Guaranteed Output Delivery Comes Free in Honest Majority MPC

Vipul Goyal[1], Yifan Song[1(✉)], and Chenzhi Zhu[2]

[1] Carnegie Mellon University, Pittsburgh, USA
vipul@cmu.edu, yifans2@andrew.cmu.edu
[2] Tsinghua University, Beijing, China
mrbrtpt@gmail.com

Abstract. We study the communication complexity of unconditionally secure MPC with guaranteed output delivery over point-to-point channels for corruption threshold $t < n/2$, assuming the existence of a public broadcast channel. We ask the question: "is it possible to construct MPC in this setting s.t. the communication complexity per multiplication gate is linear in the number of parties?" While a number of works have focused on reducing the communication complexity in this setting, the answer to the above question has remained elusive until now. We also focus on the concrete communication complexity of evaluating each multiplication gate.

We resolve the above question in the affirmative by providing an MPC with communication complexity $O(Cn\phi)$ bits (ignoring fixed terms which are independent of the circuit) where ϕ is the length of an element in the field, C is the size of the (arithmetic) circuit, n is the number of parties. This is the first construction where the asymptotic communication complexity matches the best-known semi-honest protocol. This represents a strict improvement over the previously best-known communication complexity of $O(C(n\phi + \kappa) + D_M n^2 \kappa)$ bits, where κ is the security parameter and D_M is the multiplicative depth of the circuit. Furthermore, the concrete communication complexity per multiplication gate is 5.5 field elements per party in the best case and 7.5 field elements in the worst case when one or more corrupted parties have been identified. This also roughly matches the best-known semi-honest protocol, which requires 5.5 field elements per gate.

The above also yields the first secure-with-abort MPC protocol with the same cost per multiplication gate as the best-known semi-honest protocol. Our main result is obtained by compiling the secure-with-abort MPC protocol into a fully secure one.

V. Goyal—Research supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via 2019-1902070008, an NSF award 1916939, a gift from Ripple, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

Y. Song—Research supported in part by a Cylab Presidential Fellowship and grants of Vipul Goyal mentioned above.

C. Zhu—Work done in part while at CMU.

1 Introduction

In secure multiparty computation (MPC), a set of n parties together evaluate a function f on their private inputs. This function f is public to all parties, and, may be modeled as an arithmetic circuit over a finite field. Very informally, a protocol of secure multiparty computation guarantees the privacy of the inputs of every (honest) individual except the information which can be deduced from the output. This notion was first introduced in the work [Yao82] of Yao. Since the early feasibility solutions proposed in [Yao82,GMW87], various settings of MPC have been studied. Examples include semi-honest security vs malicious security, security against computational adversaries vs unbounded adversaries, honest majority vs corruptions up to $n-1$ parties, security with abort vs guaranteed output delivery and so on.

In this work, we focus on the information-theoretical setting (i.e., security against unbounded adversaries) with guaranteed output delivery. The adversary is allowed to corrupt at most $t < n/2$ parties and is fully malicious. We assume the existence of private point-to-point communication channels and a public broadcast channel. We are interested in the communication complexity of the secure MPC, which is measured by the number of bits X via private point-to-point channels and the number of bits Y via the public broadcast channel, i.e., $X + Y \cdot \mathcal{BC}$. The first positive solutions in this setting were proposed in [RBO89,Bea89]. After those, several subsequent works [CDD+99,BTH06,BSFO12] have focused on improving the communication complexity of the protocol. Note that, by representing the functionality as an arithmetic circuit, the communication complexity of the protocol in the unconditional setting is typically dominated by the number of multiplication gates in the circuit. This is because the addition gates can usually be done locally, requiring no communication at all.

In this paper, we ask the following natural question:

"Is it possible to construct unconditional MPC with guaranteed output delivery for $t < n/2$ s.t. the communication complexity per multiplication gate is linear in the number of parties? Furthermore, what is the concrete communication complexity per multiplication gate?"

Having linear communication complexity per multiplication gate greatly benefits the scalability of the protocol, as it means that the work done by each party is independent of the number of parties but only related to the size of the circuit. While a number of works have made significant progress, this question has remained opened until now.

The best-known result in this setting is the construction in the work [BSFO12] of Ben-Sasson, Fehr and Ostrovsky. The construction in [BSFO12] has communication complexity $O(C(n\phi+\kappa)+D_M n^2\kappa)$ bits (ignoring fixed terms which are independent of the circuit), where C is the size of the circuit, ϕ is the length of a field element, κ is the security parameter and D_M is the multiplicative depth of the circuit. Comparing with the best-known result against semi-honest adversaries in [DN07], which has communication complexity $O(Cn\phi)$ bits, there is an additional term $D_M n^2\kappa$ related to the circuit. In the worst case where the circuit is "narrow and deep", $D_M n^2\kappa$ may even become the dominating term of the

communication complexity and result in $O(n^2)$ elements per gate. Ben-Sasson et. al asked if this quadratic term related to the depth of the circuit is inherent.

In a beautiful work, Ishai et al. [IKP+16] provided a general transformation from a protocol in the setting of security with abort to a protocol with guaranteed output delivery. Instantiation this transformation with the best-known protocol for security with abort, the resulting construction eliminates the quadratic term w.r.t. the circuit depth. However, the communication complexity of the resulting protocol now has a term $O(W \cdot \text{poly}(n))$, where W is the width of the circuit, and, $\text{poly}(n)$ can be at least n^4 for certain circuits. For the circuit with a large width, this term may even become the dominating term.

In the setting of $t < n/3$ corruptions (where a public broadcast channel can be securely simulated), question of getting a construction with linear communication complexity was recently resolved in the recent work of Goyal et. al [GLS19], which presented a construction with communication complexity $O(Cn\phi)$ bits. Similar results were also known in the setting of security with abort in [GIP+14,GIP15,LN17,CGH+18,NV18].

Our Results. In this work, we answer the above question in the affirmative by presenting an MPC protocol with communication complexity $O(Cn\phi)$ bits (ignoring fixed terms which are independent of the circuit). Furthermore, we also focus on the concrete efficiency, i.e., the number of elements per multiplication gate per party. Concretely, our result achieves $5.5 + \epsilon$ elements in the best case and $7.5 + \epsilon$ elements in the worst case when one or more corrupted parties have been identified, where ϵ can be an arbitrarily small constant. Comparing with the best-known result [DN07] in the semi-honest setting, our result essentially shows that achieving output delivery guarantee requires no additional cost compared to the semi-honest security.

Our main contributions lie in two aspects, (1) we present the first construction in this setting where the asymptotic communication complexity matches that in the semi-honest setting, and, (2) our protocol roughly achieves the same concrete efficiency as the best-known semi-honest protocol.

The above also yields the first secure-with-abort MPC protocol with the same cost per multiplication gate as the best-known semi-honest protocol [DN07]. Concretely, each party only needs to communicate 5.5 field elements per multiplication gate. We obtain this construction by building on the technique in [BBCG+19]. An overview of the construction can be found in Section 3.

Regarding the construction with guaranteed output delivery, our results stem from the idea of developing a suite of techniques to efficiently compile our secure-with-abort protocol into a fully secure protocol. Additionally, we introduce a technique which allows us to reuse authentication keys towards developing a more efficient verifiable secret sharing scheme. An overview of our new ideas can be found in Section 4.

Related Works. We compare our result with several related constructions in both techniques and efficiency. In the following, let C denote the size of the

circuit, ϕ denote the size of a field element, κ denote the security parameter, D_M denote the depth of the circuit, and W denote the width of the circuit. We will ignore fixed terms which are independent of the circuit.

Security with abort. In [DN07], Damgård and Nielsen introduce the best-known semi-honest protocol, which we refer to as the DN protocol. The communication complexity of the DN protocol is $O(Cn\phi)$ bits. The concrete efficiency is 6 field elements per multiplication gate (per party). In [GIP+14], Genkin, et al. show that the DN protocol is secure up to an additive attack when running in the fully malicious setting. Based on this observation, a secure-with-abort MPC protocol can be constructed by combining the DN protocol and a circuit which is resilient to an additive attack (referred to as an AMD circuit). As a result, Genkin, et al. [GIP+14] give the first construction against a fully malicious adversary with communication complexity $O(Cn\phi)$ bits (for a large enough field), which matches the asymptotic communication complexity of the DN protocol.

The construction in [CGH+18] also relies on the theorem showed in [GIP+14]. The idea is to check whether the adversary launches an additive attack. In the beginning, all parties compute a random secret sharing of the value r. For each wire w with the value x associated with it, all parties will compute two secret sharings of the secret values x and $r \cdot x$ respectively. Here $r \cdot x$ can be seen as a secure MAC of x when the only possible attack is an additive attack. In this way, the protocol requires two operations per multiplication gate. The asymptotic communication complexity is $O(Cn\phi)$ bits (for a large enough field) and the concrete efficiency is reduced to 12 field elements per multiplication gate.

An interesting observation is that the theorem showed in [GIP+14] implies that the DN protocol provides perfect privacy of honest parties (before the output phase) in the presence of a fully malicious adversary. To achieve security with abort, the only task is to check the correctness of the computation before the output phase. This observation has been used in [LN17,NV18]. In particular, the construction in [NV18] achieves the same concrete efficiency as [CGH+18] by using the Batch-wise Multiplication Verification technique in [BSFO12], i.e., 12 field elements per multiplication gate. Our construction also relies on this observation. Therefore, the main task is to efficiently verify a batch of multiplications such that the communication complexity is sublinear in the number of parties.

In [BBCG+19], Boneh, et al. introduce a very powerful tool to achieve this task when the number of parties is restricted to be a constant. Our result is obtained by instantiating this technique with a different secret sharing scheme, which allows us to overcome this restriction so that it works for any (polynomial) number of parties. Furthermore, we simplify this technique by avoiding the use of a robust secret sharing scheme and a verifiable secret sharing scheme, which are required in [BBCG+19]. Our protocol additionally makes a simple optimization to the DN protocol, which brings down the cost from 6 field elements per multiplication gate to 5.5 field elements. More details about the comparison for techniques can be found in the last paragraph of Section 3.5. A subsequent work [GLOS20] implements our construction and shows that the performance beats the previously best-known implementation result [CGH+18] in this setting.

In [BGIN19], Boyle, et al. use the technique in [BBCG$^+$19] to construct a 3-party computation with guaranteed output delivery. In particular, they implement their verification for multiplication gates. As shown in their implementation result, just the local computation of checking the correctness of 1 million multiplication gates in the 31-bit Mersenne Field requires around 1 second. Note that this does not include any computation cost related to the circuit and any communication cost. On the other hand, the implementation result from [GLOS20] shows that our construction only needs 0.7 second for computing the whole circuit in an even large field (61-bit Mersenne Field) in the 3-party setting. This shows that our construction is *several* times faster.

Guaranteed Output Delivery. The construction in [BSFO12] is most related to our result. In fact, we reuse and modify many protocols in [BSFO12] in our construction.

The communication complexity achieved by the construction in [BSFO12] is $O(C(n\phi+\kappa)+D_M n^2\kappa)$ bits. Our result removes both the quadratic term related to D_M and the term $O(C\kappa)$. Furthermore, the use of Beaver triples for multiplication gates in [BSFO12] is more expensive than the multiplication protocol in the best-known semi-honest protocol [DN07]. As a result, the communication cost per multiplication gate in [BSFO12] is a fixed 20 field elements (without considering the effect of $O(D_M n^2\kappa)$). Our result achieves $5.5 + \epsilon$ field elements per multiplication gate in the best case and $7.5 + \epsilon$ field elements in the worst case when one or more corrupted parties have been identified, where ϵ can be an arbitrarily small constant. In the best case, our result matches the best-known semi-honest protocol [DN07].

Technically, while the construction from [BSFO12] uses Beaver triples to compute multiplications in the computation phase, we directly use a modified version of the multiplication protocol of the best-known protocol [DN07] from the semi-honest setting. We note that Beaver triples provide plenty of redundancy which simplifies the checking process in the computation phase. However, the use of Beaver triples unfortunately requires a verification for each layer of the circuit, which leads to the quadratic term related to D_M. On the other hand, we start from the our secure-with-abort MPC protocol, which does not make use of Beaver triples. While this idea can potentially remove the term $O(D_M n^2)$, without the redundancy provided by Beaver triples, the verification becomes difficult and even the computation cannot proceed when malicious parties refuse to participate in the computation. We will show how to tackle these difficulties in Section 4.

In [IKP$^+$16], Ishai et al. provided a general transformation from a protocol in the setting of security with abort to a protocol with guaranteed output delivery. When instantiating their transformation with our secure-with-abort protocol, the resulting protocol can achieve 5.5 field elements per multiplication gate when the *width* of the circuit is small. However, a drawback of this transformation is that the efficiency of the resulting protocol has a large dependency on the width of the circuit. Specifically, the communication complexity of the resulting

protocol contains a term $O(W \cdot \text{poly}(n, \kappa))$ (where poly is relatively large). For the circuit with a large width, this term may even become the dominating term.

Recently, Goyal et al. [GLS19] gave the first construction against 1/3 corruption such that the communication complexity per multiplication gate is linear in the number of parties. The communication complexity is $O(Cn\phi)$ bits. Since they mainly focused on the feasibility and the protocol is perfectly secure, the concrete efficiency is 66 elements per multiplication gate.

Unfortunately the techniques developed in [GLS19] fail in the setting of honest majority. Technically, we use a significantly different approach from that in [GLS19] to remove the quadratic term related to the circuit depth. The reason for $O(D_M n^2)$ is that all parties need to ensure the correctness of multiplications in one layer before moving on to the next layer. To this end, each layer requires at least $O(n^2)$ communication, which results in $O(D_M n^2)$ overhead. While Goyal et al. [GLS19] used n-out-of-n secret sharings to overcome the layer restriction, our approach is to directly compile the our secure-with-abort protocol, which does not have the term $O(D_M n^2)$, to a fully secure one.

Other Related Works. The notion of MPC was first introduced in [Yao82,GMW87] in 1980s. Feasibility results for MPC were obtained by [Yao82,GMW87,CDVdG87] under cryptographic assumptions, and by [BOGW88,CCD88] in the information-theoretic setting. Subsequently, a large number of works have focused on improving the efficiency of MPC protocols in various settings.

A series of works focus on improving the communication efficiency of MPC with output delivery guarantee in the settings with different threshold on the number of corrupted parties. In the setting where $t < n/3$, a public broadcast channel can be securely simulated and therefore, only private point-to-point communication channels are required. A rich line of works [HMP00,HM01,DN07,BTH08], [GLS19] have focused on improving the asymptotic communication complexity in this setting. In the setting where $t < (1/3 - \epsilon)n$, packed secret sharing can be used to hide a batch of values, resulting in more efficient protocols. E.g., Damgard et al. [DIK10] introduced a protocol with communication complexity $O(C \log C \log n \cdot \kappa + D_M^2 \text{poly}(n, \log C)\kappa)$ bits.

A rich line of works have also focused on the performance of MPC in practice. Many concretely efficient MPC protocols were presented in [LP12,NNOB12,FLNW17] [ABF+17,LN17,CGH+18]. All of these works emphasized the practical running time and only provided security with abort. Some of them were specially constructed for two parties [LP12,NNOB12], or three parties [FLNW17,ABF+17].

2 Preliminaries

2.1 Model

We consider a set of parties $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ where each party can provide inputs, receive outputs, and participate in the computation. For every pair of parties, there exists a secure (private and authentic) synchronous channel so that they can directly send messages to each other. Beyond that, we also assume

the existence of a secure broadcast channel, which is available to all parties. The communication complexity is measured by the number of bits X via private channels plus the number of bits Y via the broadcast channel, i.e., $X + Y \cdot \mathcal{BC}$.

We focus on functions which can be represented as arithmetic circuits over a finite field $\mathbb{F}$ (with $|\mathbb{F}| \geq n+1$) with input, addition, multiplication, random, and output gates. Let $\phi = \log |\mathbb{F}|$ be the size of an element in $\mathbb{F}$. We use κ to denote the security parameter and let $\mathbb{K}$ be an extension field of $\mathbb{F}$ (with $|\mathbb{K}| \geq 2^\kappa$). For simplicity, we use κ to denote the size of an element in $\mathbb{K}$.

An adversary is able to corrupt at most $t < n/2$ parties, provide inputs to corrupted parties, and receive all messages sent to corrupted parties. Corrupted parties can deviate from the protocol arbitrarily. For simplicity, we assume $n = 2t + 1$. Each party P_i is assigned a unique non-zero field element $\alpha_i \in \mathbb{F} \backslash \{0\}$ as the identity.

Let c_I, c_M, c_R, c_O be the numbers of input gates, multiplication gates, random gates and output gates respectively. We set $C = c_I + c_M + c_R + c_O$ to be the size of the circuit.

2.2 Secret Sharing

In our protocol, we use the standard Shamir's secret sharing scheme [Sha79].

A *degree-d* Shamir sharing of $w \in \mathbb{F}$ is a vector $(w_1, \ldots, w_n)$ which satisfies that, there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(0) = w$ and $f(\alpha_i) = w_i$ for $i \in \{1, \ldots, n\}$. Each party P_i holds a share w_i and the whole sharing is denoted by $[w]_d$.

Properties of the Shamir's Secret Sharing Scheme. In the following, we will utilize two properties of the Shamir's secret sharing scheme.

- Linear Homomorphism:

$$\forall\ [x]_d, [y]_d,\ [x + y]_d = [x]_d + [y]_d.$$

- Multiplying two degree-d sharings yields a degree-$2d$ sharing. The secret value of the new sharing is the product of the original two secrets.

$$\forall\ [x]_d, [y]_d,\ [x \cdot y]_{2d} = [x]_d \cdot [y]_d.$$

2.3 Generating Random Sharings and Double Sharings

We introduce two basic protocols Rand and DoubleRand in the DN protocol [DN07].

The protocol Rand is used to prepare $t+1 = O(n)$ random degree-t sharings in the *semi-honest* setting. Rand will utilize a predetermined and fixed Vandermonde matrix of size $n \times (t + 1)$, which is denoted by $\boldsymbol{M}^{\mathrm{T}}$ (therefore $\boldsymbol{M}$ is a $(t+1) \times n$ matrix). An important property of a Vandermonde matrix is that any $(t+1) \times (t+1)$ submatrix of $\boldsymbol{M}^{\mathrm{T}}$ is *invertible*. The description of Rand appears in Protocol 1. The communication complexity of Rand is $O(n^2)$ field elements.

Protocol 1: RAND

1. Each party $P_i \in \mathcal{P}_{\texttt{active}}$ randomly samples a sharing $[s^{(i)}]_t$ such that the shares held by parties in $\mathcal{D}isp_i$ are set to be 0. Then P_i distributes the shares to other parties. For each $P_i \in \mathcal{C}orr$, all parties take an all-0 sharing as $[s^{(i)}]_t$.
2. All parties locally compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(t+1)}]_t)^{\mathrm{T}} = \boldsymbol{M}([s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(n)}]_t)^{\mathrm{T}}$$

and output $[r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(t+1)}]_t$.

A pair of double sharings $([r]_t, [r]_{2t})$ is a pair of two sharings of the same secret. One is a degree-t sharing and the other one is a degree-$2t$ sharing. The protocol DOUBLERAND is used to prepare $t+1 = O(n)$ random double sharings in the *semi-honest* setting. The description of DOUBLERAND appears in Protocol 2. The communication complexity of DOUBLERAND is $O(n^2)$ field elements.

Protocol 2: DOUBLERAND

1. Each party $P_i \in \mathcal{P}_{\texttt{active}}$ randomly samples a pair of double sharings $([s^{(i)}]_t, [s^{(i)}]_{2t})$ such that the shares held by parties in $\mathcal{D}isp_i$ are set to be 0. Then P_i distributes the shares to other parties. For each $P_i \in \mathcal{C}orr$, all parties take all-0 sharings as $([s^{(i)}]_t, [s^{(i)}]_{2t})$.
2. All parties locally compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(t+1)}]_t)^{\mathrm{T}} = \boldsymbol{M}([s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(n)}]_t)^{\mathrm{T}}$$
$$([r^{(1)}]_{2t}, [r^{(2)}]_{2t}, \ldots, [r^{(t+1)}]_{2t})^{\mathrm{T}} = \boldsymbol{M}([s^{(1)}]_{2t}, [s^{(2)}]_{2t}, \ldots, [s^{(n)}]_{2t})^{\mathrm{T}}$$

and output $([r^{(1)}]_t, [r^{(1)}]_{2t}), ([r^{(2)}]_t, [r^{(2)}]_{2t}), \ldots, ([r^{(t+1)}]_t, [r^{(t+1)}]_{2t})$.

3 An Overview of the Secure-with-abort Protocol

3.1 General Strategy and Protocol Overview

In [GIP+14], Genkin, et al. showed that several semi-honest MPC protocols are secure up to an additive attack in the presence of a fully malicious adversary. As one corollary, these semi-honest protocols provide full privacy of honest parties before reconstructing the output. Therefore, a straightforward strategy to

achieve security-with-abort is to (1) run a semi-honest protocol till the output phase, (2) check the correctness of the computation, and (3) reconstruct the output only if the check passes.

Fortunately, the best-known semi-honest protocol in this setting [DN07] is secure up to an additive attack. Our construction will follow the above strategy. The main task is the second step, i.e., checking the correctness of the computation before reconstructing the final results.

3.2 Review: DN Semi-Honest Protocol

The best-known semi-honest protocol was proposed in the work of Damgård and Nielsen [DN07]. The protocol consists of 4 phases: Preparation Phase, Input Phase, Computation Phase, and Output Phase. Here we give a brief description of these four phases.

Preparation Phase. In the preparation phase, all parties need to prepare several random sharings which will be used in the computation phase. Specifically, there are two kinds of random sharings needed to be prepared. The first kind is a random degree-t sharing $[r]_t$. The second kind is a pair of random sharings $([r]_t, [r]_{2t})$, which is referred to as double sharings. At a high-level, these two kinds of random sharings are prepared in the following manner:

1. Each party generates and distributes a random degree-t sharing (or a pair of random double sharings).
2. Each random sharing (or each pair of double sharings) is a linear combination of the random sharings (or the random double sharings) distributed by each party.

More details can be found in Section 2.3.

Input Phase. In the input phase, each input holder generates and distributes a random degree-t sharing of its input.

Computation Phase. In the computation phase, all parties need to evaluate addition gates and multiplication gates. For an addition gate with input sharings $[x]_t, [y]_t$, all parties just locally add their shares to get $[x+y]_t = [x]_t + [y]_t$. For a multiplication gate with input sharings $[x]_t, [y]_t$, one pair of double sharings $([r]_t, [r]_{2t})$ is consumed. All parties execute the following steps.

1. All parties first locally compute $[x \cdot y + r]_{2t} = [x]_t \cdot [y]_t + [r]_{2t}$.
2. $P_{\texttt{king}}$ collects all shares of $[x \cdot y + r]_{2t}$ and reconstructs the value $x \cdot y + r$. Then $P_{\texttt{king}}$ sends the value $x \cdot y + r$ back to all other parties.
3. All parties locally compute $[x \cdot y]_t = x \cdot y + r - [r]_t$.

Here $P_{\texttt{king}}$ is the party all parties agree on in the beginning.

Output Phase. In the output phase, all parties send their shares of the output sharing to the party who should receive this result. Then that party can reconstruct the output.

Improvement to 5.5 Field Elements. We note that in the second step of the multiplication protocol, $P_{\texttt{king}}$ can alternatively generate a degree-t sharing $[x \cdot y + r]_t$ and distribute the sharing to all other parties. Then in the third step, $[x \cdot y]_t$ can be computed by $[x \cdot y + r]_t - [r]_t$. In fact, $P_{\texttt{king}}$ can set the shares of (a predetermined set of) t parties to be 0 in the sharing $[x \cdot y + r]_t$. This means that $P_{\texttt{king}}$ need not to communicate these shares at all, reducing the communication by half. We rely on the following two observations:

- While normally setting some shares to be 0 could compromise the privacy of the secret (by effectively reducing the reconstruction threshold), note that here $x \cdot y + r$ need not to be private at all.
- Parties do not actually need to receive $x \cdot y + r$ from $P_{\texttt{king}}$. Rather, receiving shares of $x \cdot y + r$ is sufficient to allow them to proceed in the protocol.

This simple observation leads to an improvement of reducing the cost per gate from 6 elements to 5.5 elements. Note that in this construction, all multiplication gates at the same "layer" in the circuit can be evaluated in parallel. Hence, it is even possible to perform a "load balancing" such that the overall cost of different parties roughly remains the same.

3.3 Review: Batch-wise Multiplication Verification

This technique is introduced in the work of Ben-Sasson, et al. [BSFO12]. It is used to check a batch of multiplication tuples efficiently. Specifically, given m multiplication tuples

$$([x^{(1)}]_t, [y^{(1)}]_t, [z^{(1)}]_t), ([x^{(2)}]_t, [y^{(2)}]_t, [z^{(2)}]_t), \ldots, ([x^{(m)}]_t, [y^{(m)}]_t, [z^{(m)}]_t),$$

we want to check whether $x^{(i)} \cdot y^{(i)} = z^{(i)}$ for all $i \in [m]$.

The high-level idea is constructing three polynomials $f(\cdot), g(\cdot), h(\cdot)$ such that

$$\forall i \in [m], f(i) = x^{(i)}, g(i) = y^{(i)}, h(i) = z^{(i)}.$$

Then check whether $f \cdot g = h$. Here $f(\cdot), g(\cdot)$ are degree-$(m-1)$ polynomials so that they can be determined by $\{x^{(i)}\}_{i\in[m]}, \{y^{(i)}\}_{i\in[m]}$ respectively. In this case, $h(\cdot)$ should be a degree-$2(m-1)$ polynomial which is determined by $2m-1$ values. To this end, for $i \in \{m+1, \ldots, 2m-1\}$, we need to compute $z^{(i)} = f(i) \cdot g(i)$ so that $h(\cdot)$ can be computed by $\{z^{(i)}\}_{i\in[2m-1]}$.

All parties first locally compute $[f(\cdot)]_t$ and $[g(\cdot)]_t$ using $\{[x^{(i)}]_t\}_{i\in[m]}$ and $\{[y^{(i)}]_t\}_{i\in[m]}$ respectively. Here a degree-t sharing of a polynomial means that each coefficient is secret-shared. For $i \in \{m+1, \ldots, 2m-1\}$, all parties locally compute $[f(i)]_t, [g(i)]_t$ and then compute $[z^{(i)}]_t$ using the multiplication protocol in [DN07]. Finally, all parties locally compute $[h(\cdot)]_t$ using $\{[z^{(i)}]_t\}_{i\in[2m-1]}$.

Note that if $x^{(i)} \cdot y^{(i)} = z^{(i)}$ for all $i \in [2m-1]$, then we have $f \cdot g = h$. Otherwise, we must have $f \cdot g \neq h$. Therefore, it is sufficient to check whether $f \cdot g = h$. Since $h(\cdot)$ is a degree-$2(m-1)$ polynomials, in the case that $f \cdot g \neq h$,

the number of x such that $f(x) \cdot g(x) = h(x)$ holds is at most $2(m-1)$. Thus, it is sufficient to test whether $f(x) \cdot g(x) = h(x)$ for a random x. As a result, this technique compresses m checks of multiplication tuples to a single check of the tuple $([f(x)]_t, [g(x)]_t, [h(x)]_t)$. Secure techniques for checking the tuple $([f(x)]_t, [g(x)]_t, [h(x)]_t)$ are given in [BSFO12,NV18].

The main drawback of this technique is that it requires one additional multiplication operation per tuple. Our idea is to improve this technique so that the check will require fewer multiplication operations.

3.4 Extensions

We would like to introduce two natural extensions of the DN multiplication protocol and the Batch-wise Multiplication Verification technique respectively.

Extension of the DN Multiplication Protocol. In essence, the DN multiplication protocol uses a pair of random double sharings to reduce a degree-$2t$ sharing $[x \cdot y]_{2t}$ to a degree-t sharing $[x \cdot y]_t$. Therefore, an extension of the DN multiplication protocol is used to compute the inner-product of two vectors of the same dimension.

Specifically, let $\odot$ denote the inner-product operation. Given two input vectors of sharings $[\boldsymbol{x}]_t, [\boldsymbol{y}]_t$, we can compute $[\boldsymbol{x} \odot \boldsymbol{y}]_t$ using the same strategy as the DN multiplication protocol and in particular, with the *same communication cost*. This is because, just like in the multiplication protocol, here all the parties can *locally* compute the shares of the result. These shares are then randomized and sent to $P_{\texttt{king}}$ for degree reduction. This extension is observed in [CGH$^+$18].

Extension of the Batch-wise Multiplication Verification. We can use the same strategy as the Batch-wise Multiplication Verification to check the correctness of a batch of *inner-product* tuples.

Specifically, given a set of m inner-product tuples $\{([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[m]}$, we want to check whether $\boldsymbol{x}^{(i)} \odot \boldsymbol{y}^{(i)} = z^{(i)}$ for all $i \in [m]$. Here $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i\in[m]}$ are vectors of the same dimension. The only difference is that all parties will compute $\boldsymbol{f}(\cdot), \boldsymbol{g}(\cdot)$ such that

$$\forall i \in [m], \boldsymbol{f}(i) = \boldsymbol{x}^{(i)}, \boldsymbol{g}(i) = \boldsymbol{y}^{(i)},$$

and all parties need to compute $[z^{(i)}]_t = [\boldsymbol{f}(i) \odot \boldsymbol{g}(i)]_t$ for all $i \in \{m+1, \ldots, 2m-1\}$, which can be done by the extension of the DN multiplication protocol. Let $h(\cdot)$ be a degree-$2(m-1)$ polynomial such that

$$\forall i \in [2m-1], h(i) = z^{(i)}.$$

Then, it is sufficient to test whether $\boldsymbol{f}(x) \odot \boldsymbol{g}(x) = h(x)$ for a random x. As a result, this technique compresses m checks of inner-product tuples to a single check of the tuple $([\boldsymbol{f}(x)]_t, [\boldsymbol{g}(x)]_t, [h(x)]_t)$. It is worth noting that the communication cost remains the *same* as the original technique. This extension is observed in [NV18].

Using these Extensions for Reducing the Field Size. We point out that these extensions are not used in any way in the main results of [CGH+18,NV18]. In [CGH+18], the primary purpose of the extension is to check more efficiently in a small field. In more detail, [CGH+18] has a "secure MAC" associated with each wire value in the circuit. At a later point, the MACs are verified by computing a linear combination of the value-MAC pairs with random coefficients. Unlike the case in a large field, the random coefficients cannot be made public due to security reasons. Then a computation of a linear combination becomes a computation of an inner-product. [CGH+18] relies on the extension of the DN multiplication protocol to efficiently compute the inner-product of two vector of sharings. However we note that with the decrease in the field size, the number of field elements required per gate grows up and hence the concrete efficiency goes down. In [NV18], the extension of the Batch-wise Multiplication Verification technique is only pointed out as a corollary of independent interest.

3.5 Fast Verification for a Batch of Multiplication Tuples

Now we are ready to present our technique. Suppose the multiplication tuples we want to verify are

$$([x^{(1)}]_t, [y^{(1)}]_t, [z^{(1)}]_t), ([x^{(2)}]_t, [y^{(2)}]_t, [z^{(2)}]_t), \ldots, ([x^{(m)}]_t, [y^{(m)}]_t, [z^{(m)}]_t).$$

The starting idea is to transform these m multiplication tuples into one inner-product tuple. A straightforward way is just setting

$$\begin{aligned}
[\boldsymbol{x}]_t &= ([x^{(1)}]_t, [x^{(2)}]_t, \ldots, [x^{(m)}]_t) \\
[\boldsymbol{y}]_t &= ([y^{(1)}]_t, [y^{(2)}]_t, \ldots, [y^{(m)}]_t) \\
[z]_t &= \sum_{i=1}^{m} [z^{(i)}]_t.
\end{aligned}$$

However, it is insufficient to check this tuple. For example, if corrupted parties only maliciously behave when computing the first two tuples and cause $z^{(1)}$ to be $x^{(1)} \cdot y^{(1)} + 1$ and $z^{(2)}$ to be $x^{(2)} \cdot y^{(2)} - 1$, we cannot detect it by using this approach. We need to add some randomness so that the resulting tuple will be incorrect with overwhelming probability if any one of the original tuples is incorrect.

Step One: De-Linearization. Our idea is to use two polynomials with coefficients $\{x^{(i)} \cdot y^{(i)}\}$ and $\{z^{(i)}\}$ respectively. Concretely, let

$$\begin{aligned}
F(X) &= (x^{(1)} \cdot y^{(1)}) + (x^{(2)} \cdot y^{(2)})X + \ldots + (x^{(m)} \cdot y^{(m)})X^{m-1} \\
G(X) &= z^{(1)} + z^{(2)}X + \ldots + z^{(m)}X^{m-1}.
\end{aligned}$$

Then if at least one multiplication tuple is incorrect, we will have $F \neq G$. In this case, the number of x such that $F(x) = G(x)$ is at most $m - 1$. Therefore, with overwhelming probability, $F(r) \neq G(r)$ where r is a random element.

All parties will generate a random degree-t sharing $[r]_t$ in the same way as that in the preparation phase of the DN protocol. Then they reconstruct the value r. We can set

$$\begin{aligned}
[\boldsymbol{x}]_t &= ([x^{(1)}]_t, r[x^{(2)}]_t, \ldots, r^{m-1}[x^{(m)}]_t) \\
[\boldsymbol{y}]_t &= ([y^{(1)}]_t, [y^{(2)}]_t, \ldots, [y^{(m)}]_t) \\
[z]_t &= \sum_{i=1}^{m} r^{i-1}[z^{(i)}]_t.
\end{aligned}$$

Then $F(r) = \boldsymbol{x} \odot \boldsymbol{y}$ and $G(r) = z$. The inner-product tuple $([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [z]_t)$ is what we wish to verify.

Step Two: Dimension-Reduction. Although we only need to verify the correctness of a single inner-product tuple, it is unclear how to do it efficiently. It seems that verifying an inner-product tuple with dimension m would require communicating at least $O(mn)$ field elements. Therefore, instead of directly doing the check, we want to first reduce the dimension of this inner-product tuple.

Towards that end, even though we only have a single inner-product tuple, we will try to take advantage of batch-wise verification of inner-product tuples. Let k be a compression parameter. Our goal is to transform the original tuple of dimension m to be a new tuple of dimension m/k.

To utilize the extension, let $\ell = m/k$ and

$$\begin{aligned}
[\boldsymbol{x}]_t &= ([\boldsymbol{a}^{(1)}]_t, [\boldsymbol{a}^{(2)}]_t, \ldots, [\boldsymbol{a}^{(k)}]_t) \\
[\boldsymbol{y}]_t &= ([\boldsymbol{b}^{(1)}]_t, [\boldsymbol{b}^{(2)}]_t, \ldots, [\boldsymbol{b}^{(k)}]_t),
\end{aligned}$$

where $\{\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}\}_{i\in[k]}$ are vectors of dimension ℓ. For each $i \in [k-1]$, we compute $[c^{(i)}]_t = [\boldsymbol{a}^{(i)} \odot \boldsymbol{b}^{(i)}]_t$ using the extension of the DN multiplication protocol. Then set $[c^{(k)}]_t = [z]_t - \sum_{i=1}^{k-1}[c^{(i)}]_t$. In this way, if the original tuple is incorrect, then at least one of the new inner-product tuples is incorrect.

Finally, we use the extension of the Batch-wise Multiplication Verification technique to compress the check of these k inner-product tuples into one check of a single inner-product tuple. In particular, the resulting tuple has dimension $\ell = m/k$.

Note that the cost of this step is $O(k)$ inner-product operations, which is just $O(k)$ multiplication operations, and a reconstruction of a sharing, which requires $O(n^2)$ elements. After this step, our task is reduced from checking the correctness of an inner-product tuple of dimension m to checking the correctness of an inner-product tuple of dimension ℓ.

Step Three: Recursion and Randomization. We can repeat the second step $\log_k m$ times so that we only need to check the correctness of *a single* multiplication tuple in the end. To simplify the checking process for the last tuple, we make use of additional randomness.

In the last call of the second step, we need to compress the check of k multiplication tuples into one check of a single multiplication tuple. We include an

additional random multiplication tuple as a random mask of these k multiplication tuples. That is, we will compress the check of $k+1$ multiplication tuples in the last call of the second step. In this way, to check the resulting multiplication tuple, all parties can simply reconstruct the sharings and check whether the multiplication is correct. This reconstruction reveals no additional information about the original inner-product tuple because of this added randomness.

The random multiplication tuple is prepared in the following manner.

1. All parties prepare two random sharings $[a]_t, [b]_t$ in the same way as that in the preparation phase of the DN protocol.
2. All parties compute $[c]_t = [a \cdot b]_t$ using the DN multiplication protocol.

Efficiency Analysis. Note that each step of compression requires $O(k)$ inner-product (or multiplication) operations, which requires $O(kn)$ field elements. Also, each step of compression requires to reconstruct a random sharing, which requires $O(n^2)$ field elements. Therefore, the total amount of communication of verifying m multiplication tuples is $O((kn+n^2)\cdot\log_k m)$ field elements. Since the number of multiplication tuples m is bounded by $\mathsf{poly}(\kappa)$ where κ is the security parameter. If we choose $k = \kappa$, then the cost is just $O(\kappa n + n^2)$ field elements, which is independent of the number of multiplication tuples.

Therefore, the communication complexity per gate of our construction is the same as the DN semi-honest protocol.

Theorem 1. *Let n be the number of parties, κ be the security parameter and $k \in \mathbb{N}^\star$ be the compression factor. Let $\mathbb{F}$ be a finite field where $|\mathbb{F}| \geq n+1$, and ϕ be the size of a field element. Then, for any arithmetic circuit* `Circuit` *of size C over $\mathbb{F}$, there exists an n-party MPC protocol which securely (with abort) computes* `Circuit` *against a fully malicious adversary which controls up to $t \leq n/2$ parties. The communication complexity is $O(Cn\phi+(kn+n^2)\cdot\log_k C\cdot\kappa)$ bits. The concrete efficiency is 5.5 field elements per party per multiplication gate.*

We refer the readers to [GS20] for the detailed construction and the security proof.

Remark 1. An attractive feature of our approach is that the communication cost is not affected by the field size. To see this, note that the cost of our check only has a sub-linear dependence on the circuit size. Therefore, we can run the check over an extension field of the original field with large enough size, which does not influence the concrete efficiency of our construction.

As a comparison, the concrete efficiency of both constructions [CGH+18,NV18] suffer if one uses a small field. This is because in both constructions, the failure probability of the verification depends on the size of the field. For a small field, they need to do the verification several times to acquire the desired security. The same trick does not work because the cost of their checks has a linear dependency on the circuit size.

Remark 2. Compared with the constructions in [CGH+18,NV18], we also remove unnecessary checks to make the protocol as succinct as possible. Specifically, this

new technique of verifying a batch of multiplication tuples is the only check in the protocol and the remaining parts are the same as the DN protocol. In particular, we do not check the consistency/validity of the sharings.

Relation with the Technique in [BBCG$^+$19]. We note that our idea is similar to the technique in [BBCG$^+$19] when it is used to construct MPC protocols. When $n = 3$ and $t = 1$, our construction is very similar to the construction in [BBCG$^+$19]. For a general n-party setting, the construction in [BBCG$^+$19] relies on the replicated secret sharings and builds upon the sublinear distributed zero knowledge proofs constructed in [BBCG$^+$19]. However, the computation cost of the replicated secret sharings goes exponentially in the number of parties. This restricts the construction in [BBCG$^+$19] to only work for a constant number of parties. On the other hand, we explore the use of the Shamir secret sharing scheme in the n-party setting. Our idea is inspired by the extensions of the DN multiplication protocol [DN07,CGH$^+$18] and the Batch-wise Multiplication Verification [BSFO12,NV18]. This allows us to get a positive result without relying on replicated secret sharings. We also note that the construction in [BBCG$^+$19] requires the sharings (related to the distributed zero knowledge proof) to be robust and verifiable. We simplify this technique by removing the use of a robust secret sharing scheme and a verifiable secret sharing scheme.

Moreover, we explore the recursion trick to further improve the communication complexity of verifying multiplications. Compared with the construction in [BBCG$^+$19] which requires to communicate $O(\sqrt{C})$ bits, we achieve $O((kn + n^2) \cdot \log_k C \cdot \kappa)$ bits. Our protocol additionally makes a simple optimization to the DN protocol, which brings down the cost from 6 field elements per multiplication to 5.5 field elements.

4 An Overview of the Protocol with Guaranteed Output Delivery

We observe that our secure-with-abort MPC protocol does not have the factor $O(D_M n^2)$ in the communication complexity. Therefore, our starting idea is to compile our secure-with-abort protocol into one with guaranteed output delivery. Hopefully, it will help us remove the factor $O(D_M n^2)$ and achieve the same concrete efficiency as the semi-honest setting.

However, we then realize two problems with this idea. The most direct problem of using our secure-with-abort protocol is that a single error leads to an abort of the whole computation. However, our purpose is to build a protocol with guaranteed output delivery, which should ensure the success of the computation no matter how corrupted parties behave. It means that, when facing a failure in the check of the ultimate multiplication tuple in the last step of the multiplication verification, we need to find out where things went wrong and be able to proceed the computation.

Another problem is that, when a corrupted party maliciously refuses to participate in the computation or an identified corrupted party is kicked out

from the computation, the DN protocol cannot even proceed. This is because in the DN multiplication protocol, $P_{\texttt{king}}$ needs to reconstruct a degree-$2t$ sharing $[e]_{2t} := [x]_t \cdot [y]_t + [r]_{2t}$. $P_{\texttt{king}}$ needs $2t+1 = n$ shares to do reconstruction. This cannot be achieved if some party does not send its share to $P_{\texttt{king}}$.

In the following, we will tackle these two problems.

4.1 Efficient Verification Using Virtual Transcripts

Recall that in our secure-with-abort protocol, all parties together check the correctness of a single multiplication tuple in the last step of the multiplication verification (i.e., Step Three: Recursion and Randomization). We refer to this multiplication tuple as the *ultimate multiplication tuple*.

To be able to identify the corrupted parties that deviate from the protocol when a failure occurs in the check of the ultimate multiplication tuple, our idea is to compute a *virtual transcript* of the ultimate multiplication tuple. A virtual transcript can be seen as the transcript where all parties directly compute the ultimate multiplication tuple using the DN multiplication protocol. Although the transcript does not correspond to a real execution, all parties should agree on the messages they sent in a virtual transcript. In the case that a failure occurs in the check of the ultimate multiplication tuple, all parties can open the whole virtual transcripts to identify the parties which behaved maliciously.

We first recall the extension of the Batch-wise Multiplication Verification [NV18].

Extension of the Batch-wise Multiplication Verification [NV18]. Suppose we have ℓ inner-product tuples $\{([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [z^{(i)}]_t)\}_{i=1}^{\ell}$ and would like to verify whether $z^{(i)} = \boldsymbol{x}^{(i)} \odot \boldsymbol{y}^{(i)}$ for all $i \in [\ell]$. The extension of Batch-wise Multiplication Verification [NV18] works as follows.

1. Let $\boldsymbol{F}(\cdot), \boldsymbol{G}(\cdot)$ be two vectors of degree-$(\ell-1)$ polynomials such that

   $$\forall i \in [\ell], \quad \boldsymbol{F}(i) = \boldsymbol{x}^{(i)},\ \boldsymbol{G}(i) = \boldsymbol{y}^{(i)}.$$

   All parties can locally compute the shares of $[\boldsymbol{F}(\cdot)]_t$ and $[\boldsymbol{G}(\cdot)]_t$ by using their shares of $[\boldsymbol{x}^{(1)}]_t, \ldots, [\boldsymbol{x}^{(\ell)}]_t$ and $[\boldsymbol{y}^{(1)}]_t, \ldots, [\boldsymbol{y}^{(\ell)}]_t$, i.e., by doing interpolation on their own vectors of shares.
2. All parties compute $[\boldsymbol{x}^{(i)}]_t = [\boldsymbol{F}(i)]_t, [\boldsymbol{y}^{(i)}]_t = [\boldsymbol{G}(i)]_t$ for all $i \in \{\ell+1, \ldots, 2\ell-1\}$.
3. For all $i \in \{\ell+1, \ldots, 2\ell-1\}$, all parties compute $[z^{(i)}]_t$ where $z^{(i)} = \boldsymbol{x}^{(i)} \odot \boldsymbol{y}^{(i)}$ using the extension of the DN multiplication protocol.
4. Let $H(\cdot)$ be a degree-$2(\ell-1)$ polynomial such that

   $$\forall i \in [2\ell-1], \quad H(i) = z^{(i)}.$$

   All parties can locally compute the shares of $[H(\cdot)]_t$ by using their shares of $[z^{(1)}]_t, \ldots, [z^{(2\ell-1)}]_t$, i.e., by doing interpolation on their own shares.

Note that if all inner-product tuples $\{([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [z^{(i)}]_t)\}_{i=1}^{2\ell-1}$ are correct, we should have $\boldsymbol{F} \odot \boldsymbol{G} = H$. Otherwise, $\boldsymbol{F} \odot \boldsymbol{G} \neq H$, and the number of λ such that $\boldsymbol{F}(\lambda) \odot \boldsymbol{G}(\lambda) = H(\lambda)$ is bounded by $2(\ell-1)$. Therefore, to verify the original ℓ inner-product tuples, it is sufficient to sample a random point λ and only verify $([\boldsymbol{F}(\lambda)]_t, [\boldsymbol{G}(\lambda)]_t, [H(\lambda)]_t)$. We refer to $([\boldsymbol{F}(\lambda)]_t, [\boldsymbol{G}(\lambda)]_t, [H(\lambda)]_t)$ as the final inner-product tuple.

Preparing Virtual Transcript for the Final Inner-product Tuple. We note that the transcript of the extension of the DN multiplication protocol contains 7 sharings

$$([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [r]_t, [r]_{2t}, [e]_{2t}, [e]_t, [z]_t).$$

Here $([r]_t, [r]_{2t})$ is a pair of double sharings, $[e]_{2t} := [\boldsymbol{x}]_t \odot [\boldsymbol{y}]_t + [r]_{2t}$ is the degree-$2t$ sharing sent to $P_{\texttt{king}}$, $[e]_t$ is the degree-t sharing distributed by $P_{\texttt{king}}$, and $[z]_t = [e]_t - [r]_t$ is the output sharing of the DN multiplication protocol.

The idea of the virtual transcript is to recover the missing parts $[r]_t, [r]_{2t}, [e]_{2t}, [e]_t$. Therefore, in the case that the check of the final inner-product tuple fails, by examining the corresponding virtual transcripts, we can find out where things went wrong and potentially identify a corrupted party.

Recall that the final inner-product tuple $([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [z]_t)$ is derived by using polynomial interpolation on $2\ell-1$ inner-product tuples. In a similar way, we derive $[r]_t, [r]_{2t}, [e]_{2t}, [e]_t$ by polynomial interpolation on the corresponding values in the transcripts of these $2\ell-1$ inner-product tuples.

In more detail, given the transcripts of the original m inner-product tuples

$$\{([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t, [z^{(i)}]_t)\}_{i=1}^{\ell},$$

we want to compute the transcript of the resulting tuple.

Let $\{([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t, [z^{(i)}]_t)\}_{i=\ell+1}^{2\ell-1}$ denote the transcripts generated in the extension of the Batch-wise Multiplication Verification. Recall that $[\boldsymbol{F}(\cdot)]_t, [\boldsymbol{G}(\cdot)]_t, [H(\cdot)]_t$ satisfy that

$$\forall i \in [2\ell-1]: \ [\boldsymbol{F}(i)]_t = [\boldsymbol{x}^{(i)}]_t, \ [\boldsymbol{G}(i)]_t = [\boldsymbol{y}^{(i)}]_t, \ [H(i)]_t = [z^{(i)}]_t.$$

Let $[R(\cdot)]_t, [R(\cdot)]_{2t}, [E(\cdot)]_{2t}, [E(\cdot)]_t$ be sharings of polynomials of degree $2(\ell-1)$ such that

$$\begin{aligned} \forall i \in [2\ell-1]: \quad & [R(i)]_t = [r^{(i)}]_t, \ [R(i)]_{2t} = [r^{(i)}]_{2t}, \\ & [E(i)]_{2t} = [e^{(i)}]_{2t}, \ [E(i)]_t = [e^{(i)}]_t. \end{aligned}$$

Therefore, we have $[E(\cdot)]_{2t} = [\boldsymbol{F}(\cdot)]_t \odot [\boldsymbol{G}(\cdot)]_t + [R(\cdot)]_{2t}$ and $[H(\cdot)]_t = [E(\cdot)]_t - [R(\cdot)]_t$. It means that, for every λ, one can regard

$$([\boldsymbol{F}(\lambda)]_t, [\boldsymbol{G}(\lambda)]_t, [R(\lambda)]_t, [R(\lambda)]_{2t}, [E(\lambda)]_{2t}, [E(\lambda)]_t, [H(\lambda)]_t)$$

as a transcript of the following steps:

1. All parties first locally compute $[E(\lambda)]_{2t} := [\boldsymbol{F}(\lambda)]_t \odot [\boldsymbol{G}(\lambda)]_t + [R(\lambda)]_{2t}$.

2. $P_{\texttt{king}}$ collects all shares of $[E(\lambda)]_{2t}$ and reconstructs the secret $E(\lambda)$. Then $P_{\texttt{king}}$ generates a degree-t sharing $[E(\lambda)]_t$ and distributes the shares to all other parties.
3. All parties locally compute $[H(\lambda)]_t = [E(\lambda)]_t - [R(\lambda)]_t$.

To this end, all parties locally compute the shares of $[R(\cdot)]_t, [R(\cdot)]_{2t}$ by using their shares of $[r^{(1)}]_t, \ldots, [r^{(2\ell-1)}]_t$ and $[r^{(1)}]_{2t}, \ldots, [r^{(2\ell-1)}]_{2t}$. Then set $[E(\cdot)]_{2t} = [\boldsymbol{F}(\cdot)]_t \odot [\boldsymbol{G}(\cdot)]_t + [R(\cdot)]_{2t}$ and $[E(\cdot)]_t = [H(\cdot)]_t + [R(\cdot)]_t$. $P_{\texttt{king}}$ further computes $[E(\cdot)]_{2t}$ by using the sharings $[e^{(1)}]_{2t}, \ldots, [e^{(2m-1)}]_{2t}$ it received, and $[E(\cdot)]_t$ by using the sharings $[e^{(1)}]_t, \ldots, [e^{(2m-1)}]_t$ it distributed.

All parties generate a random element λ as challenge. The transcript

$$([\boldsymbol{F}(\lambda)]_t, [\boldsymbol{G}(\lambda)]_t, [R(\lambda)]_t, [R(\lambda)]_{2t}, [E(\lambda)]_{2t}, [E(\lambda)]_t, [H(\lambda)]_t)$$

is what we want to verify.

Preparing Virtual Transcript for the Ultimate Multiplication Tuple. We will follow the multiplication verification in our secure-with-abort protocol and prepare a virtual transcript for the tuple generated in each step. Suppose the transcripts of the original m multiplication tuples are

$$\{([x^{(i)}]_t, [y^{(i)}]_t, [r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t, [z^{(i)}]_t)\}_{i=1}^m,$$

and we want to verify that $z^{(i)} = x^{(i)} \cdot y^{(i)}$ for all $i \in [m]$.

Step One: De-Linearization. Recall that in Step One, all parties first generate a random element λ and set

$$\begin{aligned}
[\boldsymbol{x}]_t &= ([x^{(1)}]_t, \lambda[x^{(2)}]_t, \ldots, \lambda^{m-1}[x^{(m)}]_t) \\
[\boldsymbol{y}]_t &= ([y^{(1)}]_t, [y^{(2)}]_t, \ldots, [y^{(m)}]_t) \\
[z]_t &= \sum_{i=1}^{m} \lambda^{i-1}[z^{(i)}]_t.
\end{aligned}$$

The virtual transcript for $([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [z]_t)$ can be prepared by setting

$$([r]_t, [r]_{2t}, [e]_{2t}, [e]_t) = \sum_{i=1}^{m} \lambda^{i-1}([r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t).$$

The transcript $([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [r]_t, [r]_{2t}, [e]_{2t}, [e]_t, [z]_t)$ is what we need to verify. Note that this transcript corresponds to a single inner-product tuple of dimension m.

Step Two: Dimension-Reduction. Recall that in Step Two, we want to reduce the dimension of the inner-product tuple from Step One. Let

$$([\boldsymbol{x}]_t, [\boldsymbol{y}]_t, [r]_t, [r]_{2t}, [e]_{2t}, [e]_t, [z]_t)$$

denote the transcript. Recall that $[\boldsymbol{x}]_t, [\boldsymbol{y}]_t$ are first chopped into k equal parts:

$$[\boldsymbol{x}]_t = ([\boldsymbol{x}^{(1)}]_t, [\boldsymbol{x}^{(2)}]_t, \ldots, [\boldsymbol{x}^{(k)}]_t)$$
$$[\boldsymbol{y}]_t = ([\boldsymbol{y}^{(1)}]_t, [\boldsymbol{y}^{(2)}]_t, \ldots, [\boldsymbol{y}^{(k)}]_t),$$

where $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i\in[k]}$ are vectors of dimension m/k. For each $i \in [k-1]$, all parties compute $[z^{(i)}]_t = [\boldsymbol{x}^{(i)} \odot \boldsymbol{y}^{(i)}]_t$ by using the extension of the DN multiplication protocol. Let $([r^{(i)}]_t, [r^{(i)}]_{2t})$ be the corresponding double sharings used by the parties, $[e^i]_{2t}, [e^{(i)}]_t$ be the sharings which $P_{\texttt{king}}$ received and sent respectively. Hence,

$$([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t, [z^{(i)}]_t)$$

denote the transcript for the inner-product tuple $([\boldsymbol{x}^{(i)}]_t, [\boldsymbol{y}^{(i)}]_t, [z^{(i)}]_t)$. So far, we have only used $[\boldsymbol{x}]_t, [\boldsymbol{y}]_t$ from the input inner-product tuple. To ensure that if the input transcript of the inner-product tuple is incorrect, then one of the new generated transcripts is also incorrect, the transcript of the last tuple is computed from the input transcript. By setting

$$\begin{aligned}&([r^{(k)}]_t, [r^{(k)}]_{2t}, [e^{(k)}]_{2t}, [e^{(k)}]_t, [z^{(k)}]_t)\\ =\ &([r]_t, [r]_{2t}, [e]_{2t}, [e]_t, [z]_t) - \sum_{i=1}^{k-1}([r^{(i)}]_t, [r^{(i)}]_{2t}, [e^{(i)}]_{2t}, [e^{(i)}]_t, [z^{(i)}]_t),\end{aligned}$$

the transcript for $([\boldsymbol{x}^{(k)}]_t, [\boldsymbol{y}^{(k)}]_t)$ is

$$([\boldsymbol{x}^{(k)}]_t, [\boldsymbol{y}^{(k)}]_t, [r^{(k)}]_t, [r^{(k)}]_{2t}, [e^{(k)}]_{2t}, [e^{(k)}]_t, [z^{(k)}]_t).$$

Now we can use the extension of the Batch-wise Multiplication Verification [NV18] to compress these k transcripts of inner-product tuples into one transcript of a single inner-product tuple as we described above.

Step Three: Recursion and Randomization. In this step, all parties first recursively invoke Step Two to reduce the dimension of the inner-product tuple from m to k. In the meantime, all parties will also recursively prepare the virtual transcripts.

All parties then prepare a random multiplication tuple, and include this tuple when doing the last call of the compression. After all parties prepare this random multiplication tuple and its transcript, all parties can do the same way as that in Step Two to get a transcript of a single multiplication tuple. Let

$$([x^\star]_t, [y^\star]_t, [r^\star]_t, [r^\star]_{2t}, [e^\star]_{2t}, [e^\star]_t, [z^\star]_t)$$

denote the transcript for the ultimate multiplication tuple. It can be regarded as the transcript where all parties run the following steps:

1. All parties first locally compute $[e^\star]_{2t} := [x^\star]_t \cdot [y^\star]_t + [r^\star]_{2t}$.
2. $P_{\texttt{king}}$ collects all shares of $[e^\star]_{2t}$ and reconstructs the secret $e^\star$. Then $P_{\texttt{king}}$ generates a degree-t sharing $[e^\star]_t$ and distributes the shares to all other parties.
3. All parties locally compute $[z^\star]_t = [e^\star]_t - [r^\star]_t$.

Checking the Virtual Transcript. Recall that all parties have opened $[x^\star]_t$, $[y^\star]_t, [z^\star]_t$ to verify the ultimate multiplication tuple. In the case that $([x^\star]_t, [y^\star]_t, [z^\star]_t)$ is not a correct multiplication tuple, all parties will publish their shares of $[r^\star]_t, [r^\star]_{2t}$, $[e^\star]_{2t}, [e^\star]_t$. In addition, $P_{\texttt{king}}$ will publish the whole sharing $[e^\star]_{2t}$ it received and the whole sharing $[e^\star]_t$ it distributed. Then all parties must observe one of the following cases:

- The input sharings $[x^\star]_t, [y^\star]_t$ are inconsistent.
- The pair of double sharings $([r^\star]_t, [r^\star]_{2t})$ is incorrect or inconsistent.
- Some party P_i does not follow the protocol.
- Two parties $(P_i, P_{\texttt{king}})$ do not agree on the message sent from one party to the other party.

For the first two cases, there will be another protocol to help find errors. The main observation is that each sharing $[x]_t$ can be decomposed into $[x]_t = \sum_{i=1}^{n} [x(i)]_t$ where $[x(i)]_t$ is a linear combination of the sharings dealt by P_i. In other words, P_i should be responsible for the consistency of $[x(i)]_t$. Therefore, all parties will check each $[x(i)]_t$ to find errors.

For the last two cases, we can immediately identify a corrupted party or a pair of parties which have conflict with each other. We refer to this pair of parties as a pair of *disputed parties*.

In summary, all parties will finally identify either a corrupted party or a pair of disputed parties.

4.2 Relying on a Small Surgery to Proceed

Now suppose a corrupted party causes the computation to fail and has been identified using the described checks. What do we do? A straightforward idea is to restart the whole computation with the corrupted party excluded and a smaller corruption threshold. In the worst case, however, we may need to rerun the whole protocol $O(n)$ times, which is too expensive. To reduce the penalty due to failures, we rely on Dispute Control [BTH06], which is a general strategy to achieve unconditional security efficiently.

At a high-level, the whole circuit will be partitioned into several small segments. These segments will be evaluated in sequence. In the case that a failure occurs, the computation of this segment is discarded and all parties restart to evaluate the current segment. In other words, the end of each segment is served as a checkpoint. However, one problem with this strategy is that we cannot easily restart the computation with a smaller corruption threshold. This is because all the input sharings, which come from the end of last segment, are shared using the threshold t. Changing threshold means that one need to re-share all the input sharings. In fact, it is the main reason of the factor of $O(W \cdot \mathsf{poly}(n))$ in [IKP+16], where W is the width of the circuit.

To avoid the expensive re-sharing process, we would like to keep the corruption threshold unchanged. Furthermore, we also want to keep the influence on the concrete efficiency as little as possible. To be able to let the protocol proceed

without changing the corruption threshold, our idea is to prepare the shares held by identified corrupted parties so that $P_{\texttt{king}}$ will have enough shares to reconstruct a degree-$2t$ sharing.

Notation. Recall that n is the number of all parties and t is the number of corrupted parties. We have $n = 2t + 1$. Let $\mathcal{P}$ be the set of all parties, $\mathcal{C}orr$ be the set of parties which have been identified as corrupted parties so far, and $\mathcal{P}_{\texttt{active}} = \mathcal{P} \backslash \mathcal{C}orr$ be the set of remaining parties. If a party is identified as a corrupted party, it will not participate in the rest of the computations. Hereafter, we use *all parties* to refer parties in $\mathcal{P}_{\texttt{active}}$.

Overview. Recall that for each multiplication gate with input sharings $([x]_t, [y]_t)$, all parties first prepare a pair of random double sharings $([r]_t, [r]_{2t})$. Then all parties execute the following steps to compute $[x \cdot y]_t$.

1. All parties first locally compute $[e]_{2t} := [x]_t \cdot [y]_t + [r]_{2t}$.
2. $P_{\texttt{king}}$ collects all shares of $[e]_{2t}$ and reconstructs the secret e. Then $P_{\texttt{king}}$ generates a degree-t sharing $[e]_t$ and distributes the shares to all other parties.
3. All parties locally compute $[x \cdot y]_t = [e]_t - [r]_t$.

In our construction, when a party P_d needs to generate a random sharing, we require that the shares held by parties in $\mathcal{C}orr$ should be 0. Note that, it does not break the secrecy of the random sharing since parties in $\mathcal{C}orr$ are corrupted. We observe the following two facts.

1. During the generation process of $([r]_t, [r]_{2t})$, each dealer sets the shares held by parties in $\mathcal{C}orr$ to be 0. Since $([r]_t, [r]_{2t})$ is a linear combination of the double sharings dealt by each party, the shares of $([r]_t, [r]_{2t})$ held by parties in $\mathcal{C}orr$ are all 0.
2. For each party P_i, if the i-th share of either $[x]_t$ or $[y]_t$ is 0, then the i-th share of $[x \cdot y]_{2t} := [x]_t \cdot [y]_t$ is also 0.

Our idea is doing a small "surgery" to one input sharing $[x]_t$. Roughly speaking, this means changing the shares of $[x]_t$ held by parties in $\mathcal{C}orr$ to 0 while keeping the secret value x. Let $[\tilde{x}]_t$ denote the sharing after the "surgery". Then, it satisfies that $\tilde{x} = x$ and the shares of $[\tilde{x}]_t$ held by parties in $\mathcal{C}orr$ are 0. Detailed procedure for this "surgery" will be introduced at a later point.

Recall that the shares of $[\tilde{x}]_t, [r]_{2t}$ held by parties in $\mathcal{C}orr$ are 0. Now, when we invoke the DN multiplication protocol on $([\tilde{x}]_t, [y]_t)$, the shares of $[e]_{2t} := [\tilde{x}]_t \cdot [y]_t + [r]_{2t}$ held by parties in $\mathcal{C}orr$ are also 0. Therefore, $P_{\texttt{king}}$ can reconstruct $[e]_{2t}$ by setting the shares held by parties in $\mathcal{C}orr$ to be 0. Thus, each multiplication can be evaluated in two steps, (1) doing a small "surgery" to $[x]_t$, and (2) invoking the DN multiplication protocol on $([\tilde{x}]_t, [y]_t)$. We refer to the first step as Refresh and the second step as PartialMult.

REFRESH*: Performing the "Surgery".* Since parties in $\mathcal{C}orr$ are all corrupted, there is no need to protect the secrecy of their shares. The high-level idea is letting $P_{\texttt{king}}$ learn the shares of $[x]_t$ held by parties in $\mathcal{C}orr$. Then $P_{\texttt{king}}$ distributes a random degree-t sharing $[o]_t$ such that $o = 0$ and the shares of $[o]_t, [x]_t$ held by parties in $\mathcal{C}orr$ are the same. Therefore $[\tilde{x}]_t := [x]_t - [o]_t$ is what we need.

In more detail, all parties first prepare a random degree-t sharing $[r]_t$ (as that in the DN protocol). Recall that, in the generation process of $[r]_t$, each dealer sets the shares of parties in $\mathcal{C}orr$ to be 0. Therefore, the shares of $[r]_t$ held by parties in $\mathcal{C}orr$ are 0. Then, all parties run the following steps.

1. All parties locally compute $[e]_t := [x]_t + [r]_t$. Note that the shares of $[e]_t, [x]_t$ held by parties in $\mathcal{C}orr$ are the same.
2. $P_{\texttt{king}}$ collects all shares of $[e]_t$ and computes the shares held by parties in $\mathcal{C}orr$.
3. $P_{\texttt{king}}$ generates and distributes a random degree-t sharing $[o]_t$ where $o = 0$ and the shares of $[o]_t, [e]_t$ held by parties in $\mathcal{C}orr$ are the same.
4. All parties set $[\tilde{x}]_t := [x]_t - [o]_t$.

PARTIALMULT*: Multiplying* $[\tilde{x}]_t$ *and* $[y]_t$. To compute $[z]_t$, all parties invoke the DN multiplication protocol on $([\tilde{x}]_t, [y]_t)$. All parties first prepare a pair of double sharings $([r]_t, [r]_{2t})$ (as that in the DN protocol). Recall that, the shares of $[r]_t, [r]_{2t}$ held by parties in $\mathcal{C}orr$ are 0. Then, all parties run the following steps.

1. All parties locally compute $[e]_{2t} := [\tilde{x}]_t \cdot [y]_t + [r]_{2t}$.
2. $P_{\texttt{king}}$ collects shares of $[e]_{2t}$ from parties in $\mathcal{P}_{\texttt{active}}$. For each party $P_i \in \mathcal{C}orr$, $P_{\texttt{king}}$ sets the i-th share of $[e]_{2t}$ to be 0. Then $P_{\texttt{king}}$ generates a degree-t sharing $[e]_t$ and distributes the shares to all other parties.
3. All parties locally compute $[z]_t = [e]_t - [r]_t$.

Reducing the Communication of Refresh and PartialMult. We note that, to reconstruct a degree-t sharing, $P_{\texttt{king}}$ only needs $t + 1$ shares. Therefore, there is no need to let all parties receive the shares of $[r]_t$. In the beginning of each segment, all parties agree on a set of parties $\mathcal{T} \subseteq \mathcal{P}_{\texttt{active}}$ such that (1) $|\mathcal{T}| = t + 1$, and (2) $P_{\texttt{king}} \in \mathcal{T}$. In brief, $\mathcal{T}$ contains $P_{\texttt{king}}$ and t other parties in $\mathcal{P}_{\texttt{active}}$.

When generating $[r]_t$, only parties in $\mathcal{T}$ will receive the shares of $[r]_t$. This can be achieved by requiring each dealer only sends shares to parties in $\mathcal{T}$. In the first step of REFRESH, parties in $\mathcal{T}$ compute their shares of $[x]_t + [r]_t$ and send them to $P_{\texttt{king}}$. Together with the share held by $P_{\texttt{king}}$, there are $t+1$ shares, which are enough to reconstruct the whole sharing $[e]_t := [x]_t + [r]_t$. In this way, the cost of generating random sharings for REFRESH is reduced by half.

Furthermore, when $P_{\texttt{king}}$ generates $[o]_t$, we can require that the shares of $[o]_t$ held by parties in $\mathcal{P}_{\texttt{active}} \backslash \mathcal{T}$ are set to be 0. Recall that $P_{\texttt{king}}$ learns the shares of $[x]_t$ held by parties in $\mathcal{C}orr$ and the shares of $[o]_t$ held by parties in $\mathcal{C}orr$ are the same as those of $[x]_t$. Since the shares held by parties in $\mathcal{P} \backslash \mathcal{T}$ are fixed and

$|\mathcal{P}\backslash\mathcal{T}| = t$, with these t shares and the secret value $o = 0$, $P_{\texttt{king}}$ can compute the shares of $[o]_t$ held by parties in $\mathcal{T}$. Now, $P_{\texttt{king}}$ only needs to distribute $[o]_t$ to parties in $\mathcal{T}$, and, parties in $\mathcal{P}_{\texttt{active}}\backslash\mathcal{T}$ simply set their shares of $[o]_t$ to be 0. In this way, the cost of distributing $[o]_t$ is reduced by half.

In the DN multiplication protocol, $P_{\texttt{king}}$ can set the shares of $[e]_t$ held by parties in $\mathcal{P}\backslash\mathcal{T}$ to be 0. With these $|\mathcal{P}\backslash\mathcal{T}| = t$ shares and the secret value e, $P_{\texttt{king}}$ can recover the whole sharing $[e]_t$. In this way, $P_{\texttt{king}}$ only needs to distribute $[e]_t$ to parties in $\mathcal{T}$, and, parties in $\mathcal{P}_{\texttt{active}}\backslash\mathcal{T}$ simply set their shares of $[e]_t$ to be 0. As a result, the cost of distributing $[e]_t$ is reduced by half. Note that in the overall protocol, several multiplication gates will be evaluated in parallel, and this optimization can potentially lead to a reduction in the overall communication by a factor of 1/2.

In summary, when $\mathcal{C}orr = \emptyset$, there is no need to run the "Surgery". Our approach achieves 5.5 field elements per multiplication gate, as that in our secure-with-abort protocol. When at least one party is identified as a corrupted party, our approach needs 7.5 field elements per multiplication gate.

Checking the Correctness of Refresh. We point out that the above approach does not guarantee the correctness. In particular, we need to verify Refresh in the end of the evaluation of each segment. *It is worth noting that the verification of* Refresh *also utilizes the virtual transcript idea.*

We note that the transcript of Refresh contains 5 degree-t sharings:

$$([x]_t, [\tilde{x}]_t, [r]_t, [e]_t, [o]_t).$$

Here $[x]_t$ is the input sharing, $[\tilde{x}]_t$ is the output sharing, $[r]_t$ is a random sharing which is only held by parties in $\mathcal{T}$, $[e]_t$ is the sharing $P_{\texttt{king}}$ collected from parties in $\mathcal{T}$, and $[o]_t$ is the sharing of 0 dealt by $P_{\texttt{king}}$.

Given m transcripts $\{([x^{(i)}]_t, [\tilde{x}^{(i)}]_t, [r^{(i)}]_t, [e^{(i)}]_t, [o^{(i)}]_t)\}_{i=1}^m$, we want to verify that, for each $i \in [m]$, (1) $x^{(i)} = \tilde{x}^{(i)}$ and (2) the shares of $[\tilde{x}^{(i)}]$ held by parties in $\mathcal{C}orr$ are 0. To this end, our idea is to compress m checks of the transcripts of Refresh into one check of a single transcript. As the verification of multiplications, to protect the privacy of the original m transcripts, we add a random transcript as a mask in the compression step.

The random transcript is prepared in the following manner.

1. All parties prepare two random sharings $[x^{(0)}]_t, [r^{(0)}]_t$ in the same way as that in the preparation phase of the DN protocol.
2. All parties invoke Refresh on $[x^{(0)}]_t$ with the random sharing $[r^{(0)}]_t$.

This random transcript is denoted by

$$([x^{(0)}]_t, [\tilde{x}^{(0)}]_t, [r^{(0)}]_t, [e^{(0)}]_t, [o^{(0)}]_t).$$

Compressing the Transcripts into One. Consider the following 5 sharings of polynomials:

$$[F(\lambda)]_t = \sum_{i=0}^{m} [x^{(i)}]_t \lambda^i, \quad [\tilde{F}(\lambda)]_t = \sum_{i=0}^{m} [\tilde{x}^{(i)}]_t \lambda^i, \quad [R(\lambda)]_t = \sum_{i=0}^{m} [r^{(i)}]_t \lambda^i,$$
$$[E(\lambda)]_t = \sum_{i=0}^{m} [e^{(i)}]_t \lambda^i, \quad [O(\lambda)]_t = \sum_{i=0}^{m} [o^{(i)}]_t \lambda^i.$$

Note that, by the linear homomorphism property of the Shamir secret sharing scheme, for every λ,

$$([F(\lambda)]_t, [\tilde{F}(\lambda)]_t, [R(\lambda)]_t, [E(\lambda)]_t, [O(\lambda)]_t)$$

can be seen as a virtual transcript of Refresh:

1. Parties in $\mathcal{T}$ locally compute $[E(\lambda)]_t := [F(\lambda)]_t + [R(\lambda)]_t$. Note that the shares of $[E(\lambda)]_t, [F(\lambda)]_t$ held by parties in $\mathcal{C}orr$ are the same.
2. $P_{\texttt{king}}$ collects the shares of $[E(\lambda)]_t$ from parties in $\mathcal{T}$ and computes the shares held by parties in $\mathcal{C}orr$.
3. $P_{\texttt{king}}$ generates a random degree-t sharing $[O(\lambda)]_t$ such that (1) $O(\lambda) = 0$, (2) the shares held by parties in $\mathcal{P}_{\texttt{active}} \backslash \mathcal{T}$ are 0, and (3) the shares of $[O(\lambda)]_t, [E(\lambda)]_t$ held by parties in $\mathcal{C}orr$ are the same. Then $P_{\texttt{king}}$ distributes the shares of $[O(\lambda)]_t$ to parties in $\mathcal{T}$.
4. All parties set $[\tilde{F}(\lambda)]_t := [F(\lambda)]_t - [O(\lambda)]_t$.

If at least one transcript of the original m transcripts is incorrect, then the number of λ such that $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t, [R(\lambda)]_t, [E(\lambda)]_t, [O(\lambda)]_t)$ is a correct transcript is bounded by m. Therefore, to verify the original m transcripts, it is sufficient to examine the transcript $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t, [R(\lambda)]_t, [E(\lambda)]_t, [O(\lambda)]_t)$ for a random λ. Let

$$([x^\star]_t, [\tilde{x}^\star]_t, [r^\star]_t, [e^\star]_t, [o^\star]_t)$$

denote the final virtual transcript of Refresh we want to check.

Checking the Virtual Transcript. To check the correctness of $([x^\star]_t, [\tilde{x}^\star]_t, [r^\star]_t, [e^\star]_t, [o^\star]_t)$, all parties publish their shares of $[x^\star]_t, [\tilde{x}^\star]_t$, parties in $\mathcal{T}$ publish their shares of $[r^\star]_t, [e^\star]_t, [o^\star]_t$, and $P_{\texttt{king}}$ publishes the sharing $[e^\star]_t$ it received and the sharing $[o^\star]_t$ it distributed. If it is an incorrect transcript, then all parties must observe one of the following cases:

- The input sharing $[x^\star]_t$ is inconsistent.
- After reconstructing the whole sharing $[r^\star]_t$ from the shares held by parties in $\mathcal{T}$, the shares of $[r^\star]_t$ held by parties in $\mathcal{C}orr$ are not 0.
- Some party P_i does not follow the protocol.
- Two parties $(P_i, P_{\texttt{king}})$ do not agree on the message sent from one party to the other party.

As the verification of multiplications, for the first two cases, there will be another protocol to help find errors. For the last two cases, we can immediately identify a corrupted party or a pair of disputed parties. Thus, the check of the virtual transcript guarantees that either the original m transcripts of Refresh are correct, or all parties can identify a corrupted party or a pair of disputed parties in the end.

Further Problem. We note that we need to make sure adding the surgery procedure in the protocol *will not* break the security of our secure-with-abort protocol. In fact, the security relies on the fact that the DN protocol provides perfect privacy before the output phase even when the adversary is fully malicious. *Replacing the DN protocol by another semi-honest protocol may break down the security entirely.* We refer the readers to [GSZ20] for more details.

Removing Higher Order Circuit Dependent Terms. We note that the construction from [BSFO12] uses Beaver triples to compute multiplications in the computation phase. One benefit of this method is that Beaver triples provide plenty of redundancy which simplifies the checking process in the computation phase. However, the use of Beaver triples unfortunately requires a verification for each layer of the circuit, which leads to the quadratic term related to D_M.

On the other hand, although when instantiating the transformation from [IKP$^+$16] with the best-known protocol for security with abort, the quadratic term w.r.t. the circuit depth is eliminated, it introduces a new higher order term related to the circuit width. This is because the transformation needs to change the corruption threshold whenever a new corrupted party is identified, which requires an expensive re-sharing process for the input sharings of each segment.

As a summary, we start from our secure-with-abort protocol, which does not make use of Beaver triples, to remove the quadratic term related to D_M. To avoid the expensive re-sharing process, we rely on a small surgery to proceed. Combining these two ideas, we remove both the higher order terms related to the circuit depth and the circuit width.

4.3 An Omitted Problem: Verifiable System for Checkpoints

To allow all parties to restart the computation from a checkpoint, i.e., the end of the last segment, all the output sharings of the last segment should be verifiable. This is also a problem we omit when checking the virtual transcript: If all parties finally find out that one of the input sharings is inconsistent, then there is no way to identify a new corrupted party or a new pair of disputed parties by only examining the transcript in this segment. This is because the failure comes from the sharings computed in the previous segment.

Therefore, we borrow the idea from [BSFO12] to add verifiability to the output sharings of each segment. At a high-level, for every pair of parties (P_v, P_i) where P_v acts as a verifier, P_v will generate an authentication key (μ, ν) and P_i will receive an authentication tag $\tau = \mu \cdot \mathsf{share}_i + \nu$ of its share share_i. The

authentication tag is computed using an MPC protocol. At a later point, P_v can verify the shares of P_i by asking P_i to send the associated authentication tags. Since a wrong share will be rejected by at least $t+1$ honest parties and a correct share will be rejected by at most t corrupted parties, a majority vote can decide whether a share is correct or not.

In [BSFO12], each authentication tag is used to authenticate a batch of shares. As a result, the communication cost is independent of the number of shares and therefore, does not affect the concrete efficiency per gate. We make a further improvement to this idea to achieve a larger size of batching by reusing the authentication keys. Some modifications in the verification of authentication tags are also necessary to fit this improvement. We refer the readers to [GSZ20] for more details.

4.4 Summary

In short, the whole computation proceeds as follows. All parties first partition the circuit into several small segments. These segments will be evaluated in sequence. For each segment, the computation process contains the following three steps.

Evaluation. For each segment, if no party is identified as a corrupted party, we simply use the DN protocol to evaluate the addition gates and multiplication gates in this segment. If one or more corrupted parties have been identified, for each multiplication gate,

1. All parties first run Refresh on one of the input wires to change the shares held by identified corrupted parties to be 0.
2. Then all parties evaluate this multiplication gate using the DN protocol (i.e., PartialMult).

Verification. After the evaluation, all parties first check the correctness of Refresh. Then, we use the multiplication verification of our secure-with-abort protocol to check the correctness of the multiplications. In the meanwhile, all parties prepare the virtual transcript of the ultimate multiplication tuple.

- If both checks pass, all parties accept the evaluation of this segment.
- Otherwise, a new corrupted party or a new pair of disputed parties is identified. The evaluation of the current segment is discarded and all parties re-evaluate this segment.

Checkpoint. Finally, in the case that the evaluation is accepted, all parties add verifiability to the output sharings of this segment.

Efficiency Analysis. For any constant $\epsilon > 0$, by properly choosing the parameters in the second step and the third step, it turns out that the communication complexity per multiplication gate of these two steps can be bounded by ϵ field elements per gate. We refer the readers to [GSZ20] for more details. Therefore, these two steps only have a very limited influence on the concrete efficiency.

In the best case, we simply use the DN multiplication protocol to evaluate each multiplication gate. Therefore, the concrete efficiency is 5.5 field elements per multiplication gate. When one or more corrupted parties have been identified, we also need to run Refresh per multiplication gate. Thus, the concrete efficiency is 7.5 field elements per multiplication gate. To summarize, we have the following theorem.

Theorem 2. *Let n be the number of parties, κ be the security parameter. Let $\mathbb{F}$ be a finite field where $|\mathbb{F}| \geq n+1$, and ϕ be the size of a field element. Then, for any constant $\epsilon > 0$ and any arithmetic circuit* `Circuit` *of size C over $\mathbb{F}$, there exists an n-party MPC protocol which securely computes* `Circuit` *with guaranteed output delivery against a fully malicious adversary which controls up to $t \leq n/2$ parties. The communication complexity is $O(Cn\phi)$ bits (ignoring fixed terms which are independent of the circuit). The concrete efficiency is $5.5+\epsilon$ field elements per party per multiplication gate in the best case, and $7.5+\epsilon$ field elements when one or more corrupted parties have been identified.*

We refer the readers to [GSZ20] for the detailed construction and the security proof.

References

ABF+17. Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority mpc for malicious adversariesbreaking the 1 billion-gate per second barrier. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 843–862. IEEE, 2017.

BBCG+19. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 67–97, Cham, 2019. Springer International Publishing.

Bea89. Donald Beaver. Multiparty protocols tolerating half faulty processors. In *Conference on the Theory and Application of Cryptology*, pages 560–572. Springer, 1989.

BGIN19. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS ?19, page 869?886, New York, NY, USA, 2019. Association for Computing Machinery.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

BSFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BTH06. Zuzana Beerliova-Trubiniova and Martin Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference*, pages 305–328. Springer, 2006.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography*, pages 213–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

CDD+99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 311–326, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

CDVdG87. David Chaum, Ivan B Damgård, and Jeroen Van de Graaf. Multiparty computations ensuring privacy of each partys input and correctness of the result. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 87–119. Springer, 1987.

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 *Annual International Cryptology Conference*, pages 34–64. Springer, 2018.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 445–465. Springer, 2010.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

FLNW17. Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 225–255. Springer, 2017.

GIP+14. Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

GIP15. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multiparty computation: from passive to active security via secure simd circuits. In *Annual Cryptology Conference*, pages 721–741. Springer, 2015.

GLOS20. Vipul Goyal, Hanjun Li, Rafail Ostrovsky, and Yifan Song. Fast honest-majority mpc protocols. Manuscript, 2020.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 85–114, Cham, 2019. Springer International Publishing.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

GS20. Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority mpc. Cryptology ePrint Archive, Report 2020/134, 2020. https://eprint.iacr.org/2020/134.

GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. Cryptology ePrint Archive, Report 2020/189, 2020. https://eprint.iacr.org/2020/189.

HM01. Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In *Annual International Cryptology Conference*, pages 101–118. Springer, 2001.

HMP00. Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 143–161. Springer, 2000.

IKP+16. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 430–458, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

LN17. Yehuda Lindell and Ariel Nof. A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–276. ACM, 2017.

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

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

NV18. Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority mpc by batchwise multiplication verification. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security*, pages 321–339, Cham, 2018. Springer International Publishing.

RBO89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

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

Yao82. Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.