, we obtain the following theorem.

**Theorem 4** *Let $\kappa, s$ denote computational and statistical security parameters (resp.), $m$ denote the number of parties, and $\mathbb{F}$ be a constant-size field. Then there exists a protocol compiler that, given a pseudorandom generator G with seed length $\kappa$, $s$, and a description of an $m$-party functionality expressed as a depth-$d$ circuit $\text{C}$ with constant fan-in, outputs a UC-secure $O(d)$-round $m$-party protocol realizing $f$ with*

*$O(m|\mathrm{C}|) + \mathsf{poly}(m, \kappa, d)$ bits total communication complexity, and security against a static adversary corrupting a minority of parties.*

We remark that this improves over the result of Chida et al. [CGH$^+$18] that achieves $O(s)$ overhead for binary fields, and generalizes the result of Ishai et al. [IKP$^+$16] that achieves the same result, but only for a constant number of parties. We remark that the latter protocol additionally achieve constant-rate, while our protocol only achieves constant-overhead.

## 6.2 Constant Overhead MPC over Fields of Arbitrary Size

**Dishonest majority.** To obtain our result for fields of arbitrary size, we realize the $\mathcal{F}_{\text{RMULT}}$ functionality using a passively-secure OLE protocol. For fields of size $\leq s$ we rely on AG sharing, whereas for fields of size $\Omega(s)$ we use Reed-Solomon codes. Thus, we can re-derive a result of Genkin et al. [GIP$^+$14] (Theorem 5.7 in the full version), who construct an actively-secure $m$-party protocol for arbitrary functionalities (represented by an arithmetic circuit C), in the dishonest majority setting, using $O(m^2\,|\mathrm{C}|)$ calls to an OLE oracle. More precisely, we have the following theorem:

**Theorem 5** *Let $\kappa, s$ denote computational and statistical security parameters (resp.), $m$ denote the number of parties, and $\mathbb{F}$ be a field. Then there exists a protocol compiler that, given a pseudorandom generator $G$ with seed length $\kappa$, $s$, a constant-round implementation of the $\mathcal{F}_{\text{OLE}}$ functionality over $\mathbb{F}$ with total communication complexity $\mathrm{CC}_{\text{OLE}}$, and a description of an $m$-party functionality expressed as a depth-$d$ arithmetic circuit $\mathrm{C}$ over $\mathbb{F}$ with constant fan-in, outputs a UC-secure $O(d)$-round $m$-party protocol realizing $f$ with communication complexity $O(m^2\,|\mathrm{C}|\,\mathrm{CC}_{\text{OLE}}) + \mathsf{poly}(m, \kappa, d)$ field elements, with security against an active adversary corrupting an arbitrary number of parties.*

This result asymptotically matches the communication complexity of the best known passively-secure protocol [GMW87] using a passively-secure OLE protocol. Furthermore, for sufficiently wide circuits, we can show that the overhead of our protocols is 2. We present the concrete parameters in the full version [HVW19].

**Honest majority.** Just as in Section 6.1, we can obtain constant overhead over the best passively-secure protocol in the honest majority setting:

**Theorem 6** *Let $\kappa, s$ denote computational and statistical security parameters (resp.), $m$ denote the number of parties, and $\mathbb{F}$ be a field. Then there exists a protocol compiler that, given a pseudorandom generator $G$ with seed length $\kappa$, $s$, and a description of an $m$-party functionality expressed as a depth-$d$ arithmetic circuit $\mathrm{C}$ over $\mathbb{F}$ with constant fan-in, outputs a UC-secure $O(d)$-round $m$-party protocol realizing $f$ with total communication complexity $O(m|\mathrm{C}|) + \mathsf{poly}(m, \kappa, d)$ bits, where security holds against a static adversary corrupting a minority of parties.*

Applying the analysis of concrete parameters (see above and the full version [HVW19]) we re-derive the result of Chida et al. [CGH$^+$18] who show an overhead-2 actively-secure honest-majority protocol. Their result applies to arbitrary circuits over sufficiently large fields, whereas ours achieves overhead of 2 for sufficiently wide circuits.

## Acknowledgments

The first author is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant 1316/18. The second author is supported by Google Faculty Research Grant, NSF Award CNS-1618884 and Intelligence Advanced Research Projects Activity (IARPA) via 2019-19-020700009. The views expressed are those of the author and do not reflect the official policy or position of Google, the Department of Defense, the National Science Foundation, or the U.S. Government. The third author is supported by ISF grants 1861/16 and 1399/17, and AFOSR Award FA9550-17-1-0069.

## References

ADI$^+$17. Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO*, pages 223–254, 2017.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

CC06. Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *CRYPTO*, pages 521–536, 2006.

CCD87. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *CRYPTO*, page 462, 1987.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

CGH$^+$18. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, pages 34–64, 2018.

DGN$^+$17. Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. TinyOLE: Efficient actively secure two-party computation from oblivious linear function evaluation. In *CCS*, pages 2263–2276, 2017.

DI06. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, pages 1–18, 2013.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

FY92. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710, 1992.

GIP$^+$14. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, pages 495–504, 2014.

GIW16. Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary amd circuits from secure multiparty computation. In *TCC-B*, 2016.

GLNP15. Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *CCS*, pages 567–578, 2015.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In *CRYPTO*, pages 85–114, 2019.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

Hai08. Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In *TCC*, pages 412–426, 2008.

HIK$^+$11. Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

HIMV19. Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. LevioSA: Lightweight secure arithmetic computation. In *To appear CCS*, 2019.

HIV17. Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *TCC*, pages 3–39, 2017.

HK12. Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *J. Cryptology*, 25(1):158–193, 2012.

HKK$^+$14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, pages 458–475, 2014.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *ASIACRYPT*, pages 598–628, 2017.

HVW19. Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. *IACR Cryptology ePrint Archive*, 2019:1250, 2019.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

IKP$^+$16. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *CRYPTO*, pages 430–458, 2016.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, pages 158–189, 2018.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

LOP11. Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *CRYPTO*, pages 259–276, 2011.

LP07. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

LP12. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

LPO11. Yehuda Lindell, Benny Pinkas, and Eli Oxman. The IPS compiler: Optimizations, variants and concrete efficiency. *IACR Cryptology ePrint Archive*, 2011:435, 2011.

LPSY15. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015.

LR15. Yehuda Lindell and Ben Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *CCS*, pages 579–590, 2015.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

NO09. Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *TCC*, pages 368–386, 2009.

RR16. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 297–314, 2016.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

SS13. Abhi Shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In *CCS*, pages 523–534, 2013.

ST04. Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In *ASIACRYPT*, pages 119–136, 2004.

WMK17. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In *EUROCRYPT*, pages 399–424, 2017.

WRK17a. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37, 2017.

WRK17b. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, pages 39–56, 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015.