Round-Optimal Secure Multi-Party Computation

Shai Halevi [*] Carmit Hazay[†] Antigoni Polychroniadou[‡]
Muthuramakrishnan Venkitasubramaniam[§]

Abstract

Secure multi-party computation (MPC) is a central cryptographic task that allows a set of mutually distrustful parties to jointly compute some function of their private inputs where security should hold in the presence of a malicious adversary that can corrupt any number of parties. Despite extensive research, the precise round complexity of this "standard-bearer" cryptographic primitive is unknown. Recently, Garg, Mukherjee, Pandey and Polychroniadou, in Eurocrypt 2016 demonstrated that the round complexity of any MPC protocol relying on black-box proofs of security in the plain model must be at least four. Following this work, independently Ananth, Choudhuri and Jain, CRYPTO 2017 and Brakerski, Halevi, and Polychroniadou, TCC 2017 made progress towards solving this question and constructed four-round protocols based on non-polynomial time assumptions. More recently, Ciampi, Ostrovsky, Siniscalchi and Visconti in TCC 2017 closed the gap for two-party protocols by constructing a 4-round protocol from polynomial-time assumptions. In another work, Ciampi, Ostrovsky, Siniscalchi and Visconti TCC 2017 showed how to design a 4-round multi-party protocol for the specific case of multi-party coin-tossing.

In this work, we resolve this question by designing a 4-round actively secure multi-party (two or more parties) protocol for general functionalities under standard polynomial-time hardness assumptions.

Keywords: Secure Multi-Party Computation, Garbled Circuits, Round Complexity, Additive Errors

[*]IBM T.J. Watson. Email: shaih@alum.mit.edu. Research supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236

[†]Bar-Ilan University. Email: carmit.hazay@cs.biu.ac.il. Research supported 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.

[‡]Cornell Tech University of Rochester. Email: antigoni@cornell.edu. Supported by the National Science Foundation under Grant No. 1617676, IBM under Agreement 4915013672 and the Packard Foundation under Grant 2015-63124. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

[§]University of Rochester. Email: muthuv@cs.rochester.edu. Research supported by Google Faculty Research Grant and NSF Award CNS-1526377.

Contents

1 Introduction

Secure multi-party computation. A central cryptographic task, *secure multi-party computation* (MPC), considers a set of parties with private inputs that wish to jointly compute some function of their inputs while preserving privacy and correctness to a maximal extent [Yao86, CCD87, GMW87, BGW88].

In this work, we consider MPC protocols that may involve two or more parties for which security should hold in the presence of *active* adversaries that may corrupt any number of parties (i.e. dishonest majority). More concretely, *we are interested in identifying the precise round complexity of MPC protocols for securely computing arbitrary functions.*

In [GMPP16], Garg, et al., proved a lower bound of four rounds for MPC protocols that relies on black-box simulation. Following this work, in independent works, Ananth, Choudhuri and Jain [ACJ17] and Brakerski, Halevi and Polychroniadou, [BHP17] showed a matching upper bound by constructing 4-round protocols based on the Decisional Diffie-Hellman (DDH) and Learning With Error (LWE) assumptions, respectively, albeit with super-polynomial hardness. More recently, Ciampi, Ostrovsky, Siniscalchi and Visconti in [COSVb] closed the gap for two-party protocols by constructing a 4-round protocol from standard polynomial-time assumptions. The same authors in another work [COSVa] showed how to design a 4-round multi-party protocol for the specific case of multi-party coin-tossing.

The state-of-affairs leaves the following fundamental question regarding round complexity of cryptographic primitives open:

> *Does there exist 4-round secure multi-party computation protocols for general functionalities based on standard polynomial-time hardness assumptions and black-box simulation?*

We remark that many prior works made progress towards answering this question while relaxing one or more requirements. In the two-party setting, the recent work of Ciampi et al. [COSVb] showed how to obtain a 4-round protocol based on trapdoor permutations. Assuming trusted setup, namely, a common reference string, 2-round constructions can be obtained [GGHR14, MW16] or 3-round assuming tamper-proof hardware tokens [HPV16].[1] In the case of passive adversaries, (or even the slightly stronger setting of semi-malicious[2] adversaries) three round protocols based on the Learning With Errors assumption have been constructed by Brakerski et al. [BHP17]. Ananth et al. gave a 5-round protocol based on DDH [ACJ17]. Under subexponential hardness assumptions, 4-round constructions were demonstrated in [BHP17, ACJ17]. Under the relaxation of superpolynomial simulation, the work of Badrinarayanan et al. [BGJ+17] shows how to obtain 3-round MPC assuming subexponentially secure LWE and DDH. For specific multi-party functionalities 4-round constructions have been obtained, e.g., coin-tossing and zero-knowledge by Ciampi et al. [COSVb, GRRV14]. Finally, if we assume an honest majority, the work of Damgard and Ishai [DI05] provided a 3-round MPC protocol.

1.1 Our Results

The main result we establish is a 4-round multi-party computation protocol for general functionalities in the plain model based on standard polynomial-time hardness assumptions. Slightly more formally, we establish the following theorem.

[1] Where in this model the lower bound is 2 rounds.

[2] A semi-malicious adversary is allowed to invoke a corrupted party with arbitrary chosen input and random tape, but otherwise follows the protocol specification honestly as a passive adversary.

Theorem 1.1 (Informal) *Assuming enhanced trapdoor permutations and public-key encryptions schemes that admit affine homomorphisms with equivocation (cf. Definition 2.5), there exists a 4-round multi-party protocol that securely realizes arbitrary functionalities in the presence of active adversaries corrupting any number of parties.*

Informally, an affine homomorphic encryption scheme admits an affine transformation that takes as input a ciphertext $c = \mathsf{Enc}(m)$ and two values a, b and outputs c' that decrypts to $a \cdot m + b$. Equivocation further requires that given c' computed via an affine transformation with inputs c, a, b there exists an "explain" procedure that can generate randomness for any other a', b' such that $a \cdot m + b = a' \cdot m + b'$ that explains c' as obtained via the affine transformation on inputs c, a', b'. We show how to instantiate such an encryption scheme by relying on standard additively homomorphic encryption schemes (or slight variants thereof). More precisely, we instantiate such an encryption scheme using LWE, DDH and Quadratic Residuosity (QR) hardness assumptions. This theorem addresses our motivating question and resolves the round complexity of multiparty computation protocols.

1.2 Our Techniques

The starting point of our techniques is the beautiful work of Ananth, Choudhuri and Jain [ACJ17] where they observe that the task of securely computing an arbitrary functionality reduces to the task of securely computing many 3-bit multiplications in parallel, using specialized randomized encodings. Relying on an elegant 3-round protocol for realizing 3-bit multiplications (which in turn can be based on 2-round oblivious transfer), the authors first constructed a 4-round passively secure protocol. Making this protocol secure against active parties required enforcing correctness of the players actions in the 3rd and 4th rounds. The specific approach pursued in [ACJ17] for the proof in the 3rd round (oversimplified) involved designing a special 3-round non-malleable zero-knowledge proofs, for which they had to rely on super-polynomial assumptions. A 4-round proof to enforce correctness in the last round was shown based on standard assumptions. Finally, the authors relied on the construction of Applebaum, Ishai and Kushilevitz [AIK06] which relies on the existence of PRGs in NC^1 to instantiate their specialized randomized encoding.

On a high-level, our approach is to (a) weaken the requirement of the proof in the 3rd round from non-malleable zero-knowledge to non-malleable witness indistinguishability and (b) eliminate the proof required in the 4th round. Step (a) is required to eliminate super-polynomial assumptions (recalling here that 3-round zero-knowledge is impossible with black-box simulation [GK96]), and Step (b) eases the simulator by requiring only a specific type of rewinding, namely, rewinding from the 3rd to the 2nd round.

Tackling problem (a) First, we modify the 3-bit multiplication protocol described in Figure 1 that is based on three executions of oblivious transfers (OTs) as follows. We modify the three invocations of the OTs to be carried out using six invocations of OTs. Namely, each OT in the original protocol will be replaced with two parallel invocations of OTs where the receiver uses the same choice bit in both invocations and the sender secret-shares its two strings between the two instances. Next, between every ordered pair of parties (P_i, P_j), P_i commits to two messages in parallel using a 3-round non-malleable commitment. We will also run in parallel a (resettable) ZAP proof starting in the second and completing in the third round where P_i proves to P_j that one of the two messages in the non-malleable commitments is a "valid" witness to its actions in the first three rounds. As it turns out, the only way we could make this approach work, is to weaken the requirements of the "validity" predicate. More precisely, we modify the predicate as follows:

- For actions of a party playing the role of the receiver in a pair of OTs (that replaced a single OT in the original protocol), the predicate will only check if the witness is consistent with only one of the two OTs and,

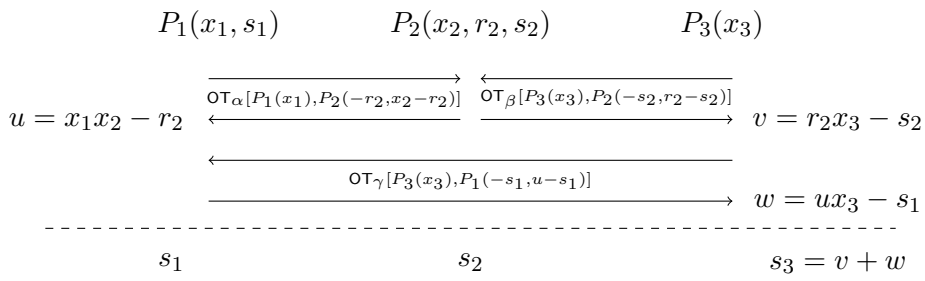

Figure 1: The 3-round 3-bit multiplication protocol from [ACJ17] via a 2-round oblivious transfer protocol denoted by $\mathsf{OT}[\mathrm{Rec}(b), \mathrm{Sen}(m_0, m_1)]$ where b is the receiver's input and (m_0, m_1) is sender's input in the OT interaction. The values u, v, w denote the output values of $\mathsf{OT}_\alpha, \mathsf{OT}_\beta, \mathsf{OT}_\gamma$ executions, respectively. At the end of the protocol, parties P_1, P_2, P_3 output s_1, s_2, s_3, respectively. The receiver's message in OT_γ can be sent in the same round as the senders messages in $\mathsf{OT}_\alpha, \mathsf{OT}_\beta$.

- For actions of a party playing the role of the sender it will require consistency with both invocations.
- Furthermore, the validity will not guarantee that the inputs (u, s_1) of P_1 in the OT executed between P_1 and P_3 are "correct", where correct means u is the result of the OT computation executed between P_1 and P_2 in the first two rounds.

With these modifications, we show by a careful selection of intermediate hybrids that we can switch the inputs of the honest parties in the first three rounds from the honest input to a random input while extracting the inputs of the corrupted parties. A crucial modification we consider is to rely on a non-malleable commitment that admits invalid commitments. Such commitments are usually the backbone used to bootstrap to full-fledged non-malleable commitments (see [GRRV14, Khu17] for few examples). A common issue with such commitments is the problem of "over-extraction" where an invalid commitment can fool an extractor to extract the wrong message. We will rely on the ZAP to enforce that at least one of the two non-malleable commitments made between each pair of parties is "valid", thereby, guaranteeing at least one of the two extracted messages to be correct. Finally, we rely on the technique introduced in [COSV16] for transforming any non-malleable commitment to an *input-delayed* non-malleable commitment and apply the transformation to the 3-round non-malleable commitment scheme implicit in [GRRV14].

Moving back to our weaker validity predicate in the ZAP statement, a first downside is that a malicious receiver can use two different inputs in a pair of OT invocations when it was supposed to have the same choice bit. This is easy to address as such an attack only restricts the adversary to learning random values that are independent of the sender's real inputs. Another downside of such a weakening is that if P_1 is controlled by the adversary, it can choose u arbitrarily, in particular, independent of the result of the first OT. The effect of such an attack in the 3-bit multiplication can be modelled as an error being introduced in an intermediate part of the computation. Formally, we can show that, a malicious party controlling party P_1 can choose a value e that is independent of the inputs of the honest parties, and influence the result of the computation to be $(x_1x_2 + e) \cdot x_3$ instead of $x_1x_2x_3$. Suppose we had a degree-3 randomized encoding that is "resilient" to such additive errors then we would have been done. In fact, transformations presented in the works [GIP+14, GIP15, GIW16] provide precisely such a mechanism to transform any circuit C to be resilient to (arbitrary) additive attacks on the intermediate computation of the function. However, all the current transformations increase the degree of the circuit and since we start with a degree-3 randomized encoding, we cannot pursue this approach. Our approach to tackle this problem is to rely on a special degree-3 encoding of a function f to f'. More precisely, for the specific encoding, by carefully assigning roles in the 3-bit multiplication protocol to realize this encoding, we will be able to show that errors e introduced

in all the 3-bit multiplications with a corrupted P_1 can be effectively translated to an additive attack to the underlying computation of f. Now, if we precompile our function f to $\hat{f}$ following the transformations of [GIP+14, GIP15, GIW16] and then apply the special degree-3 randomized encoding $\hat{f}'$ we will obtain an encoding that is of degree-3 and resilent to the additive errors introduced by corrupted P_1. In fact, the specific randomized encoding that we will rely on is the multiparty garbling of Beaver, Micali and Rogaway [BMR90].

Tackling problem (b) Towards eliminating the proof of the actions in the 4th round, we focus our discussion on BMR garbling which is needed for tackling (a). In this approach, the messages shared in the final round are reconstructed by the parties to compute the garbling circuit, input keys and output translation tables (i.e. a table to interpret the results of the garbled circuit evaluation). First, we point out that a rushing adversary can wait until it receives the 4th round message from all parties and then share its message. If no proof is made in the 4th round, it can introduce errors in any part of its message. We leverage a property of the BMR garbling identified in the works of [LPSY15, HSS17]: if the errors introduced by the adversary cause the honest parties to abort during the computation, the probability of abort can be shown to be independent of the actual values in the wires of the circuit. In slight more detail, an adversary can cause an honest party to abort in the evaluation of a garbled gate g by introducing errors in a specific row among the four rows of the garbled circuit. However whether or not the active path will pass through the bad row is randomized by all parties and hence the abort event is independent of the wire values. Finally, the adversary can modify the translation table arbitrarily making the honest party output the wrong answer. This can be fixed by a standard technique of precompiling f to additionally receiving keys to a MAC from the parties and output a MAC of the output under each key along with the output.

A modular presentation: the case of defensible simulation. In order to make our presentation more modular, we introduce the notion of defensible simulation. Focussing on a 4-round protocol, we will first consider a simpler scenario where we design a 4-round protocol that achieves full simulation against active adversaries that provide a valid defense in a special output tape at the end of the third round. If the adversary does not provide a defense then no guarantee is made. We call this *defensible simulation* and show a protocol to realize arbitrary functionalities. The concrete steps we take are to first design a protocol that achieves defensible simulation for 3-bit multiplication and then compile it to securely compute multiple instances of degree-3 polynomials. Finally, we show that by securely computing the BMR garbling when instantiated with degree-3 polynomials we can securely compute arbitrary functionalities up to defensible simulation.

Next, we design a protocol to securely realize degree-3 polynomials against active adversaries where we show that for any active adversary there is a simulator that can generate an indistinguishable view while outputting a valid defense at the end of third round. Then we can combine the simulation strategy against defensible adversaries to obtain full simulation. We remark that the reason to split the proofs into two segments is to separate the proofs involving the security reductions required for the garbling (with additive errors on the wires) and the reductions for non-malleability.

We remark that the notion of defensible simulation is introduced merely as a conceptual step to make the presentation modular. We leave it as future work to initiate a careful study of this notion as a tool for obtaining round-efficient protocols.

1.3 Related Work

The earliest MPC protocol is due to Goldreich, Micali and Wigderson [GMW87]. The round complexity of this approach is proportional to the circuit's multiplication depth (namely, the largest number of multiplication gates in the circuit on any path from input to output) and can be non-constant for most functions.

A different approach was taken in [BMR90], extending the celebrated garbled circuits technique of [Yao86] to the multi-party setting. This *constant-round* protocol, developed by Beaver, Micali and Rogaway, achieved security in the presence of passive adversaries (and against active adversaries in the honest majority setting). This was later improved by Katz, Ostrovsky and Smith [KOS03] to obtain the first constant-round MPC protocol secure against active adversaries in the dishonest majority setting while relying on non-black-box simulation [Bar01].

The first constant-round MPC protocols that relied on black-box simulation were obtained by Goyal [Goy11] and Lin and Pass [LP11]. Namely, the work of Lin et al. [LPV08] reduced the task of designing round-efficient MPC protocols to designing round-efficient non-malleable commitment schemes whereas [Goy11, LP11] provided the first constant-round non-malleable commitment schemes. Following these works, a long line of research has focussed on improving the precise round complexity of *concurrent* non-malleable commitments [GLOV12,GRRV14,GPR16] leading up to the work of Ciampi et al. [COSV17] who provided a 4-round non-malleable commitment scheme based on one-way functions and more recently a non-malleable zero-knowledge argument based on one-way functions by Ciampi et al. [COSVa]. A lower bound of 2 rounds was established by Pass regarding the round complexity of concurrent non-malleable commitment schemes. Finally, Khurana in [Khu17] resolved the round-complexity question by providing a 3-round non-malleable commitments scheme based on one-way permutations.

1.4 A Roadmap

In Section 4.1 we provide our 4-round 3-bit multiplication protocol Π_{DMULT} with defensible simulation. In Section 4.2 we provide a 4-round protocol Π_{dpoly} that securely computes parallel multiple instances of Π_{DMULT}. In Section 4.3 we compile the protocol Π_{dpoly} to the defensible simulatable protocol Π_f which securely computes arbitrary functionalities (via the BMR garbling). Finally, in Sections 5 and 6 we compile protocol Π_f that is secure against defensible adversaries to a 4-round fully maliciously secure protocol.

2 Preliminaries

Basic notations. We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ'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 $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$.

We specify next the definitions of computationally indistinguishable and statistical distance.

Definition 2.1 *Let $X = \{X(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be two distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \overset{c}{\approx} Y$, if for every* PPT *machine $\mathcal{D}$, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large κ's,*

$$\left|\Pr\left[\mathcal{D}(X(a,\kappa), 1^\kappa) = 1\right] - \Pr\left[\mathcal{D}(Y(a,\kappa), 1^\kappa) = 1\right]\right| < \frac{1}{p(\kappa)}.$$

Definition 2.2 *Let X_κ and Y_κ be random variables accepting values taken from a finite domain $\Omega \subseteq \{0,1\}^\kappa$. The* statistical distance *between X_κ and Y_κ is*

$$SD(X_\kappa, Y_\kappa) = \frac{1}{2}\sum_{\omega\in\Omega}\left|\Pr[X_\kappa = \omega] - \Pr[Y_\kappa = \omega]\right|.$$

We say that X_κ and Y_κ are ε-close *if their statistical distance is at most $SD(X_\kappa, Y_\kappa) \leq \varepsilon(\kappa)$. We say that X_κ and Y_κ are* statistically close*, denoted $X_\kappa \approx_s Y_\kappa$, if $\varepsilon(\kappa)$ is negligible in κ.*

2.1 Additive Secret-Sharing

In an additive secret-sharing scheme, N parties hold shares the sum of which yields the desired secret. By setting all but a single share to be a random field element, we ensure that any subset of $\mathrm{N}-1$ parties cannot recover the initial secret.

Definition 2.3 (Additive secret-sharing) *Let $\mathbb{F}_2$ be a finite field and let $\mathrm{N} \in \mathbb{N}$. Consider the secret-sharing scheme $\mathcal{S}^{\mathrm{N}} = (\mathsf{Share}, \mathsf{Recover})$ defined below.*

- *The algorithm* Share *on input (s, N) performs the following:*
  1. *Generate $(s_1, \ldots, s_{\mathrm{N}-1})$ uniformly at random from $\mathbb{F}_2$ and define $s_{\mathrm{N}} = s - \sum_{i=1}^{\mathrm{N}-1} s_i$.*
  2. *Output $(s_1, \ldots, s_{\mathrm{N}})$ where s_i is the share of the i^{th} party.*
- *The recovery algorithm* Recover *on input $(s_1, \cdots, s_{\mathrm{N}})$, outputs $\sum_{i=1}^{\mathrm{N}} s_i$.*

It is easy to show that the distribution of any $N-1$ of the shares is the uniform one on $\mathbb{F}_2^{\mathrm{N}-1}$ and hence independent of s.

Secret-sharing notation. In the sequel for a value $s \in \mathbb{F}_2$ we denote by $[s]$ a random additive secret sharing of s. That is, $[s] \leftarrow \mathsf{Share}(s, \mathrm{N})$ where $[s] = (s_1, \ldots, s_{\mathrm{N}})$.

2.2 Pseudorandom Functions

Informally speaking, a pseudorandom function (PRF) is an efficiently computable function that looks like a truly random function to any PPT observer. The [BMR90] garbling technique from [LPSY15], which we adapt in this paper, is proven secure based on a pseudorandom function (PRF) with multiple keys, defined below.

Definition 2.4 (Pseudorandom function with multiple keys) *Let $F : \{0,1\}^\kappa \times \{0,1\}^n \mapsto \{0,1\}^n$ be an efficient, length preserving, keyed function. F is a* pseudorandom function under multiple keys *if for all polynomial-time distinguishers $\mathcal{D}$, there exists a negligible function* negl *such that:*

$$\left| \Pr[\mathcal{D}^{F_{\bar{k}}(\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{D}^{\bar{f}(\cdot)}(1^\kappa) = 1] \right| \leq \mathsf{negl}(\kappa).$$

where $F_{\bar{k}} = F_{k_1}, \ldots, F_{k_{m(n)}}$ are the pseudorandom function F keyed with polynomial number of randomly chosen keys $k_1, \ldots, k_{m(n)}$ and $\bar{f} = f_1, \ldots, f_{m(n)}$ are $m(n)$ random functions from $\{0,1\}^n \mapsto \{0,1\}^n$. The probability in both cases is taken over the randomness of $\mathcal{D}$ as well.

When the keys are independently chosen then security with multiple keys is implied by the standard security PRF notion, by a simple hybrid argument.

2.3 Affine Homomorphic PKE

We rely on public-key encryption schemes that admit an affine homomorphism and an equivocation property. As we demonstrate via our instantiations, most standard additively homomorphic encryption schemes satisfy these properties. Specifically, we provide instantiations based on Quadratic Residuosity (QR), Decisional Diffie-Hellman (DDH), and Learning With Errors (LWE) assumptions.

Definition 2.5 (Affine homomorphic PKE) *We say that a public key encryption scheme* $(\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* affine homomorphic *if*

- *Affine Transformation: There exists an algorithm* AT *such that for every* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$, $m \in \mathcal{M}_\kappa$, $r_c \leftarrow \mathcal{D}_{rand}(1^\kappa)$ *and every* $a, b \in \mathcal{M}_\kappa$, $\mathsf{Dec}_{\mathrm{SK}}(\mathsf{AT}(\mathrm{PK}, c, a, b)) = am + b$ *holds with probability 1, where* $c = \mathsf{Enc}_{\mathrm{PK}}(m; r_c)$, *where* $\mathcal{D}_{rand}(1^\kappa)$ *is the distribution of randomness used by* Enc.

- *Equivocation: There exists an algorithm* $\mathsf{Explain}$ *such that for every* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$, *every* $m, a_0, b_0, a_1, b_1 \in \mathcal{M}_\kappa$ *such that* $a_0 m + b_0 = a_1 m + b_1$ *and every* $r_c \leftarrow \mathcal{D}_{rand}(1^\kappa)$, *it holds that the following distributions are statistically close over* $\kappa \in \mathbb{N}$*:*

  - $\{\sigma \leftarrow \{0,1\}; r \leftarrow \mathcal{D}_{rand}(1^\kappa); c^* \leftarrow \mathsf{AT}(\mathrm{PK}, c, a_\sigma, b_\sigma; r) : (m, r_c, c^*, r, a_\sigma, b_\sigma)\}$, *and*
  - $\{\sigma \leftarrow \{0,1\}; r \leftarrow \mathcal{D}_{rand}(1^\kappa); c^* \leftarrow \mathsf{AT}(\mathrm{PK}, c, a_\sigma, b_\sigma; r);$
    $t \leftarrow \mathsf{Explain}(\mathrm{SK}, a_\sigma, b_\sigma, a_{1-\sigma}, b_{1-\sigma}, m, r_c, r) : (m, r_c, c^*, t, a_{1-\sigma}, b_{1-\sigma})\}$,

  where $c = \mathsf{Enc}_{\mathrm{PK}}(m; r_c)$.

In what follows, we demonstrate how to meet Definition 2.5 under a variety of hardness assumptions.

2.3.1 An Instantiation based on LWE

Definition 2.6 (LWE [Reg09]) *Let* κ *be the security parameter, let* $q = q(\kappa)$ *be integers and let* $\chi = \chi(\kappa)$, *be error distributions over* $\mathbb{Z}$. *The* $LWE_{\kappa,q,\chi}$ *assumption says that for any polynomial* $m = m(\kappa)$,

$$(A, s \cdot A + e) \approx_c (A, z)$$

where $A \leftarrow \mathbb{Z}_q^{\kappa \times m}$, $s \leftarrow \mathbb{Z}_q^\kappa, e \leftarrow \chi^m$ *and* $z \leftarrow \mathbb{Z}_q^m$.

- **Setup:** Let κ be the security parameter, $m = \mathsf{poly}(\kappa)$, $q > \mathsf{superpoly}(\kappa)$ and error bound $\sigma = \mathsf{poly}(\kappa)$ and $\sigma' = \mathsf{superpoly}(\kappa)$.

- **Key-Generation:** Choose at random $A \in \mathbb{Z}_q^{\kappa \times m}$, $s \leftarrow D_{\mathbb{Z}^\kappa, \sigma}$, $e \leftarrow D_{\mathbb{Z}^m, \sigma}$. Set $b = sA + e \mod q$ and $\mathrm{PK} = \begin{pmatrix} A \\ b \end{pmatrix} \in \mathbb{Z}_q^{(\kappa+1) \times m}$ where $\mathrm{SK} = (s| - 1) \in \mathbb{Z}_q^{(\kappa+1)}$.

- **Encryption:** Given the public key PK and a plaintext $m \in \{0,1\}$, choose $r_c \leftarrow D_{\mathbb{Z}^m, \sigma}$ and output the ciphertext $c = \mathrm{PK} \cdot r_c + \left\lfloor \frac{q}{2} \right\rfloor \cdot (0 \ldots 0\ m)^\top \in \mathbb{Z}_q^{(\kappa+1)}$.

- **Decryption:** Given the secret key SK and the ciphertext c, compute the inner-product $d = \langle \mathrm{SK}, c \rangle \mod q$. Output 1 if $|d| > \frac{q}{4}$ and 0 if $|d| < \frac{q}{4}$.

- **Affine transformation:** Given PK, c and $a, b \in \{0,1\}$, choose $r \leftarrow D_{\mathbb{Z}^m,\sigma'}$ and set $c' = \mathrm{PK} \cdot r + \left\lfloor \frac{q}{2} \right\rfloor \cdot (0 \ldots 0\ b)^\top \in \mathbb{Z}_q^{(\kappa+1)}$. Output ciphertext c^* as follows:

$$c^* = \begin{cases} c', & \text{if } a = 0 \\ c' + c \mod q, & a = 1, b = 0 \\ c' - c \mod q & a = 1, b = 1 \end{cases}$$

- **Equivocation:** Note that $c^* = \mathrm{PK} \cdot r^* + \left\lfloor \frac{q}{2} \right\rfloor \cdot (0 \ldots 0\ w)^\top$ where $w = a_\sigma \cdot m \oplus b_\sigma$, and $r^* = r + a_\sigma(1 - 2 \cdot b_\sigma) \cdot r_c$. Given randomness r_c, r and $a_{1-\sigma}, b_{1-\sigma}$ output t such that $t + a_{1-\sigma}(1 - 2 \cdot b_{1-\sigma}) \cdot r_c = r + a_\sigma(1 - 2 \cdot b_\sigma) \cdot r_c$.

2.3.2 An Instantiation based on DDH

Definition 2.7 (The Decisional Diffie-Hellman (DDH) Problem) *Let $(\mathbb{G}, \cdot)$ be a cyclic group of prime order p and with generator g. Let α, β, γ be sampled uniformly at random from $\mathbb{Z}_p$ (i.e., $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$). The DDH problem asks to distinguish the distributions $(g, g^\alpha, g^\beta, g^{\alpha\beta})$ and $(g, g^\alpha, g^\beta, g^\gamma)$.*

We next describe the El-Gamal encryption scheme [Gam85] over characteristic-2 fields. Since this encryption scheme is defined over larger fields, to compute the encryption of $a \oplus b$, we compute $a + b - 2ab = a \oplus b$.

- **Key-Generation:** Choose a random generator $g \in \mathbb{G}$ and a random number $\mathrm{SK} \in \mathbb{Z}_p$ and compute $\mathrm{PK} = g^{\mathrm{SK}}$.
- **Encryption:** Given the public key PK and a plaintext $m \in \{0,1\}$, choose $r_c \leftarrow \mathbb{Z}_p$ and output the ciphertext $c = (c_1, c_2) = (g^{r_c}, \mathrm{PK}^{r_c} g^m)$.
- **Decryption:** Given the secret key SK and the ciphertext $c = (c_1, c_2)$, compute $m = c_2 (c_1^{\mathrm{SK}})^{-1}$.
- **Affine transformation:** Given PK, c and $a, b \in \{0,1\}$, choose $r \leftarrow \mathbb{Z}_p$ and set $c^* = (c_1^*, c_2^*) = (c_1^{a(1-2\cdot b)} g^r, c_2^{a(1-2\cdot b)} \mathrm{PK}^r g^b) = (g^{r_c \cdot a(1-2\cdot b)+r}, \mathrm{PK}^{r_c \cdot a(1-2\cdot b)+r} g^{b+m\cdot a(1-2\cdot b)})$.
- **Equivocation:** Note that the ciphertext $c_1^* = g^{r_c \cdot a_\sigma(1-2\cdot b_\sigma)+r}$ and $c_2^* = \mathrm{PK}^{r_c \cdot a_\sigma(1-2\cdot b_\sigma)+r} g^{b_\sigma + m \cdot a_\sigma(1-2\cdot b_\sigma)}$ corresponds to an encryption of $m \cdot a_\sigma + b_\sigma \mod 2$. Given randomness r_c, r and $a_{1-\sigma}, b_{1-\sigma}$ output t such that $t + r_c \cdot a_{1-\sigma}(1 - 2 \cdot b_{1-\sigma}) = r + r_c \cdot a_\sigma(1 - 2 \cdot b_\sigma) \mod p$.

2.3.3 An Instantiation based on QR

In a group $\mathbb{G}$, an element $y \in \mathbb{G}$ is a quadratic residue if there exists an $x \in \mathbb{G}$ with $x^2 = y$. We denote the set of quadratic residues modulo N (an RSA composite) by $\mathcal{QR}_N$ and the set of quadratic non-residues by $\mathcal{QNR}_N$. Finally, denote by $\mathcal{QNR}_N^{+1}$ the set of quadratic non-residue modulo N with $\mathcal{J}(x) = +1$ (namely, the Jacobi symbol is $+1$).

Definition 2.8 (The Quadratic Residue (QR) Problem) *Let $N = pq$ be an RSA composite. Let qr be sampled uniformly at random from $\mathcal{QR}_N$ and let qnr be sampled uniformly at random from $\mathcal{QNR}_N^{+1}$. The QR problem asks to distinguish the distributions (N, qr) and (N, qnr).*

We next describe the Goldwasser-Micali encryption scheme [GM84] over characteristic-2 fields.

- **Key-Generation:** Choose an RSA composite (N, q, p) and a random $z \leftarrow \mathcal{QNR}_N^{+1}$. The public key is $\mathrm{PK} = \langle N, z \rangle$ and the private key is $\mathrm{SK} = \langle p, q \rangle$
- **Encryption:** Given the public key PK and a plaintext $m \in \{0, 1\}$, choose $r_c \leftarrow \mathbb{Z}_N^*$ and output the ciphertext $c = z^m \cdot r_c^2 \bmod N$.
- **Decryption:** Given the secret key SK and the ciphertext c, determine whether c is a quadratic residue modulo N using SK. If yes, output 0; otherwise, output 1.
- **Affine transformation:** Given PK, c and $a, b \in \{0, 1\}$, choose $r \leftarrow \mathbb{Z}_p$ and set $c^* = c^{a_0} \cdot z^{b_0} \cdot r^2$. Note that the ciphertext $c^* = c^{a_0} \cdot z^{b_0} \cdot r^2$ corresponds to an encryption of $a_0 m + b_0 \mod 2$.
- **Equivocation:** Given randomness r, r_c and $a_{1-\sigma}, b_{1-\sigma}$ output $t = r_c^{a_\sigma} \cdot r / r_c^{a_{1-\sigma}}$.

2.4 Oblivious Transfer from Affine Homomorphic Encryption

1-out-of-2 oblivious transfer (OT) is a functionality that is engaged between a sender Sen with a pair of inputs (s_0, s_1) and a receiver Rec with an input bit b to securely compute s_b delivered to the receiver. An Oblivious Linear-function Evaluation (OLE) over a field $\mathbb{F}$ takes a field element $x \in \mathbb{F}$ from the receiver and a pair $(a, b) \in \mathbb{F}^2$ from the sender and delivers $ax + b$ to the receiver. Note that in the case of binary fields, OLE can be realized via a single call to standard (bit-)OT.

For ease of exposition, we rely on the more intuitive notation $a \cdot \mathsf{OT}[1] + \mathsf{Enc}_{\mathrm{PK}}(b)$ instead of using $\mathsf{AT}(\mathrm{PK}, \mathsf{OT}[1], a, b)$. A 2-round OLE with semi-honest security can be constructed from an affine homomorphic encryption scheme with plaintext space $\mathbb{F}$, where the affine transformation of plaintexts corresponds to an affine operation in the field, as follows:

Protocol 1 (Oblivious Transfer from Affine Homomorphic Encryption)

Input: *The sender has inputs* $(a, b) \in \mathbb{F}$ *and the receiver has inputs* x.

$\mathrm{Rec} \rightarrow \mathrm{Sen}$: *The receiver generates keys* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and sends* $\mathsf{OT}[1] = \mathsf{Enc}_{\mathrm{PK}}(x)$ *to* Sen.

$\mathrm{Sen} \rightarrow \mathrm{Rec}$: *The sender responds with* $a \cdot \mathsf{OT}[1] + \mathsf{Enc}_{\mathrm{PK}}(b)$.

We define the defense of the receiver and sender as the input and randomness used in the protocol.

An oblivious-transfer protocol on bits can be obtained by considering $\mathbb{F} = \mathbb{F}_2$ and requiring the sender to encode it inputs (s_0, s_1) by sampling random bits $a, b \in \{0, 1\}$ such that $a \oplus b = s_1$ and $b = s_0$.

Note that the security of the above OT protocol against a corrupted sender is computational and is derived from the privacy of the underlying public-key encryption scheme, whereas the security against a corrupted receiver is statistical due to the affine homomorphism. In this work we are interested in the stronger defensible privacy notion and formalize the privacy guarantees for this OT protocol below. Specifically, we show that this protocol satisfies the two properties stated in the following propositions.

Proposition 2.1 (Privacy against defensible senders) *For every malicious* PPT *adversarial sender* Sen^**, the following two distributions are indistinguishable:*

- $\{\Gamma(\mathbf{View}_{\mathrm{Sen}^*}(\langle \mathrm{Sen}^*(s_0, s_1), \mathrm{Rec}(0) \rangle (1^\kappa))\}_{\kappa \in \mathbb{N}, s_0, s_1 \in \{0,1\}}$

- $\{\Gamma(\mathbf{View}_{\mathrm{Sen}^*}(\langle \mathrm{Sen}^*(s_0, s_1), \mathrm{Rec}(1)\rangle(1^\kappa))\}_{\kappa\in\mathbb{N}, s_0, s_1\in\{0,1\}}$

where $\Gamma(v) = v$ *only if* Sen^* *outputs a valid defense for the interaction, and is otherwise* $\perp$.

The proof of this proposition follows directly from the semantic security of the underlying encryption scheme.

Proposition 2.2 (Privacy against defensible receivers) *For every malicious* PPT *adversarial sender* Rec^**, the following two distributions are indistinguishable for* $b \in \{0,1\}$*:*

- $\{s_{1-b} \leftarrow \{0,1\}; \widetilde{s}_{1-b} \leftarrow \{0,1\}; v \leftarrow \mathbf{View}_{\mathrm{Rec}^*}(\langle \mathrm{Sen}(s_0, s_1), \mathrm{Rec}^*(b)\rangle(1^\kappa) : \Gamma(v, s_{1-b})\}_{\kappa\in\mathbb{N}, s_b\in\{0,1\}}$
- $\{s_{1-b} \leftarrow \{0,1\}; \widetilde{s}_{1-b} \leftarrow \{0,1\}; v \leftarrow \mathbf{View}_{\mathrm{Rec}^*}(\langle \mathrm{Sen}(s_0, s_1), \mathrm{Rec}^*(b)\rangle(1^\kappa) : \Gamma(v, \widetilde{s}_{1-b})\}_{\kappa\in\mathbb{N}, s_b\in\{0,1\}}$

where $\Gamma((v, x)) = (v, x)$ *only if* Rec^* *outputs a valid defense for the interaction on input* b*, and is otherwise* $\perp$.

The proof of this proposition follows directly from equivocation protocol of the affine homomorphic encryption scheme.

In our final protocol, we use a simple extension of the basic OT protocol that will admit a simulation that can equivocate its defense in addition to the two properties listed above. We describe this protocol next.

Protocol 2 (Double Oblivious Transfer)

Input: *The sender has inputs* $(s_0, s_1) \in \{0,1\}$ *and the receiver has an input* $x \in \{0,1\}$.

> $\mathrm{Rec} \to \mathrm{Sen}$: *The receiver generates keys* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa; r_{\mathsf{Gen}}), (\widetilde{\mathrm{PK}}, \widetilde{\mathrm{SK}}) \leftarrow \mathsf{Gen}(1^\kappa; \tilde{r}_{\mathsf{Gen}})$ *and sends* $\mathsf{OT}[1] = \mathsf{Enc}_{\mathrm{PK}}(x; r_{\mathsf{Enc}}), \widetilde{\mathsf{OT}}[1] = \mathsf{Enc}_{\widetilde{\mathrm{PK}}}(x; \tilde{r}_{\mathsf{Enc}})$ *to* Sen.
>
> $\mathrm{Sen} \to \mathrm{Rec}$: *The sender responds with* $\mathsf{OT}[2] = a \cdot \mathsf{OT}[1] + \mathsf{Enc}_{\mathrm{PK}}(b), \widetilde{\mathsf{OT}}[2] = \tilde{a} \cdot \widetilde{\mathsf{OT}}[1] + \mathsf{Enc}_{\widetilde{\mathrm{PK}}}(\tilde{b})$ *where* $a, \tilde{a}, b, \tilde{b}$ *are sampled randomly subject to* $b + \tilde{b} = s_0$ *and* $a + \tilde{a} = s_0 \oplus s_1$*. The receiver outputs* $\mathsf{Dec}_{\mathrm{SK}}(\mathsf{OT}[2]) \oplus \mathsf{Dec}_{\widetilde{\mathrm{SK}}}(\widetilde{\mathsf{OT}}[2])$

We define the defense of the sender as the input and randomness used in the protocol. The defense of the receiver only requires providing the input and randomness for one of the two parallel executions*, namely, either* $\mathrm{PK}, \mathrm{SK}, r_{\mathsf{Gen}}, x, r_{\mathsf{Enc}}$ *or* $\widetilde{\mathrm{PK}}, \widetilde{\mathrm{SK}}, \tilde{r}_{\mathsf{Gen}}, x, \tilde{r}_{\mathsf{Enc}}$

We remark that the proof of privacy against defensible senders holds even though the defense provides the receiver's input for only one of the two parallel executions because the sender's inputs are additively secret shared.

2.5 Tag Based Mon-Malleable Commitments

Let $\mathsf{nmcom} = \langle C, R\rangle$ be a k-round commitment protocol where C and R represent (randomized) committer and receiver algorithms, respectively. Denote the messages exchanged by $(\mathsf{nm}_1, \ldots, \mathsf{nm}_k)$ where nm_i denotes the message in the i-th round.

For some string $u \in \{0,1\}^\kappa$, tag $\mathsf{id} \in \{0,1\}^t$, non-uniform PPT algorithm M with "advice" string $z \in \{0,1\}^*$, and security parameter κ, consider the following experiment: M on input $(1^\kappa, z)$, interacts with C who commits to u with tag id; simultaneously, M interacts with $R(1^\kappa, \widetilde{\mathsf{id}})$ attempting to commit to a related value $\widetilde{u}$, again using identity $\widetilde{\mathsf{id}}$ of its choice (M's interaction with C is called the left interaction,

and its interaction with R is called the right interaction); M controls the scheduling of messages; the output of the experiment is denoted by a random variable $\mathsf{nmc}^{M}_{\langle C,R\rangle}(u,z)$ that describes the view of M in both interactions and the value $\widetilde{u}$ which M commits to R in the right execution unless $\widetilde{\mathsf{id}} = \mathsf{id}$ in which case $\widetilde{u} = \perp$, i.e., a commitment where the adversary copies the identity of the left interaction is considered invalid.

Definition 2.9 (Tag based non-malleable commitments) *A commitment scheme* $\mathsf{nmcom} = \langle C, R\rangle$ *is said to be non-malleable* with respect to commitments *if for every non-uniform* PPT *algorithm* M *(man-in-the-middle), for every pair of strings* $(u^0, u^1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$*, every tag-string* $\mathsf{id} \in \{0,1\}^t$*, every* $\kappa \in \mathbb{N}$*, every (advice) string* $z \in \{0,1\}^*$*, the following two distributions are computationally indistinguishable:*

$$\mathsf{nmc}^{M}_{\langle C,R\rangle}(u^0, z) \stackrel{c}{\approx} \mathsf{nmc}^{M}_{\langle C,R\rangle}(u^1, z)$$

Parallel non-malleable commitments. We consider a strengthening of nmcom in which M can receive commitments to m strings on the "left", say $(u_1, \ldots, u_m)$, with tags $(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ and makes m commitments on the "right" with tags $(\widetilde{\mathsf{id}}_1, \ldots, \widetilde{\mathsf{id}}_m)$. We assume that m is a fixed, possibly a-priori bounded, polynomial in the security parameter κ. In the following let $i \in [m], b \in \{0,1\}$: We say that a nmcom is an m-bounded parallel non-malleable commitment if for every pair of sequences $\{u_i^b\}$ the random variables $\mathsf{nmc}^{M}_{\langle C,R\rangle}(\{u_i^0\}, z)$ and $\mathsf{nmc}^{M}_{\langle C,R\rangle}(\{u_i^1\}, z)$ are computationally indistinguishable where $\mathsf{nmc}^{M}_{\langle C,R\rangle}(\{u_i^b\}, z)$ describes the view of M and the values $\{\widetilde{u}_i^b\}$ committed by M in the m sessions on the right with tags $\{\widetilde{\mathsf{id}}_i\}$ while receiving parallel commitments to $\{u_i^b\}$ on left with tags $\{\mathsf{id}_i\}$.

We will rely on a benign form of non-malleable commitments that is secure against "synchronizing" man-in-the middle adversaries. Where a man-in-the-middle adversary is said to be synchronous if its interaction in the non-malleable commitment on the left and right are executed in parallel (i.e. lock-step). We produce the following definitions verbatim from [Khu17].

Definition 2.10 (One-Many weak non-malleable commitments with respect to synchronizing adversaries [Khu17]) *A statistically binding commitment scheme* $\langle C, R\rangle$ *is said to be* one-many weak non-malleable with respect to synchronizing adversaries*, if there exists a probabilistic over-extractor* E_{nm} *parameterized by* ϵ*, that given a PPT synchronizing MIM which participates in one left session and* $p = \mathsf{poly}(\kappa)$ *right sessions, and given the transcript of a main-thread interaction* τ*, outputs a set of values* $m_1, m_2, \ldots m_p$ *in time* $\mathsf{poly}(n, 1/\epsilon)$*. These values are such that:*

- *For all* $j \in [p]$*, if the* j^{th} *commitment in* τ *is a commitment to a valid message* u_j*, then* $m_j = u_j$ *over the randomness of the extractor and the transcript, except with probability* ϵ/p*.*
- *For all* $j \in [p]$*, if the* j^{th} *commitment in* τ *is a commitment to some invalid message (which we will denote by* $\perp$*), then* m_j *need not necessarily be* $\perp$*.*

Definition 2.11 (Resettable reusable WI argument) *We say that a two-message delayed-input interactive argument* (P, V) *for a language* L *is* resettable reusable witness indistinguishable*, if for every PPT verifier* V^**, every* $z \in \{0,1\}^*$*,* $Pr[b = b'] \leq 1/2 + \mu(\kappa)$ *in the following experiment, where we denote the first round message function by* $m_1 = \mathsf{wi}_1(r_1)$ *and the second round message function by* $\mathsf{wi}_2(x, w, m_1, r_2)$*. The challenger samples* $b \leftarrow \{0,1\}$*.* V^* *(with auxiliary input z) specifies* $(m_1^1, x^1, w_1^1, w_2^1)$ *where* w_1^1, w_2^1 *are (not necessarily distinct) witnesses for* x^1*.* V^* *then obtains second round message* $\mathsf{wi}_2(x^1, w_b^1, m_1^1, r)$ *generated with uniform randomness* r*. Next, the adversary specifies arbitrary* $(m_1^2, x^2, w_1^2, w_2^2)$*, and obtains second round message* $\mathsf{wi}_2(x^2, w_b^2, m_1^2, r)$*. This continues* $m(\kappa) = \mathrm{poly}(\kappa)$ *times for a-priori unbounded* m*, and finally* V^* *outputs* b*.*

ZAPs (and more generally, any two-message WI) can be modified to obtain resettable reusable WI, by having the prover apply a PRF on the verifier's message and the public statement in order to generate the randomness for the proof. This allows to argue, via a hybrid argument, that fresh randomness can be used for each proof, and therefore perform a hybrid argument so that each proof remains WI. In our construction, we will use resettable reusable ZAPs.

2.6 Additive Attacks and AMD Circuits

In what follows we borrow the terminology and definitions verbatim from [GIP+14, GIW16]. We note that in this work we work with binary fields $\mathbb{F}_2$.

Definition 2.12 (Additive attack) *An additive attack* $\mathbf{A}$ *on a circuit* C *is a fixed vector of field elements which is independent from the inputs and internal values of* C*.* $\mathbf{A}$ *contains an entry for every wire of* C*, and has the following effect on the evaluation of the circuit. For every wire* ω *connecting gates* a *and* b *in* C*, the entry of* $\mathbf{A}$ *that corresponds to* ω *is added to the output of* a*, and the computation of the gate* b *uses the derived value. Similarly, for every output gate* o*, the entry of* $\mathbf{A}$ *that corresponds to the wire in the output of* o *is added to the value of this output.*

Definition 2.13 (Additively corruptible version of a circuit) *Let* $\mathrm{C} : \mathbb{F}^{I_1} \times \ldots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1} \times \ldots \times \mathbb{F}^{O_n}$ *be an* n*-party circuit containing* W *wires. We define the additively corruptible version of* C *to be the* n*-party functionality* $f^{\mathbf{A}} : \mathbb{F}^{I_1} \times \ldots \times \mathbb{F}^{I_n} \times \mathbb{F}^{W} \to \mathbb{F}^{O_1} \times \ldots \times \mathbb{F}^{O_n}$ *that takes an additional input from the adversary which indicates an additive error for every wire of* C*. For all* $(x, \mathbf{A})$*,* $f^{\mathbf{A}}(x, \mathbf{A})$ *outputs the result of the additively corrupted* C*, denoted by* $\mathrm{C}^{\mathbf{A}}$*, as specified by the additive attack* $\mathbf{A}$ *(*$\mathbf{A}$ *is the simulator's attack on* C*) when invoked on the inputs* x*.*

Definition 2.14 (Additively-secure implementation) *Let* $\varepsilon > 0$*. We say that a randomized circuit* $\widehat{\mathrm{C}} : \mathbb{F}^n \to \mathbb{F}^t \times \mathbb{F}^k$ *is an* ε*-additively-secure implementation of a function* $f : \mathbb{F}^n \to \mathbb{F}^k$ *if the following holds.*

- Completeness. *For every* $\mathbf{x} \in \mathbb{F}^n$*,* $\Pr[\mathrm{C}(\mathbf{x}) = f(\mathbf{x})] = 1$.
- Additive-attack security. *For any additive attack* $\mathbf{A}$ *there exist* $a^{\mathrm{In}} \in \mathbb{F}^n$*, and a distribution* $\mathbf{A}^{\mathrm{Out}}$ *over* $\mathbb{F}^k$*, such that for every* $x \in \mathbb{F}^n$*,* $SD(\mathrm{C}^{\mathbf{A}}(x), f(\mathbf{x} + a^{\mathrm{In}}) + \mathbf{A}^{\mathrm{Out}}) \leq \varepsilon$.

Theorem 2.15 ([GIW16], Theorem 2) *For any boolean circuit* $\mathrm{C} : \{0,1\}^n \to \{0,1\}^m$*, and any security parameter* κ*, there exists a* $2^{-\kappa}$*-additively-secure implementation* $\widehat{\mathrm{C}}$ *of* C*, where* $|\widehat{\mathrm{C}}| = \mathsf{poly}(|\mathrm{C}|, n, \kappa)$*.*

We remark that the actual theorem achieves tighter parameters, namely, better overhead than what is reported in the theorem. We state this theorem in weaker form as it is sufficient for our work.

2.7 The [BMR90] Garbling

An extension of Yao garbled circuits approach [Yao86] for any number of parties n introduced by Beaver, Micali and Rogaway in [BMR90] leading to the first constant-round protocol. This protocol has an offline phase in which the garbled circuit is created, and an online phase in which the garbled circuit is evaluated. The [BMR90] garbling technique involves garbling each gate separately using pseudorandom generators (or pseudorandom functions) while ensuring consistency between the wires. This method was recently improved by Lindell et al. in [LPSY15] which introduced an NC^0 functionality for this task, while demonstrating that the PRF values submitted by each party need not be checked for consistency (or computed by

the functionality), as inconsistency would imply an abort by at least one honest party. Moreover, an abort event is independent of the honest parties' inputs due to the way each gate is garbled. In more details, the garbling functionality $\mathcal{F}_{\mathrm{GC}}$ used in [LPSY15] is a modification of the garbling functionality introduced in [BMR90], and is applicable for any number of parties n. Namely, let C denote the circuit computed by the parties which contains W wires and a set of G gates. Then for every wire w party P_i inputs to the functionality two keys $k^i_{w,0}, k^i_{w,1}$ and the PRF computations based on these keys (see equation 1 below). Moreover, the functionality does not ensure that these values are consistent (namely, that the PRF values are computed correctly). The remaining computation is similar. Loosely speaking, the parties pick a masking bit for every wire in the computed circuit and the functionality creates the garbling for each gate which includes four rows such that each row is combined out of n ciphertexts. To be more concrete, for every wire w, each party P_i picks a wire masking value share λ^i_w so that the actual masking equals $\lambda_w = \bigoplus_{i=1}^n \lambda^i_w$. Next, for every wire w that is associated with party P_i the functionality reveals the masking bit λ_w to party P_i. Looking ahead, this phase is required so that in the online phase each party P_i can determine the public value associated with its input wires. Namely, each party P_i broadcasts $\lambda_w \oplus \rho_w$ where ρ_w is the input bit for wire w. In response, every party P_j broadcasts its key $k^j_{w,\lambda_w \oplus \rho_w}$. Upon collecting the keys from all parties, the players can start evaluating the garbled circuit. We conclude the description of the functionality with the phase where the functionality reveals λ_w for every output wire w, which constitutes the translation table and allows the parties to reconstruct the circuit's output.

We will now describe the technical details of the BMR garbling. Namely, for every NAND gate $g \in G$ with input wires $1 \leq a, b \leq W$ and output wire c, the garbled row $r_1, r_2 \in \{0,1\}$ in gate g is expressed as the concatenation of $\mathbf{R}_{g,r_1,r_2} = \{R^j_{g,r_1 r_2}\}_{j=1}^n$, where

$$R^{g,j}_{r_1 r_2} = \bigoplus_{i=1}^{n} \left(\mathsf{F}_{k^i_{a,r_1}}(g,j,r_1,r_2) \oplus \mathsf{F}_{k^i_{b,r_2}}(g,j,r_1,r_2) \right) \oplus k^j_{c,0} \oplus \left(\chi_{r_1,r_2} \wedge (k^j_{c,1} \oplus k^j_{c,0}) \right) \tag{1}$$

where F is a PRF, $k^i_{a,0}, k^i_{a,1}$ and $k^i_{b,0}, k^i_{b,1}$ are the respective input keys of party P_i, whereas $k^i_{c,0}, k^i_{c,1}$ are its output keys. Furthermore, for every a, b and r_1, r_2 as above the selector variable χ_{r_1,r_2} is defined by,

$$\chi_{r_1,r_2} = ((\lambda_a \oplus r_1) \cdot (\lambda_b \oplus r_2) \oplus 1) \oplus \lambda_c$$

such that the AND computation $\chi_{r_1,r_2} \wedge (k^j_{c,1} \oplus k^j_{c,0})$ is defined between the bit χ_{r_1,r_2} and the κ length string $(k^j_{c,1} \oplus k^j_{c,0})$ bitwise (namely, the AND of χ_{r_1,r_2} is computed with every bit in $k^j_{c,1} \oplus k^j_{c,0}$). The slightly modified offline functionality $\mathsf{BMR}_{\mathrm{Offline}}$ is formally described in Figure 4. We note that the main difference is regarding an additive error that the adversary can embed into the garbling that is captured by our functionality; see more discussion in Section 4.3.

3 Defensible Simulation

In this section we introduce a new notion that can be seen as a generalization of the notion of *defensible privacy* introduced by Haitner et al. [HIK+11]. A *defense* for a corrupted party is input and random tape provided by the adversary after the protocol execution concludes. A *defense* is considered *good* or *valid* if it explains the actions of the corrupted party in the protocol execution. Namely, a valid defense demonstrates that the messages generated by the corrupted party in the execution can be re-enacted by an honest party using the input and random tape provided in the defense. In other words, a valid defense is supposed to be a "proof" of honest behavior. Given this notion of a defense, a protocol is said to achieve *privacy against*

defensible adversaries or *defensible privacy*, if it assures that no adversary can learn anything more than the prescribed output whenever it provides a valid defense. If an invalid defense is presented, the protocol is not required to provide any guarantee.

We introduce a related yet incomparable notion of *defensible simulation* which "generalizes" defensible privacy in two ways. First, we do not require a "canonical" defense, namely the entire input and random tape as part of the defense. We will only require a "validity" predicate that will assure that whenever the defense output by the adversary satisfies the "validity" predicate privacy is assured. A second generalization is that we will provide simulation security as opposed to only privacy. In other words, given any polynomial-time adversary that corrupts a subset of the parties, there exists a polynomial-time simulator that can simulate the adversary's view in an ideal world whenever the adversary outputs a valid defense. In comparison to defensible privacy, we will relax the requirement that the adversary provides the defense at the end of computation. In our definition (and in the protocols we construct) we will achieve simulation only when the adversary outputs a valid defense before a specific round during protocol execution. This could be before the last round and therefore makes our definition incomparable to defensible privacy.[3]

We consider the standard definitions of real and ideal model for executions. See Appendix A.2 for a formal description. We now provide a formal definition of defensible privacy.

Definition 3.1 *An r-round protocol π with validity predicate* valid *is said to* securely compute f with abort in the presence of defensible adversaries *if there exists a round $r' \leq r$ such that for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [n]$, the following distributions are indistinguishable*

- $\left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)\right\}_{\kappa\in\mathbb{N},x_i,z\in\{0,1\}^*}$
- $\left\{\widetilde{\mathbf{REAL}}_{\pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)\right\}_{\kappa\in\mathbb{N},x_i,z\in\{0,1\}^*}$

where κ is the security parameter and the random variable $\widetilde{\mathbf{REAL}}$ is set to **REAL** *only if the adversary outputs a* valid *defense* def *at the end of the $(r')^{th}$-round and $\widetilde{\mathbf{REAL}} = \perp$ otherwise. A defense* def *is* valid *if* $\mathsf{valid}(P_i, \tau, \mathsf{def}_i) = 1$ *for all $i \in \mathcal{I}$ where $\mathcal{A}$ controls the parties $\mathcal{I} \subseteq [n]$,* $\mathsf{def} = \{\mathsf{def}_i\}_{i\in\mathcal{I}}$ *and τ is the transcript of the interaction until the end of the $(r')^{th}$-round.*

We remark that we will additionally extend the definition to functionalities to receive an "additive" error from the adversary in the style of [GIP+14, GIW16]. Recall from Definition 2.13 that, given any function f, we can consider an additively corruptible version $f^{\mathbf{A}}$ that takes additional input from the adversary which indicates an additive corruption for every wire of C. For all $(x, \mathbf{A})$, $f^{\mathbf{A}}(x, \mathbf{A})$ outputs the result of the additively corrupted C as specified by the additive attack $\mathbf{A}$ (where $\mathbf{A}$ is the simulator's attack on C) when invoked on the inputs x. For some of our protocols, we will achieve defensible simulation of the additively corruptible version of the function f; see the formalization in Section 2.6.

4 Warmup MPC: The Case of Defensible Simulation

4.1 Step 1: Defensibly Simulatable Protocol for $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$

In this section, we realize the three-bit product functionality with additive errors $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$. Informally, $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ is an additively corruptible version of $\mathcal{F}_{\text{MULT}}$ that additional takes as input additive attack $\mathbf{A}$ from the ad-

[3] Looking ahead, in our 4-round protocol we need to check the validity of the defense in the 3^{rd} round.

Functionality $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$

$\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ runs with parties $\mathcal{P} = \{P_1, P_2, P_3\}$ and an adversary $\mathcal{S}$ who corrupts a subset $I \subset [n]$ of parties.

1. For each $i \in \{1, 2, 3\}$, the functionality receives x_i from party P_i. If P_1 is corrupted, then it additionally receives e_{In} from $\mathcal{S}$.
2. Upon receiving the inputs from all parties, evaluate $y = (x_1 x_2 + e_{\text{In}})x_3$ and sends it to $\mathcal{S}$.
3. Upon receiving $(\mathsf{deliver}, e_{\text{Out}})$ from $\mathcal{S}$, the functionality sends $y + e_{\text{Out}}$ to all parties.

Figure 2: Additively corruptible 3-bit multiplication functionality.

versary. We will realise a restricted version of $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ that will accept additive errors e_{In} if P_1 is corrupted and e_{Out} (for any corruption scenario) and compute the function $(x_1 x_2 + e_{\text{In}})x_3 + e_{\text{Out}}$. We remark here that such a restricted class of errors will be sufficient for our final protocol. The formal description of $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ is presented in Figure 2. Our protocol relies on the following building blocks:

1. AFFINE HOMOMORPHIC ENCRYPTION: $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an affine homomorphic public-key encryption scheme over $\mathbb{F}_2$ with homomorphic addition operation $+$ (cf. Section 2.3). In fact, we rely on the two-round semi-honest oblivious transfer (cf. Section 2.4) which is based on the affine homomorphic public-key encryption scheme over $\mathbb{F}_2$.
2. ADDITIVE SECRET SHARING: $(\mathsf{Share}, \mathsf{Recover})$ (cf. Section 2.1) for sharing 0.

The details of our protocol are as follows.

Protocol 3 (3-bit Multiplication protocol Π_{DMULT})

Input: *Parties P_1, P_2, P_3 are given inputs $(x_1, s_1), (x_2, s_2, r_2)$ and x_3, respectively.*

ROUND 1:

- *Party P_1 samples $(\text{PK}_a, \text{SK}_a) \leftarrow \mathsf{Gen}$ and broadcasts $\text{PK}_a, \mathsf{OT}_\alpha[1] = \mathsf{Enc}(\text{PK}_a, x_1)$ for P_2.*
- *Party P_3 samples $(\text{PK}_\beta, \text{SK}_\beta) \leftarrow \mathsf{Gen}$ and broadcasts $\text{PK}_\beta, \mathsf{OT}_\beta[1] = \mathsf{Enc}(\text{PK}_\beta, x_3)$ for P_2.*
- *Party P_3 samples $(\text{PK}_\gamma, \text{SK}_\gamma) \leftarrow \mathsf{Gen}$ and broadcasts PK_γ for P_1.*
- *Each party P_j samples random secret shares of 0, $(z_j^1, z_j^2, z_j^3) \leftarrow \mathsf{Share}(0, 3)$ and sends z_j^i to party P_i.*

ROUND 2:

- *Party P_2 responds with $\mathsf{OT}_\alpha[2] = x_2 \cdot \mathsf{OT}_\alpha[1] - \mathsf{Enc}(\text{PK}_a, r_2)$ for P_1. P_1 computes $u = \mathsf{Dec}(\text{SK}_a, \mathsf{OT}_\alpha[2])$.*
- *Party P_2 responds with $\mathsf{OT}_\beta[2] = r_2 \cdot \mathsf{OT}_\beta[1] - \mathsf{Enc}(\text{PK}_\beta, s_2)$ for P_3. P_3 computes $v = \mathsf{Dec}(\text{SK}_\beta, \mathsf{OT}_\beta[2])$.*
- *Party P_3 broadcasts $\mathsf{OT}_\gamma[1] = \mathsf{Enc}(\text{PK}_\gamma, x_3)$ to P_1.*

ROUND 3:

- *Party P_1 broadcasts $\mathsf{OT}_\gamma[2] = u \cdot \mathsf{OT}_\gamma[1] - \mathsf{Enc}(\text{PK}_\gamma, s_1)$ for P_3. P_3 computes $w = \mathsf{Dec}(\text{SK}_\gamma, \mathsf{OT}_\gamma[2])$ and computes $s_3 = v + w$.*

ROUND 4:

- *Party P_j broadcasts $S_j = s_j + \sum_{i=1}^{3} z_i^j$.*

- OUTPUT: *All parties set the final output to* $Z = S_1 + S_2 + S_3$.

Theorem 4.1 *Protocol* Π_{DMULT} *securely computes* $\mathcal{F}_{\text{MULT}}^{\mathbf{A}}$ *(cf. Figure 2) in the presence of a defensible adversary.*

Proof: We first argue that the protocol is correct assuming no errors. Recall here that $u = x_1x_2 - r_2$, $v = x_3r_2 - s_2$ and $w = x_3u - s_1$. Therefore,

$$\begin{aligned} S_1 + S_2 + S_3 = s_1 + s_2 + s_3 = s_1 + s_2 + (v + w) &= s_1 + s_2 + (x_3r_2 - s_2) + (x_3u - s_1) \\ &= x_3r_2 + x_3(x_1x_2 - r_2) \\ &= x_1x_2x_3 \end{aligned}$$

as required. We continue with the security proof.

We first define the validity condition for the OT subprotocol used in the computation. Then, the validity of the entire protocol is defined to be the conjunction of the validity predicates for the different OT protocols that the party participates in. A defense is valid for an OT sub-protocol if it contains an input, random tape and errors that demonstrates that the actions of the adversary in the interaction were consistent with the honest strategy on that input and randomness upto additive errors. Now, we show that our protocol achieves defensible simulation up to additive errors for each corruption scenario.

We will show that we can simulate any adversary up to additive errors if a defense is presented by the adversary at the end of the third round. Additive errors can be introduced in the computation, depending on which of the parties are corrupted. In more detail, if the adversary controls party P_1, it can introduce an additive error e_{In} in the computation of $u = x_1 \cdot x_2 - r_2$ that will result in the computation of the function $(x_1 \cdot x_2 + e_{\text{In}}) \cdot x_3$. In addition, an adversary will always be able to introduce an additive error to the output of the computation by transmitting $S_j + e_{\text{Out}}$ in the final round if it controls party P_j (for any $j \in \{1, 2, 3\}$). We will demonstrate how the simulator, using the defense provided by the adversary, can simulate both the errors e_{In} and e_{Out}.

On a high-level, a random input will be chosen for each honest party and the actions of the honest party in the first three rounds will be simulated using that input. Upon receiving a valid defense for the corrupted parties at the end of the third round, the simulator extracts (from the defense) the adversary's input and additive attack e_{In} (in case P_1 is corrupted), which are sent to the ideal functionality. Upon receiving the output from the ideal functionality, the simulator generates random shares for the honest party that add up to the result and feeds them to the adversary in the fourth round. If the adversary sends its fourth round message, an additive error e_{Out} is extracted and sent to the ideal functionality with an instruction to forward the result (incorporating the error) to all parties.

We formally describe our simulation for each corruption scenario and argue correctness of simulation by showing that for every PPT adversary $\mathcal{A}$ controlling a subset of the parties I, there exists a PPT simulator $\mathcal{S}$ such that the two distributions $\widetilde{\mathbf{REAL}}_{\Pi_{\text{DMULT}},\mathcal{A}(z),I}(\kappa, x_1, x_2, x_3)$ and $\mathbf{IDEAL}_{\mathcal{F}_{\text{MULT}}^{\mathbf{A}},\mathcal{S}(z),I}(\kappa, x_1, x_2, x_3)$ are indistinguishable, where $\widetilde{\mathbf{REAL}} = \mathbf{REAL}$ only if $\mathcal{A}$ outputs a valid defense at the end of the third round and defined to be $\perp$ otherwise. The formal arguments of correctness for each corruption scenario are presented for completeness.

We argue correctness of simulation in the different corruption scenarios below.

Simulation when P_1 is corrupted. In this case, the simulator generates messages from P_2 and P_3 by providing random inputs. Next, it extracts an input x_1 and an error e_{In} that will be fed to the ideal functionality. We recall first that at the end of the third round, the simulator receives a defense from the adversary that

includes the adversary's input x_1 (used by P_1 as the receiver in the OT beginning in Round 1), and other values u, s_1 (used by P_1 as the sender for the OT executed in Round 2). If the defense is invalid then the simulator aborts. Otherwise, it proceeds.

The simulator feeds x_1, e_{In} to the ideal functionality where the intermediate error $e_{\text{In}} = u - u'$ and u' is the actual output received by P_1 in the first OT. Namely, $u' = x_1\widetilde{x}_2 - r_2$, where $\widetilde{x}_2, r_2$ were used in the simulation of P_2's message. Upon receiving the output y from the ideal functionality, the simulator samples $\widetilde{S}_2$ and $\widetilde{S_3}$ subject to $S_1 + \widetilde{S}_2 + \widetilde{S}_3 = y$ where $S_1 = s_1 + z_2^1 + z_3^1 - z_1^2 - z_1^3$. Then it feeds them to the adversary as the fourth round messages received from P_2 and P_3.

Finally, if the adversary provides its fourth round message S_1', the simulator computes $e_{\text{Out}} = S_1' - S_1$ and sends e_{Out} to the ideal functionality as the additive error for the output, and instructs it to deliver the result to P_2 and P_3.

We will prove indistinguishability of simulation via a standard hybrid argument via the following intermediate experiments:

Hybrid H_0: This experiment is the execution in the real world. The output of the experiments is defined to be $\widetilde{\mathbf{REAL}}_{\Pi_{\text{DMULT}},\mathcal{A}(z),\{P_1\}}(\kappa, x_1, x_2, x_3)$ namely the view of the adversary $\mathcal{A}$ concatenated with the output of parties P_2 and P_3.

Hybrid H_1: In this experiment, whenever the defense provided for P_1 by the adversary (x_1, u, s_1) is valid, P_3 computes s_3 differently. Namely, P_3 sets $s_3 = (x_1x_2+(u-u'))x_3-s_1-s_2$ where $u' = x_1x_2-r_2$. The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment and the output of the honest parties. The output of the honest parties in this hybrid are set to $\perp$ unless the adversary on behalf of P_1 sends the final message S_1', in which case the output of the honest parties are set to the sum of the fourth round messages from all parties.

Indistinguishability of hybrids follows from the fact that in the H_0,

$$s_3 = r_2x_3 - s_2 + ux_3 - s_1 = (x_1x_2 - u')x_3 + ux_3 - s_1 - s_2 = (x_1x_2 + (u - u'))x_3 - s_1 - s_2$$

which is the computation performed in H_1 for s_3. Since the view of the adversary is indistinguishable and the output of the honest parties is the sum of the messages exchanged in the final round (that is part of the view of the adversary), the output of the experiments H_1 and H_0 are indistinguishable.

Hybrid H_2^1: In this experiment, we switch the input of P_3 in the first of the two OTs, namely OT_β,to a random input. Observe that from the previous hybrid P_3 does not use v, the result of the first OT, in computing s_3. Indistinguishability of the hybrids follows from the privacy of the OT protocol against defensible senders (cf. Proposition 2.1) which in turn relies on the semantic security of the underlying encryption scheme.

Hybrid H_2^2: In this experiment, we switch the input of P_3 in the second of the two OTs,namely OT_γ, to a random input. Observe that from the previous hybrid P_3 does not use w, the result of the second OT, in computing s_3. Indistinguishability of the hybrids follows from the privacy of the OT protocol against defensible senders (cf. Proposition 2.1).

Hybrid H_3: In this experiment, we switch the input of P_2 from the real value x_2 to a random value $\widetilde{x}_2$. The rest of the protocol follows identically as in the previous hybrid incorporating the modification made to P_2's message. More precisely, P_3's final message is set to $(x_1x_2 + (u - u'))x_3 - s_1 - s_2$ where $u' = x_1\widetilde{x}_2 - r_2$.

Towards arguing indistinguishability of hybrids, first we observe that s_2 is distributed identically in the two hybrids. Next, we observe that u' computed as $x_1x_2 - r_2$ in the previous hybrid is distributed identically to u' computed as $x_1\widetilde{x}_2 - r_2$ in this hybrid over a random choice of r_2. Then, using the privacy of the OT protocol against defensible receivers (cf. Proposition 2.2) which in turn relies on the equivocation property of the underlying affine homomorphic encryption scheme, the distribution of the ciphertext $\mathsf{OT}_\alpha[2]$ computed as $c = x_2 \cdot \mathsf{OT}_\alpha[1] - \mathsf{Enc}(\mathrm{PK}_a, r_2)$ in H_2^2 and $c' = \widetilde{x}_2 \cdot \mathsf{OT}_\alpha[1] - \mathsf{Enc}(\mathrm{PK}_a, r_2)$ as computed in H_3 over a random choice for r_2 are statistically close.

Finally, we can conclude that the distribution of s_3 is statistically close as its computation only depends on the distribution of $(x_1, x_2, x_3, s_1, s_2, u, u')$ whose distribution is statistically close.

Hybrid H_4: This experiment is the execution in the ideal world. The output of the experiment is set as $\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{MULT}}^{\mathbf{A}}, \mathcal{S}(z), P_1}(\kappa, x_1, x_2, x_3)$. We observe that the difference between this experiment and H_3 is in the generation of P_2 and P_3's final round messages and the output of honest parties.

In the simulation, $\widetilde{S}_2$ and $\widetilde{S}_3$ are sampled randomly subject to $S_1 + \widetilde{S}_2 + \widetilde{S}_3 = y$ where $S_1 = s_1 + z_2^1 + z_3^1 - z_1^2 - z_1^3$, $y = (x_1x_2 + (u - u'))x_3$ and the output of honest parties are $y + (S_1' - S_1)$.

In H_3, $S_2 = s_2 + z_1^2 + z_2^2 + z_3^2$ and $S_3 = s_3 + z_1^3 + z_2^3 + z_3^3$, where s_2 is chosen at random and s_3 is set as $(x_1x_2 + (u - u'))x_3 - s_1 - s_2$. The output of honest parties is $S_1' + S_2 + S_3$.

We observe that $y = S_1 + S_2 + S_3 = S_1 + \widetilde{S}_2 + \widetilde{S}_3$. To conclude that H_3 and the ideal world are identically distributed, we observe that conditioned on the view of the adversary, S_2 and S_3 are uniformly distributed at random subject to $S_2 + S_3 = y - S_1$.

Simulation wihen P_2 is corrupted. In this case, the simulator generates P_1 and P_3's messages using random inputs until the end of the third round and aborts if no valid defense is output. If a valid defense is produced, the simulator collects P_2's defense x_2, r_2 and s_2 and sends x_2 to the ideal functionality. Upon receiving y from the ideal functionality, it feeds randomly sampled $\widetilde{S_1}$ and $\widetilde{S_3}$ subject to $\widetilde{S_1} + S_2 + \widetilde{S_3} = y$ as P_1 and P_3's fourth round messages, respectively where $S_2 = s_2 + z_1^2 + z_3^2 - z_2^1 - z_2^3$.

Finally, if the adversary provides its fourth round message S_2', the simulator computes $e_{\mathrm{Out}} = S_2' - S_2$ and sends e_{Out} to the ideal functionality as the additive error for the output, and instructs it to deliver the result to all parties.

We rely on the following intermediate experiments to prove indistinguishability of simulation:

Hybrid H_0: This experiment is the execution in the real world where the output of the experiment is $\widetilde{\mathbf{REAL}}_{\Pi_{\mathrm{DMULT}}, \mathcal{A}(z), \{P_2\}}(\kappa, x_1, x_2, x_3)$.

Hybrid H_1: In this experiment, given the defense provided for P_2 by the adversary (x_2, r_2, s_2), the simulator generates s_3 for P_3 as $x_1x_2x_3 - s_1 - s_2$.The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment concatenated with the output of the honest parties. The output of the honest parties in this hybrid are set to $\perp$ unless the adversary on behalf of P_1 sends the final message S_2', in which case the output of the honest parties are set to the sum of the fourth round messages from all parties.

Indistinguishability of hybrids follows from the fact that in the real world P_3 outputs

$$s_3 = v + w = r_2x_3 - s_2 + ux_3 - s_1 = r_2x_3 + (x_1x_2 - r_2)x_3 - s_1 - s_2 = x_1x_2x_3 - s_1 - s_2$$

Hybrid H_2^1: In this experiment, we switch the input of P_3 in the first of the two OTs, namely OT_β, to a random input. Indistinguishability of the hybrids follows from the privacy of OT against defensible senders.

Hybrid H_2^2: In this experiment, we switch the input of P_3 in the second of the two OTs, namely OT_γ, to a random input. Indistinguishability again follows from the defensible privacy of OT against defensible senders.

Hybrid H_3^1: In this experiment, we modify u used by P_1 in $\mathsf{OT}_\gamma[2]$to a random value. Indistinguishability follows from the defensible privacy of OT against defensible receivers.

Hybrid H_3^2: In this experiment, we switch the input of P_1 in the first OT, namely OT_α to a random input. Indistinguishability again follows from the defensible privacy of OT against defensible senders.

Hybrid H_4: This experiment is the execution in the ideal world. Indistinguishability follows analogous to the previous corruption scenario.

Simulation when P_3 is corrupted. Here P_1 and P_2 are simulated using random inputs. If a valid defense x_3 is obtained, the simulator feeds x_3 to the ideal functionality and learns y. It chooses $\widetilde{S_1}$ and $\widetilde{S_2}$ randomly subject to $\widetilde{S_1} + \widetilde{S_2} + S_3 = y$ as P_1 and P_2's fourth round message respectively, where $S_3 = s_3 + z_1^3 + z_2^3 - z_3^1 - z_3^1$, $s_3 = v + w$ and v and w are the outputs received by P_3 in the OTs.

Finally, if the adversary provides its fourth round message S_3', the simulator computes $e_{\text{Out}} = S_3' - S_3$ and sends e_{Out} as the additive error for the output to the ideal functionality and instructs it to deliver the result to all parties.

We will prove indistinguishability of simulation with the following intermediate experiments:

Hybrid H_0: This experiment is the execution in the real world where the output of the experiment is $\widetilde{\mathbf{REAL}}_{\Pi_{\text{DMULT}},\mathcal{A}(z),\{P_3\}}(\kappa, x_1, x_2, x_3)$.

Hybrid H_1: In this experiment, given P_3's defense x_3, P_1's final message is set to $x_1x_2x_3 - s_2 - s_3$ where $s_3 = v + w$, $v = r_2x_3 - s_2$ and $w = ux_3 - s_1$ are the outputs received by P_3 in the OTs. The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment concatenated with the output of the honest parties. If no fourth messages is output by the adversary, the honest parties outputs are set to $\perp$ and otherwise it is the sum of the fourth round messages exchanged by all parties.

The experiments H_0 and H_1 are identical as the final message generated for P_1 in H_1 is

$$\begin{aligned} x_1x_2x_3 - s_2 - s_3 &= x_1x_2x_3 - s_2 - (v + w) \\ &= x_1x_2x_3 - s_2 - (r_2x_3 - s_2) - (ux_3 - s_1) \\ &= x_1x_2x_3 - r_2x_3 - (x_1x_2 - r_2)x_3 + s_1 \\ &= s_1 \end{aligned}$$

which is the final message sent by P_1 in H_0.

Hybrid H_2^1: In this experiment, we switch u used by P_1 in the $\mathsf{OT}_\gamma[2]$ with P_3 to a random value $\widetilde{u}$ and set P_1's output as $x_1x_2x_3 - s_2 - s_3$ where $s_3 = v + w$ and $w = \widetilde{u}x_3 - s_1$.

Towards arguing indistinguishability, we observe that s_2 is distributed identically in the two hybrids. We also observe that the distribution of $ux_3 - s_1$ in the previous hybrid and $\widetilde{u}x_3 - s_1$ in the current

hybrid are identically distributed over a random choice of s_1. This in turn means that s_3 used in the computation of s_1 is identically distributed. Finally, from the privacy of the OT protocol against defensible receivers which in turn follows from the equivocation property of the underlying encryption scheme, the distribution of the ciphertext $\mathsf{OT}_\gamma[2]$ computed in H_1 and H_2^1 over a random choice for s_1 are statistically close.

Hybrid H_2^2: In this experiment, we switch the input of P_1 from x_1 in $\mathsf{OT}_\alpha$to a random value $\widetilde{x_1}$. Indistinguishability follows from privacy against defensible receivers for the OT protocol.

Hybrid H_3: In this experiment we switch the input of P_2 from x_2 to a random value $\widetilde{x_2}$ in the $\mathsf{OT}_\alpha[2]$interaction with P_1. This modification does not affect any of P_1's messages and the remaining computation. Therefore, indistinguishability follows from privacy against defensible senders of the OT protocol.

Hybrid H_4: This experiment is the execution in the ideal world. Indistinguishability follows by arguing that the final messages, which are the only difference between the hybrids, are identically distributed.

Simution when P_1 and P_2 are corrupted. In this case only P_3's messages are simulated using random inputs until the third round. Given the defenses of P_1 and P_2, namely (x_1, u, s_1) for P_1 and (x_2, r_2, s_2) for P_2, it sends x_1, x_2 as P_1 and P_2's inputs respectively, $e_{\text{In}} = u - u'$, where u' is the actual output received by P_1 from $\mathsf{OT}_\alpha[2]$(we remark that the transcript of interaction between all parties is broadcasted except for the shares of zero distributed in the first round) as the intermediate error to the ideal functionality. It receives y from the functionality and computes $\widetilde{S_3} = y + z_1^3 + z_2^3 - z_3^1 - z_3^2$ and feeds that to the adversary as P_3's final message.

Finally, if the adversary provides fourth round messages S_1' and S_2', the simulator computes $e_{\text{Out}} = S_1' + S_2' - S_1 - S_2$ and sends e_{Out} as the additive error for the output to the ideal functionality and instructs it to deliver the result to all parties.

We will prove indistinguishability of simulation via a standard hybrid argument with the following intermediate experiments:

Hybrid H_0: This experiment is the execution in the real world. The output of the experiment is defined to be $\widetilde{\mathbf{REAL}}_{\Pi_{\text{DMULT}},\mathcal{A}(z),\{P_1,P_2\}}(\kappa, x_1, x_2, x_3)$.

Hybrid H_1: In this experiment, given the defenses (x_1, u, s_1) of P_1 and (x_2, r_2, s_2) for P_2 the simulator computes s_3 for P_3 as $(x_1x_2 + (u - u'))x_3 - s_1 - s_2$ where $u' = x_1x_2 - r_2$. The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment and the outputs of the honest parties are set to $\perp$ unless the adversary on behalf of P_1 and P_2 sends the final messages S_1', S_2', in which case the outputs of the honest parties are set to the sum of the fourth round messages from all parties.

Indistinguishability of hybrids follows from the fact that in the real world P_3 outputs

$$s_3 = r_2x_3 - s_2 + ux_3 - s_1 = (x_1x_2 - u')x_3 + ux_3 - s_1 - s_2 = (x_1x_2 + (u - u'))x_3 - s_1 - s_2$$

Hybrid H_2^1: In this experiment, we switch the input of P_3 in the first of the two OTs, namely OT_β, to a random input. Indistinguishability of the hybrids follows from the privacy of OT against defensible senders.

Hybrid H_2^2: In this experiment, we switch the input of P_3 in the second of the two OTs, namely OT_γ, to a random input. Indistinguishability again follows from the defensible privacy of OT against defensible senders.

Hybrid H_3: This experiment is the execution in the ideal world. The output of the experiment is set as $\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{MULT}}^{\mathbf{A}},\mathcal{S}(z),\{P_1,P_2\}}(\kappa, x_1, x_2, x_3)$. Indistinguishability follows as the final message is identically distributed.

P_1 and P_3 are corrupted. Here, P_2's messages are simulated using random inputs. Upon receiving defenses (x_1, u, s_1) for P_1 and x_3 for P_3 from the adversary, it sends x_1, x_3 as P_1 and P_3's inputs respectively and $e_{\mathrm{In}} = u' - u$ where $u' = x_1x_2 - r_2$ as the intermediate error to the ideal functionality. Upon receiving y from the ideal functionality, it generates P_2's final message as $\widetilde{S_2} = y - S_1 - S_3$ where $S_1 + S_3 = s_1 + (r_2x_3 - s_2) + (ux_3 - s_1) + z_1^2 + z_1^3 - z_2^1 - z_2^3$ and feeds it to the adversary as P_2's fourth message.

Finally, if the adversary provides fourth round messages S_1' and S_3', the simulator computes $e_{\mathrm{Out}} = (S_1' - S_1) + (S_3' - S_3)$ and sends e_{Out} as the additive error for the output to the ideal functionality and instructs it to deliver the result to all parties.

We will prove indistinguishability of simulation via a standard hybrid argument with the following intermediate experiments:

Hybrid H_0: This experiment is the execution in the real world. The output of the experiment is defined to be $\widetilde{\mathbf{REAL}}_{\Pi_{\mathrm{DMULT}},\mathcal{A}(z),\{P_1,P_3\}}(\kappa, x_1, x_2, x_3)$.

Hybrid H_1: In this experiment, we switch s_2 to $(x_1x_2 + (u - u'))x_3 - s_1 - s_3$ where $s_3 = v + w$, v and w are the outputs received by P_3 in the OTs, namely $(\mathsf{OT}_\beta, \mathsf{OT}_\gamma)$, (which can be computed from the P_2's inputs and P_1 and P_3's defenses) and u' is P_1's output in its first OT_α. The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment and the output of the honest parties. If the fourth messages S_1', S_3' are not sent on behalf of corrupted parties, the output of the honest party is set to $\perp$. Otherwise, the output of P_2 is set to the sum of the fourth round messages from all parties.

The experiments H_0 and H_1 are identical as

$$\begin{aligned}(x_1x_2 + (u - u'))x_3 - s_1 - s_3 &= x_1x_2x_3 + (u - u')x_3 - s_1 - (v + w)\\ &= x_1x_2x_3 + (u - u')x_3 - s_1 - (r_2x_3 - s_2) - (ux_3 - s_1)\\ &= x_1x_2x_3 - u'x_3 - r_2x_3 + s_2\\ &= x_1x_2x_3 - (x_1x_2 - r_2)x_3 - r_2x_3 + s_2\\ &= s_2\end{aligned}$$

Hybrid H_2: In this experiment, we switch P_2's input x_2 in $\mathsf{OT}_\beta$to P_3 to a random input $\widetilde{x_2}$. Indistinguishability follows from the privacy against defensible receivers and that u' and v received by P_1 and P_3 are uniformly distributed over random choices of r_2 and s_2 independent of x_2 and $\widetilde{x_2}$ in both experiments.

Hybrid H_3: This experiment is the execution in the ideal world. Indistinguishability can be argued analogously to previous scenarios.

P_2 and P_3 are corrupted. Here, P_1's messages are simulated using random values for x_1, u and s_1. Upon receiving defenses (x_2, r_2, s_2) for P_2 and x_3 for P_3 from the adversary, it sends x_2, x_3 as P_2 and P_3's inputs respectively, to the ideal functionality and receives y as the output. It generates P_1's final message as $\widetilde{S_1} = y - S_{2,3}$ where $S_{2,3} = s_2 + (r_2x_3 - s_2) + (ux_3 - s_1) + z_1^2 + z_1^3 - z_2^1 - z_3^1$.

Finally, if the adversary provides fourth round messages S_2' and S_3', the simulator computes $e_{\mathrm{Out}} = (S_2' - S_2) + (S_3' - S_3)$ and sends e_{Out} as the additive error for the output to the ideal functionality and instructs it to deliver the result to all parties.

We will prove indistinguishability of simulation via a standard hybrid argument with the following intermediate experiments:

Hybrid H_0: This experiment is the execution in the real world. The output of the experiment is defined to be $\widetilde{\mathbf{REAL}}_{\Pi_{\mathrm{DMULT}},\mathcal{A}(z),\{P_2,P_3\}}(\kappa, x_1, x_2, x_3)$.

Hybrid H_1: In this experiment, given the defenses x_2, r_2, s_2 and x_3 for P_2 and P_3 we switch s_1 for P_1 to $x_1x_2x_3 - s_2 - s_3$ where $s_3 = v + w$, v and w are the outputs received by P_3 in OT_β and OT_γ executions, respectively. The output of the experiment is defined to be the view of the adversary $\mathcal{A}$ in the experiment and the output of the honest parties. This is set to $\perp$ if the adversary fails to provide fourth round messages for P_2 and P_3 and if they are transmitted the output is the sum of all fourth round messages.

The experiments H_0 and H_1 are identical as

$$\begin{aligned} x_1x_2x_3 - s_2 - s_3 &= x_1x_2x_3 - s_2 - (v + w) \\ &= x_1x_2x_3 - s_2 - (r_2x_3 - s_2) - (ux_3 - s_1) \\ &= x_1x_2x_3 - r_2x_3 - (x_1x_2 - r_2)x_3 + s_1 \\ &= s_1 \end{aligned}$$

Hybrid H_2: In this experiment, we switch the randomness u used by P_1 in the OT_γ with P_3 to a random value $\widetilde{u}$, and P_1's output to $x_1x_2x_3 - s_2 - s_3$ where $s_3 = v + w$ and $w = \widetilde{u}x_3 - s_1$. Indistinguishability of the experiments follows from the privacy of OT against defensible receivers and s_1 being uniformly distributed.

Hybrid H_3: This experiment is the execution in the ideal world. Indistinguishability can be proved analogously as in previous corruptions.

■

4.2 Step 2: Defensibly Simulatable Protocol for $\mathcal{F}^{\mathbf{A}}_{\mathbf{dpoly}}$

In this section, we construct a protocol Π_{dpoly} for realizing parallel evaluation of degree-3 polynomials using the protocol developed in the previous section. We will achieve simulation of defensible adversaries up to additive errors. Just as in the previous section, the adversary will only be able to introduce errors in specific intermediate values in the computation. The high-level idea is that since we rely on an additive secret-sharing scheme, any degree-3 polynomial can be computed by first computing shares for the evaluation of each monomial (of degree-3) of the polynomial and then revealing the sum of the shares received for each monomial. We will additionally require each party to share a "0" with all parties and blind their answers with the share. This is required just as in the previous protocol to guarantee privacy. The main issue however is that the local inputs used by the adversary for the individual computations of each monomial should be consistent with a global input. However, this can be easily incorporated into the validity condition of the defense. In other words, a defense for Π_{dpoly} will include a global input and local defenses where for each instance of the Π_{DMULT} the local input induced by the global input according to the specifications of $\mathcal{F}_{\mathrm{dpoly}}$ are valid for Π_{DMULT}.

Before we describe our protocol we introduce some notation.

Functionality $\mathcal{F}_{\text{dpoly}}^{\mathbf{A}}$

$\mathcal{F}_{\text{dpoly}}^{\mathbf{A}}$ runs with parties $\mathcal{P} = \{\overline{P}_1, \ldots, \overline{P}_n\}$ and an adversary $\mathcal{S}$ who corrupts a subset $I \subset [n]$ of parties. The parties have common input degree-3 polynomials $p_1, \ldots, p_M$ that describe the functionality to be evaluated.
Let $\mathsf{Eval}(t, e_{\text{In}})[\mathbf{x}_1, \ldots, \mathbf{x}_n] = (x_{t,1} \cdot x_{t,2} + e_{\text{In}}) \cdot x_{t,3}$.

Inputs: $\overline{P}_i$ has input vector $\mathbf{x}_i$.

Computation:

1. For each $i \in [n]$, party $\overline{P}_i$ sends $\mathbf{x}_i$ to the functionality. Additionally, for every monomial M_t, if the party controlling P_1 is corrupt, the adversary provides an error e_{In}^t.
2. Upon receiving the inputs from all parties, evaluate $p_1, \ldots, p_M$ as

$$y_\ell = \sum_{t \in \mathsf{Terms}_\ell} \mathsf{Eval}(t, e_{\text{In}}^t)[\mathbf{x}_1, \ldots, \mathbf{x}_n],\ \forall \ell \in [M]$$

   and send the results $y_1, \ldots, y_M$ to $\mathcal{S}$.
3. Upon receiving $(\mathsf{Deliver}, (e_{\text{Out}}^1, \ldots, e_{\text{Out}}^M))$ from $\mathcal{S}$, send $(y_1 + e_{\text{Out}}^1, \ldots, y_M + e_{\text{Out}}^M)$ to all parties.

Figure 3: Additively corruptible parallel degree-3 polynomial evaluation functionality.

Notations and conventions. We refer to the parties participating in the protocol Π_{dpoly} by $\overline{P}_1, \ldots, \overline{P}_n$, however when a subset of the parties participate in an instance of Π_{DMULT} we refer to the roles in this instance by P_1, P_2, P_3 according to Π_{DMULT}. Let $[\mathbf{x}_j]_i$ denote the i^{th} field element in $\mathbf{x}_j$. Define $\mathcal{M} = \{M_1, \ldots, M_q\}$ the multi-set that contains all monomials for each polynomial p_i, where each monomial is a product of three variables and let M denote the number of all polynomials. We assume an implicit mapping for each monomial $M_t = (x_{t,1}, x_{t,2}, x_{t,3})$ that maps t to $[\mathbf{x}_i]_\alpha, [\mathbf{x}_j]_\beta, [\mathbf{x}_k]_\gamma$ for some $i, j, k \in [n]$ and $\alpha, \beta, \gamma \in [t]$. Given the monomial $(x_{t,1}, x_{t,2}, x_{t,3})$ we assume that parties P_1,P_2 and P_3 hold $x_{t,1}$, $x_{t,2}$ or $x_{t,3}$ respectively. Let the indices corresponding to the monomials within polynomial p_ℓ denoted by the set Terms_ℓ. Finally, let $\mathsf{Role}(t, i) \in \{1, 2, 3\}$ denote the role $\overline{P}_i$ plays in the computation of M_t and is set to $\perp$ if it does not participate in the computation of M_t.

We continue with the protocol description.

Protocol 4 (Parallel polynomial protocol Π_{dpoly})

INPUT: *Parties $\overline{P}_1, \ldots, \overline{P}_n$ are given input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$.*

- ROUND 1: *Each party $\overline{P}_j$ samples M random secret shares of 0, $\{(z_{j,\ell}^1, \ldots, z_{j,\ell}^n)\}_{\ell \in [M]}$ using $\mathsf{Share}(0, n)$ and sends $(z_{j,1}^i, \ldots, z_{j,M}^i)$ to party $\overline{P}_i$.*
- ROUNDS 1,2,3: *For each monomial $M_t = (x_{t,1}, x_{t,2}, x_{t,3})$, let parties $\overline{P}_i, \overline{P}_j, \overline{P}_k$ execute Π_{DMULT} until the end of the 3^{rd} round. Let $s_{t,i}, s_{t,j}, s_{t,k}$ be the messages parties $\overline{P}_i, \overline{P}_j, \overline{P}_k$ are respectively set to broadcast in the 4^{th}-round as part of protocol Π_{DMULT}.*
- ROUND 4: *In the 4^{th} round, for every $\ell \in [M]$, $\overline{P}_j$ broadcasts $S_{\ell,j} = \sum_{t \in \mathsf{Terms}_\ell} s_{t,j} + \sum_{i=1}^n z_{i,\ell}^j$.*
- OUTPUT: *All parties output $Z_1, \ldots, Z_M$ where $Z_\ell = \sum_{j \in [n]} S_{\ell,j}$.*

Validity condition for defense. A defense for Π_{dpoly} provides an input vector $\mathbf{x}$ and defenses $(\mathsf{def}_1, \ldots, \mathsf{def}_q)$. A defense for party $\overline{P}_i$ is valid, if for every monomial M_t where $\overline{P}_j$ participates, it holds that for every i such that $\mathsf{Role}(t, i) = j$, def_t is a valid defense for party P_i with input $[\mathbf{x}]_\alpha$ where α is according to the implicit mapping from t to inputs of the parties.

Theorem 4.2 *Protocol Π_{dpoly} securely computes $\mathcal{F}_{\text{dpoly}}^{\mathbf{A}}$ in the presence of a defensible adversary corrupting any number of parties.*

Correctness follows directly from the correctness of protocol Π_{DMULT}. We continue with the security argument.

We show that any adversary can be simulated up to additive errors if it presents a defense at the end of the third round. Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $I \subset [n]$, then we prove that there exists a PPT simulator $\mathcal{S}$ with access to an ideal functionality $\mathcal{F}_{\text{dpoly}}^{\mathbf{A}}$.

More formally, the simulator $\mathcal{S}$ proceeds as follows:

- For every honest party $P_j \in \bar{I}$, $\mathcal{S}$ chooses a random input $\widetilde{\mathbf{x}}_j$ and simulates the first three rounds by following the honest strategy for party P_j.
- After the third round, if the adversary fails to provide a valid defense, the simulator halts outputting $\perp$. Otherwise, given the defenses of parties controlled by the adversary, it obtains $\{(\mathbf{x}_j, \mathsf{def}_1^j, \ldots, \mathsf{def}_q^j)\}_{j \in I}$. Next, for every $t \in [q]$, following the simulation strategy of Π_{DMULT}, it obtains the alleged share $s_{t,j}$, which P_j (controlled by the adversary) is supposed to output in the 4th round, and the error e_{In}^t if P_1 is corrupted. The simulator sends $\{\mathbf{x}_j\}_{j \in I}$ and the errors e_{In}^t to the ideal functionality.
- Upon receiving $y_1, \ldots, y_M$, the simulator generates the 4^{th} message for the honest parties as follows. First, it computes the alleged 4-th round messages that the parties controlled by the adversary are supposed to send by computing for every $\ell \in [M]$, $S_{\ell,j} = \sum_{t \in \mathsf{Terms}_\ell} s_{t,j} + \sum_{i=1}^n z_{i,\ell}^j$ where $s_{t,j}$ was extracted from the defenses in the previous step and $z_{i,\ell}^j$ are zero shares shared in the first round. Next, it computes $S_{\ell,j}$ for honest parties $P_j \in \bar{I}$ at random subject to $y_\ell = \sum_{j \in [n]} S_{\ell,j}$.

The security proof follows a sequence of intermediate hybrids where the honest parties' messages in the first three rounds are switched from being generated using the real input $\mathbf{x}_j$ to a random input $\widetilde{\mathbf{x}}_j$ for each instance of Π_{DMULT}. The intermediate hybrids to switch the inputs in an instance of Π_{DMULT} follow identically as in the proof of security of the Π_{DMULT} as presented in the previous section and according to the specific corruption scenario. For these set of hybrids, one needs the actual inputs and outputs of each Π_{DMULT} instance which is provided to the simulator in these hybrids (while noting that our final simulation will not require this information). Finally, we rely on the zero blinding shares and the correctness of the computation to prove indistinguishability of the last of these intermediate hybrids and the final simulation. More precisely, the only difference between the hybrids are how the fourth round messages are generated and the zero blinding shares can be used to argue that they will be identically distributed.

Instantiating the solution for $\mathbb{GF}_2$. The preceding two sections present protocols for products evaluation and degree-3 polynomials over any finite field. For our general functionalities, we will be relying on randomizing polynomials that will be over $\mathbb{GF}_2$ and will restrict our discussion to only $\mathbb{GF}_2$. In the preceding discussion it was assumed that each monomial is of degree-3. This is without loss of generality in the case of computing over $\mathbb{GF}_2$ because one can transform any degree-2 and degree-1 term to a degree-3 term as follows: $xy \to x^2y$ and $x \to x^3$.

4.3 Step 3: Defensibly Simulatable Protocol for Arbitrary Functionalities

We recall that from the works of [DI06, LPSY15] that securely realizing arbitrary functionalities f can be reduced to securely realizing the "BMR"-encoding of the boolean circuit C that computes f. Our starting point is the observation that the BMR encoding of a boolean circuit C is an instance of Π_{dpoly}. However, in order to tolerate the errors that are introduced from our realization of Π_{dpoly}, we will have to rely on pre-processing that will make the BMR functionality resilient to such additive attacks following the transformation of [GIW16]. More precisely, we take the following steps:

1. Let MAC_k be a family of MAC functions. We modify the function f to $f'(x, k_1, \ldots, k_n)$ defined as $(f(x), \mathsf{MAC}_{k_1}(f(x)), \ldots, \mathsf{MAC}_{k_n}(f(x)))$ where the additional inputs are such that party P_j provides k_j. In our protocol the adversary will be able adaptively add an additive error to the output of the function computed. The MAC keys will prevent any adversary from being able to do it in an undetectable way. Let C be the boolean circuit that computes f'.

2. Let $\mathsf{Enc}, \mathsf{Dec}$ be the encoding and decoding procedures of an AMD code [CDF+08] of length m. We modify C to obtain C' that takes as input x and computes $\mathsf{Enc}(\mathrm{C}(\mathsf{Dec}(x)))$.

3. We next compile C' to $\widehat{\mathrm{C}}'$ via the transformation presented in [GIW16] that will be resilient to additive attacks. Roughly, the guarantee of [GIW16] is that any additive attack on the circuit's wires can be translated into an additive attack on the input and output to the circuit (cf. Theorem 2.15).[4]

4. We describe the general BMR offline functionality $\mathsf{BMR}_{\mathrm{Offline}}$ in Figure 4 for an arbitrary polynomial sized circuit D that we are going to instantiate with $D = \widehat{\mathrm{C}}'$. Our functionality, similar to $\mathcal{F}^{\mathbf{A}}_{\mathrm{dpoly}}$ will allow for errors to be introduced by the adversary into the computation. The adversary submits two types of errors: (1) errors that can flip the keys encrypted in the garbled rows and (2) errors that will be added to the output of the function, i.e., the garbled circuit. As we show the first kind of error will be handled using the transformation of Genkin et al [GIW16] and the second kind of error will be handled using the approach of [HSS17]. [5]

5. We will consider the following encoding for the $k^j_{w,0}, k^j_{w,1}$. Each party P_j picks two functions from a pairwise independent hash function family $h^j_{w,0}, h^j_{w,1} : \{0,1\}^{4\kappa} \mapsto \{0,1\}^{\kappa}$ and encodes the output keys of each gate as $h^j_{w,\beta}(T^j_{w,\beta}) \oplus k^j_{w,\beta}$ where $\{T^j_{w,\beta}\}_{\beta\in\{0,1\}}$ are chosen uniformly at random.[6] This change is required in order to tolerate adversaries that introduce different errors on different bits of the keys. Relying on hash functions ensures that either one key is fully recovered or no key is recovered at all.[7]

[4] Given the additive attack on the wires of $\widehat{\mathrm{C}}'$, extracting the equivalent attack on the input and outputs of C' is not proven to be an efficient procedure (explicitly) in [GIP+14, GIW16]. Nevertheless the constructions in these works can be shown to satisfy this property. Roughly speaking, the first step in the circuit is to decode the input that is encoded via an AMD code, whereas the final step encodes the output using an AMD code. Therefore, the equivalent input and output errors can be identified by looking only at the input and output wires. We thank the authors of [GIW16] for clarifying this. We note that in this work we only require to extract the equivalent error on the input.

[5] The offline BMR functionality from [HSS17] allows the adversary to submit an error, chosen adaptively by the adversary after seeing the garbled circuit, which is added to the garbled circuit by the functionality.

[6] This approach is analogous to realizing string OTs via bit OTs.

[7] We remark that it is possible to view the BMR polynomials as a degree-3 computation over a large field. In this case, the error cannot affect the individual parts of the key as we can use the key as a single element in the multiplication. For simplicity and for our final protocol, we restrict ourselves to binary fields.

Functionality $\mathsf{BMR}_{\text{Offline}}$

Let $F = \{\mathsf{F}_k : \{0,1\}^\kappa \to \{0,1\}^{4\kappa}\}_{k\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be a pseudorandom function ensemble. The functionality runs with parties $P_1,\ldots,P_n$ and an adversary $\mathcal{A}$ corrupting a set of parties $I \subset [n]$. The parties joint input is a boolean circuit D, expressed by a set of wires W and a set of NAND gates G.

Inputs: Party P_j inputs the following for every wire $w \in W$:

- A pair of strings $(T^j_{w,0}, T^j_{w,1})$ each of length of 4κ.
- Purported PRF values from P_i, $\mathsf{F}_{k^i_{a,r_1}}(g,j,r_1,r_2), \mathsf{F}_{k^i_{b,r_2}}(g,j,r_1,r_2)$ for every gate $g \in G$ with input wires $a,b \in W$, every $r_1,r_2 \in \{0,1\}$ and every $i \in [n]$.
- A random bit λ^j_w if w is not an input wire.
- An additive error $e^{g,j}_{r_1,r_2}$ bit for every $g \in G, r_1,r_2 \in \{0,1\}$ and $j \in I$.

Computation:

- For every wire $w \in W$ that is an output wire the functionality sets $\lambda_w = \oplus_{i=1}^n \lambda^i_w$.
- For every NAND gate $g \in G$ with input wires $1 \le a,b \le W$ and output wire c, define the selector variables
$$\chi^g_{r_1,r_2} = ((\lambda_a \oplus r_1)\cdot(\lambda_b \oplus r_2) \oplus 1) \oplus \lambda_c.$$
- Set $\mathbf{R}^g_{r_1,r_2} = R^{g,1}_{r_1,r_2} || \cdots || R^{g,n}_{r_1,r_2}$ for all $1 \le j \le n$ and $r_1,r_2 \in \{0,1\}$ as follows:
$$R^{g,j}_{r_1,r_2} = \Big(\bigoplus_{i=1}^n \mathsf{F}_{k^i_{a,r_1}}(g,j,r_1,r_2)\Big) \oplus \Big(\bigoplus_{i=1}^n \mathsf{F}_{k^i_{b,r_2}}(g,j,r_1,r_2)\Big) \oplus T^j_{c,0} \oplus \Big((\chi^g_{r_1,r_2} \oplus e^{g,j}_{r_1,r_2}) \wedge (T^j_{c,1} \oplus T^j_{c,0})\Big).$$

Open Garbling:

- On receiving Open Garbling from all parties the functionality transmits to the adversary $\mathcal{A}$ the garbling $\{R^g_{r_1,r_2}\}_{g\in G, r_1,r_2\in\{0,1\}}$ as well as λ_w for all input wires $w \in W$ which are attached to a corrupted party P_i, and all output wires.
- If $\mathcal{A}$ aborts, then the functionality halts.
- If $\mathcal{A}$ responds with $(\mathsf{OK}, e_{\text{Out}} = \{e^g_{r_1,r_2}\}_{r_1,r_2\in\{0,1\},g\in G})$, the functionality sends to all parties $R^g_{r_1,r_2} \oplus e^g_{r_1,r_2}$ for all $g \in G$ and $r_1,r_2 \in \{0,1\}$.

Figure 4: The BMR garbling functionality.

Recall from Section 2.7 that every garbled row (r_1,r_2) for any gate g in the computation of $\mathsf{BMR}_{\text{Offline}}$ can be expressed as the concatenation of $\{R^{g,j}_{r_1r_2}\}_{j=1}^n$, where (without including errors)

$$R^{g,j}_{r_1r_2} = \Big(\bigoplus_{i=1}^n \mathsf{F}_{k^i_{a,r_1}}(g,j,r_1,r_2)\Big) \oplus \Big(\bigoplus_{i=1}^n \mathsf{F}_{k^i_{b,r_2}}(g,j,r_1,r_2)\Big) \oplus T^j_{c,0} \oplus \chi_{r_1,r_2}(T^j_{c,1} \oplus T^j_{c,0})$$

and

$$\chi_{r_1,r_2} = [(\lambda^1_a \oplus \cdots \oplus \lambda^n_a \oplus r_1)\cdot(\lambda^1_b \oplus \cdots \oplus \lambda^n_b \oplus r_2) \oplus 1] \oplus (\lambda^1_c \oplus \cdots \oplus \lambda^n_c).$$

where F_k is a PRF-family and λ_w^i are the masks (for the color bit) sampled by party P_i for wire w, $r_1, r_2 \in \{0,1\}$

Observe that χ_{r_1,r_2} is a degree-2 computation, which in turn means the expressions $T_{c,0}^j \oplus \chi_{r_1,r_2}(T_{c,1}^j \oplus T_{c,0}^j)$ over all garbled rows is a collection of polynomials of degree at most 3. In particular, for every $j \in [n]$, every every gate $g \in G$ with input wires a, b and an output wire c, $R_{r_1 r_2}^{g,j}$ involves the computation of one or more of the following monomials:

- $\lambda_a^{j_1} \lambda_b^{j_2} (T_{c,1}^j \oplus T_{c,0}^j)$ for $j, j_1, j_2 \in [n]$.
- $\lambda_c^{j_1} (T_{c,1}^j \oplus T_{c,0}^j)$ for $j, j_1 \in [n]$.
- $T_{c,0}^j$.
- PRF values.

Convention of roles in the multiplication protocol. We first describe some convention regarding how each multiplication triple is computed, namely assign parties with roles P_1, P_2 and P_3 in Π_{DMULT} (Section 4.1), and what products are computed. Letting $\Delta_c^j = (T_{c,1}^j \oplus T_{c,0}^j)$, we observe that every product always involves Δ_c^j as one of its operands. Moreover, every term can be expressed as a product of three operands, where the product $\lambda_c^{j_1} \Delta_c^j$ will be (canonically) expressed as $(\lambda_c^{j_1})^2 \Delta_c^j$ and singleton monomials (e.g., the bits of the keys and PRF values) will be raised to degree 3. Then, for every polynomial involving the variables $\lambda_a^{j_1}, \lambda_b^{j_2}$ and Δ_c^j, party P_j will be assigned with the role of P_3 in Π_{DMULT} whereas the other parties P_{j_1} and P_{j_2} can be assigned arbitrarily as P_1 and P_2. In particular, the roles are chosen so as to restrict the errors introduced by a corrupted P_1 in the computation to only additive errors of the form $e_{\text{In}}\delta$ where δ is some bit in Δ_c^j, where it follows from our simulation that e_{In} will be independent of δ for honest P_j.

Given these conventions, the output of the BMR garbled circuit can be expressed as a sequence of N degree-3 polynomials $p_1, \ldots, p_N$. We can thus use Π_{dpoly} to compute the BMR garbling by adding the PRF values before broadcasting it in the fourth round. More formally,

Protocol 5 (Defensibly-simulatable protocol Π_f)

INPUT: *Parties $P_1, \ldots, P_n$ are given input $x_1, \ldots, x_n$ each of length κ, respectively, and a circuit $\widehat{\mathrm{C}}'$ as specified above. We fix the notation $[x_i]_j$ as the j^{th} bit of string x_i.*

- ROUNDS 1,2,3: *For each $i \in [M]$, parties $P_1, \ldots, P_n$ execute Π_{dpoly} for the polynomial p_i up until the 3^{rd} round of the protocol with random inputs for the $\mathcal{F}_{\text{dpoly}} = \mathsf{BMR}_{\text{Offline}}$. Along with the message transmitted in the 3^{rd} round of Π_{dpoly}, party P_j broadcasts the following:*
  - *For every input wire $w \in W$ that carries some input bit $[x_j]_k$ from P_j's input, P_j broadcasts $\Lambda_w = \lambda_w \oplus [x_j]_k$.*

  For every $j \in [n]$, let $\{S_{\ell,j}\}_{\ell \in M}$ be the output of party P_j for the M polynomials. It reassembles the output shares to obtain $\widetilde{S}_{r_1,r_2}^{g,j}$ for every garbled row.
- ROUND 4: *Finally for every gate $g \in G$ and $r_1, r_2 \in \{0,1\}$, P_j ($j \in [n]$) broadcasts the following:*
  - $\widetilde{R}_{r_1,r_2}^{g,i} = \mathsf{F}_{k_{a,r_1}^j}(g, j, r_1, r_2) \oplus \mathsf{F}_{k_{b,r_2}^j}(g, i, r_1, r_2) \oplus \widetilde{S}_{r_1,r_2}^{g,i}$ *for every $i \in [n]$.*
  - k_{w,Λ_w}^j *for every input wire w.*
  - λ_w^j *for every output wire w.*

– $(\Gamma^j_{w,0}, \Gamma^j_{w,1}) = (h(T^j_{w,0}) \oplus k^j_{w,0}, h(T^j_{w,1}) \oplus k^j_{w,1})$ *for every wire* w.

- OUTPUT: *Upon collecting* $\{\widetilde{R}^{g,j}_{r_1 r_2}\}_{j\in[n],g\in[G],r_1,r_2\in\{0,1\}}$, *the parties compute each garbled row by* $R^{g,j}_{r_1,r_2} = \bigoplus_{j=1}^n \widetilde{R}^{g,j}_{r_1,r_2}$. *Then using the keys corresponding to the input wires* w, *they evaluate the circuit to obtain* Λ_w *for every output wire as follows:*

  Consider a standard (arbitrary) topological ordering of the gates. The parties will evaluate the circuit according to the topological order. Let g *be a gate in this order with input wires* a, b *and output wire* c. *If a party does not have masks* Λ_a, Λ_b *or keys* (k_a, k_b) *corresponding to the input wires when processing gate* g *it aborts. Otherwise, it will compute*

$$T^j_c = R^{g,j}_{r_1,r_2} \oplus \bigoplus_{i=1}^{n} \left(\mathsf{F}_{k^i_{a,\Lambda_a}}(g, j, \Lambda_a, \Lambda_b) \oplus \mathsf{F}_{k^i_{b,\Lambda_b}}(g, j, \Lambda_a, \Lambda_b)\right)$$

  where $k_a = (k^1_a, \ldots, k^n_a)$ *and* $k_b = (k^1_b, \ldots, k^n_b)$. *Party* P_j *identifies* Λ_c *such that* $T^j_c = T^j_{c,\Lambda_c}$. *If no such* Λ_c *exists the party aborts. Otherwise, each party defines* $k^i_c = \Gamma^i_{c,\Lambda_c} \oplus h(T^j_c)$. *Let* $k_c = (k^1_c, \ldots, k^n_c)$. *The evaluation is completed when all gates in the topological order are processed. Finally given* Λ_w *for every output wire, the parties compute the output carried in wire* w *as* $\Lambda_w \oplus \left(\bigoplus_{j=1}^n \lambda^j_w\right)$ *and decode the outcome using* Dec.

This concludes the description of our protocol. We next prove the following theorem,

Theorem 4.3 (Defensible simulation) *Let* C *be an* n*-party circuit that computes* f *and assume the existence of a 2-round OT* Π_{OT} *(as specified in Section 2.4). Then Protocol* Π_f *securely computes* f *in the presence of a defensible adversary that corrupts at most* $n-1$ *parties.*

Proof overview: Informally, our proof will proceed following a standard set of hybrid experiments that change the garbled circuit obtained from the BMR garbling from a real garbling to a simulated garbling. The main difference is that in our protocol the adversary has the capability of introducing errors in the computation of some of the garbled rows. Recall that, using [GIW16], we first encode the computation to be resilient to such attacks where any error introduced by the adversary will have the effect of introducing an additive error independent of the inputs in the BMR garbling. Another source of error is due to the ability of the (rushing) adversary to choose its shares based on the garbled circuit. A generic way to handle this would be to rely on delayed-input non-malleable zero-knowledge proofs [COSVb]. We will instead rely on the fact that the garbled circuit itself is resilient to such additive attacks (this was proved formally in [HSS17]). In the previous section we defined Π_{dpoly} and prove how to defensibly simulate parallel invocations of Π_{DMULT}. On a high-level, we will use the simulation of Π_{dpoly} to enforce a particular (fake) garbling outcome and prove indistinguishability relying on the pseudorandomness of the PRF values.

We continue with the complete proof.

Proof: Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $I \subset [n]$, then we prove that there exists a PPT simulator $\mathcal{S}$ with access to an ideal functionality $\mathcal{F}$ that implements f, and simulates the adversary's view whenever it outputs a valid defense at the end of the third round. We use the terminology of active keys to denote the keys of the BMR garbling that are revealed during the evaluation. Inactive keys are the hidden keys. Denoting the set of honest parties by $\bar{I}$, our simulator $\mathcal{S}$ is defined below.

The description of the simulation.

- Recall that the parties engage in an instance of Π_{dpoly} to realized the $\mathsf{BMR}_{\text{Offline}}$ functionality in the first three rounds. The simulator samples random inputs for honest parties and generates the messages of the honest parties honestly using the random inputs. For every input wire that is associated with an honest party's input, the simulator chooses a random Λ_w and sends these bits to the adversary as part of the 3^{rd} message. At this point if the adversary outputs a valid defense it proceeds, and otherwise aborts. From this defense, the simulator obtains λ_w^j and $k_{w,0}^j, k_{w,0}^j \oplus k_{w,1}^j$ for every corrupted party P_j and wire w that is an output of some NAND gate, as well as the PRF values. Finally, it obtains the error $e_{r_1,r_2}^{g,j}$ for every gate g, $r_1, r_2 \in \{0,1\}$ and $j \in I$, where $e_{r_1,r_2}^{g,j}$ is a vector of errors introduced by the adversary for the plaintext encrypted in row (r_1, r_2) in the garbling of gate g.[8]

- Next, the simulator chooses a random $\Lambda_w \leftarrow \{0,1\}$ for every internal wire $w \in W$ that is an output of some NAND gate. It further samples a single key k_w^j for every honest party $P_j \in \bar{I}$ and wire $w \in W$.

- Upon receiving the adversary's public values for its input bits, the simulator extracts the adversary's input. Namely, for each input wire $w \in W$ that is associated with a corrupted party's input, the simulator computes $\rho_w = \Lambda_w \oplus \lambda_w$ and the errors in the input wires and fixes the adversary's input $\{\mathbf{x}_I\}$ to be the concatenation of these bits incorporating the errors. $\mathcal{S}$ sends $\mathsf{Dec}(\mathbf{x}_I)$ to the trusted party computing f, receiving the output $\tilde{y}$. $\mathcal{S}$ fixes $y = \mathsf{Enc}(\tilde{y})$ where recall $\mathsf{Enc}, \mathsf{Dec}$ are the encoding and decoding functions corresponding to an AMD code. Let $y = (y_1, \ldots, y_m)$.

- For every gate g, given Λ_a, Λ_b, we define $\widetilde{\Lambda}_c$, where a, b are the input wires and c is the output wire of g. Pick a random honest party P_j. If majority of the bits of $e_{r_1,r_2}^{g,j} = 1$ then we set $\widetilde{\Lambda}_c = 1 \oplus \Lambda_c$, and otherwise set $\widetilde{\Lambda}_c = \Lambda_c$. To simulate the output of the garbled row given the errors, we sample T_{c,Λ_c}^j at random and construct T_c^j such that it is equal to the bits of T_{c,Λ_c}^j where $e_{r_1,r_2}^{g,j} = 0$ and at random if $e_{r_1,r_2}^{g,j} = 1$. For every gate g, and honest party P_j, $\mathcal{S}$ constructs $Y_{\Lambda_a,\Lambda_b}^{g,j} = (e_{\Lambda_a,\Lambda_b}^{g,j} \oplus T_c^j)$. For corrupted parties P_j, $\mathcal{S}$ sets $Y_{\Lambda_a,\Lambda_b}^{g,j} = (e_{\Lambda_a,\Lambda_b}^{g,j} \oplus T_{c,\widetilde{\Lambda}_c}^j)$. For the t^{th} output wire w, $\mathcal{S}$ defines $(\bigoplus_{i=1}^n \lambda_w^i) = \Lambda_w \oplus y_t$.

  Denote the simulated garbled circuit by $\mathrm{GC}_{\mathcal{S}}$. On behalf of every honest party P_j, $\mathcal{S}$ broadcasts $(r, h(T_{w,\Lambda_w}^j) \oplus k_w^j)$ if $\Lambda_w = 1$ and $(h(T_{w,\Lambda_w}^j) \oplus k_w^j, r)$ if $\Lambda_w = 0$ where r is sampled randomly.

- Next, the simulator provides the fourth message on behalf of the honest parties to the adversary. Namely, for every active row, i.e. for every gate g, the row Λ_a, Λ_b, the shares of the honest parties are computed assuming the output of the polynomials defined in $\mathsf{BMR}_{\text{Offline}}$ are $Y_{\Lambda_a,\Lambda_b}^{g,i}$ for every i masked with the PRF under the keys k_a^j, k_b^j as $\tilde{R}_{\Lambda_a,\Lambda_b}^{g,j}$. For the remaining three rows the simulator sends random strings.

- If the adversary provides its fourth message $\{\widetilde{R}_{r_1 r_2}^{g,j}\}_{j\in[n],g\in[G],r_1,r_2\in\{0,1\}}$, the simulator reconstructs the garbling $\mathrm{GC}_{\mathcal{A}}$ and evaluates it on behalf of the honest parties. If one of the following events does not occur then the simulator sends $\perp$ to the trusted party computing f. First, the simulator checks that the output key of every key obtained during the evaluation is the active key k_{c,Λ_c}^j encrypted by the simulator. In addition, the simulator checks that the outcome of $\mathrm{GC}_{\mathcal{A}}$ is y. Otherwise, the simulator sends an OK message to the trusted party to deliver $\tilde{y}$ to the honest parties.

[8] The errors are bits and are extracted for each monomial where the corrupted party plays the role of P_1. For simplicity, we can collect all the errors for the polynomial representing each bit in the garbled row and then express it as a vector corresponding to each key.

Claim 4.1 $\widetilde{\mathbf{REAL}}_{\pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n) \approx \mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)$.

Proof: We consider a sequence of intermediate experiments and prove indistinguishability of simulation against defensible adversaries via standard hybrid arguments.

Hybrid H_0**:** This is the real experiment. Let $\widetilde{\mathbf{REAL}}_{\pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)$ denote the output of this experiment.

Hybrid H_1**:** In this hybrid, we follow the simulation strategy of Π_{dpoly}, namely, sampling a random global inputs for the honest parties, and using the honest strategy with the sampled inputs. To sample the fourth round we continue to use the simulation strategy of Π_{dpoly} to first generate the shares $S_{t,j}$ that each honest party outputs for each term t and then use it to generate the fourth message according to honest strategy. Recall that the simulation strategy of Π_{dpoly} requires the outputs of each individual polynomial and the defenses of the corrupted parties. The defense for this protocol is defined identically as for Π_{dpoly} and to generate the outputs of the polynomials the simulator emulates $\mathcal{F}_{\text{dpoly}}$. The inputs of the corrupted parties in this emulation is provided by the simulator of Π_{dpoly} using the defense supplied by the adversary and the uncorrupted parties, the simulator uses the original global inputs sampled for them. Using the simulated shares $S_{t,j}$, the simulator continues the honest strategy using $S_{t,j}$ to generated the fourth message. Indistinguishability of H_1 from H_0 follows essentially from the simulation strategy for Π_{dpoly}. Concretely, we need to consider all intermediate hybrids used in the proof of security of Π_{dpoly}.

Next, we consider an arbitrary topological ordering on the gates $g \in G$ of the circuit. We will inductively consider a sequence of hybrids H_2^g in this ordering. We will maintain as invariant that while at gate g, the simulation does not require the knowledge of an inactive key denoted by $k^j_{w,\bar{\Lambda}_w}$ for every honest party P_j, where w is an output of a previously considered gate in this ordering. It follows from the construction that this holds for all gates in the input layer of the circuit (i.e. carrying the real inputs of the parties).

Hybrid H_2^g**:** Consider an arbitrary gate g in the topological order with input wires a, b and output wire c. By the topological order, $k^j_{a,\bar{\Lambda}_a}$ and $k^j_{b,\bar{\Lambda}_b}$ are not used in the garbling of the gates from the set $[g-1]$. Therefore, we can replace the three inactive rows that involve a PRF value under these keys with a random string where indistinguishability follows from the pseudorandomness of the PRF.

Hybrid $\widetilde{H}_2^g$**:** Consider an honest party P_j. We recall that the keys are encrypted by $h^j_{c,\beta}(T^j_{c,\beta}) \oplus k^j_{c,\beta}$ using a pairwise independent hash function and a public random string $T^j_{c,\beta}$ of length 4κ. We note here that depending on which bits of $e^{g,j}_{r_1,r_2}$ are 0 and 1 bits, the adversary learns the bits from the respective positions in $T^j_{c,0}$ and $T^j_{c,1}$ respectively when decrypting the garbled row (Λ_a, Λ_b) of g. If the adversary learns less than 2κ bits of some string $T^j_{c,\beta}$, it is ensured by the leftover hash lemma that except with probability $2^{-\kappa}$, the key $k^j_{c,\beta}$ is hidden. If a majority of the bits in $e^{g,j}_{r_1,r_2}$ are 0 then we set $\Lambda_c = \widetilde{\Lambda_c}$ and otherwise we set $\Lambda_c = 1 \oplus \widetilde{\Lambda_c}$ where

$$\widetilde{\Lambda}_c = \text{NAND}(\rho_a, \rho_b) \oplus (\bigoplus_{i=1}^{n} \lambda^i_c) \text{ where}$$

$$\rho_a = \Lambda_a \oplus (\bigoplus_{i=1}^{n} \lambda^i_a), \;\; \rho_b = \Lambda_b \oplus (\bigoplus_{i=1}^{n} \lambda^i_b).$$

To simulate the output of the garbled row given the errors, we sample T^j_{c,Λ_c} at random and set the bits of $T^j_{c,\bar{\Lambda}_c}$ where $e^{g,j}_{r_1,r_2} \neq \beta_j$ randomly. We further replace the value $h^j_{c,\bar{\Lambda}_c}(T_{c,\bar{\Lambda}_c}) \oplus k_{c,\bar{\Lambda}_c}$ with a random string from $\{0,1\}^\kappa$. From the pairwise independence we can conclude that $\widetilde{H}^g_3$ and H^g_3 are statistically close.

Hybrid H_3: In this hybrid, we modify the λ^j_w shares of the honest parties $P_j \in \bar{I}$ for the output wires. Recall that these values are revealed in the fourth round and influence the "translation table" (namely, the output is identified by $\Lambda_w \oplus (\oplus^n_{j=1}\lambda^j_w)$). In the previous hybrid these values were honestly revealed as in the real world. In this hybrid we first extract the input of the adversary and its additive errors on the circuit wires and evaluate $\widehat{\mathrm{C}}'$ under this input while incorporating the errors on the wires, and finally fix the output to be the result of this computation.

In more detail, in order to extract the adversary's input for each input wire $w \in W$ that is associated with a corrupted party's input, the simulator computes $\rho_w = \Lambda_w \oplus \lambda_w$ and fixes the adversary's input $\{\mathbf{x}_I\}$ to be the concatenation of these bits incorporating the errors. Next, to extract the errors on the intermediate wires of the computation under $\widehat{\mathrm{C}}'$, the simulator computes $e_w = \widetilde{\Lambda_w} \oplus \Lambda_w$ where $\widetilde{\Lambda_w}$ corresponds to the "correct value" as defined below. Finally, to fix the output of the computation, let $y = (y_1, \ldots, y_m)$ be the result of the computation. Then, for the t^{th} output wire w, $\mathcal{S}$ samples shares λ^j_w for the honest parties $P_j \in \bar{I}$ subject to $(\bigoplus^n_{i=1} \lambda^i_w) = \Lambda_w \oplus y_t$.

The outputs of this hybrid are identically distributed to the outputs in the previous hybrid due to the following correctness argument.

By construction the actual computation performed while evaluating the garbled circuit matches the computation under $\widehat{\mathrm{C}}'$ with errors $\mathbf{A} = \{e_w\}_w$.

Hybrid H_5: In this hybrid we follow the simulation strategy. Namely, the simulator extracts the adversary's input, sends $\mathsf{Dec}(\mathbf{x}_I)$ to the trusted party computing f and receives the output $\tilde{y}$. $\mathcal{S}$ fixes $y = \mathsf{Enc}(\tilde{y})$. Let $y = (y_1, \ldots, y_m)$. Moreover, $\mathcal{S}$ defines $(\bigoplus^n_{i=1} \lambda^i_w) = \Lambda_w \oplus y_t$ for the t^{th} output wire w. Statistical indistinguishability of this experiment from the previous experiment follows directly from the additive security of $\widehat{\mathrm{C}}'$ as established in [GIW16]]. Furthermore, this hybrid produces a distribution according to $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)$.

□

This concludes the proof.

■

5 4-Round Actively Secure Multi-Party Computation

In this section we prove our main theorem. Towards that, we take the following steps:

1. First, we consider protocol Π_{DMULT} where the oblivious-transfer protocol is instantiated differently. More precisely, each OT performed in Π_{DMULT} relying on the affine homomorphic encryption scheme is replaced with one where the parties engage in two executions of the OT protocol in parallel. The receiver uses its original input in both instance whereas the sender secret shares its input across the two instances. We describe this modified protocol Π_{R3MUL} in Section 5.1 and define validity for defenses in Figure 5.

2. Next, we design a new protocol to realize $\mathcal{F}_{\text{dpoly}}$. In the previous section, we designed a protocol that achieved security against defensible adversaries. The protocol in this section will additionally be proven secure against (fully) active adversaries. Moreover, we will produce a simulator that can additionally output a defense at the end of the third round of the protocol.

3. Finally, we modify the protocol in Section 4.3 where we replace Π_{dpoly} with Π_{ppoly} from Step 2 and conclude our main theorem. We remark that the simulation of our final protocol is not achieved my modularly composing the protocol presented in this section with simulation provided against defensible adversaries in Section 4.3. Instead, we will directly rely on the simulation strategy used in that section combined with the simulator for Π_{ppoly} in this section (that additionally provides a defense) to achieve simulation of the complete protocol.

Following these steps, we obtain the following theorem.

Theorem 5.1 (Main) *Assuming the existence of affine homomorphic encryption (cf. Definition 2.5), there exists a 4-round multi-party protocol that securely realizes arbitrary functionalities in the presence of static, active adversaries corrupting any number of parties.*

The complete proof is found in Section 6 and proof overview in Section 5.2.

5.1 Modified 3-bit Multiplication Protocol Π_{R3MUL}

Below we describe our modified 3-bit multiplication protocol.

Protocol 6 (Double 3-bit Multiplication protocol Π_{R3MUL})

For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment and encryption functions, unless specified explicitly.

Input: *Parties P_1, P_2, P_3 are given input $(x_1, s_1), (x_2, s_2)$ and (x_3), respectively.*

ROUND 1:

- *Party P_1 runs the key generation algorithm* Gen *twice and samples* $(\text{PK}_a, \text{SK}_a)$ *and* $(\text{PK}_{\widetilde{a}}, \text{SK}_{\widetilde{a}})$. *Next it computes ciphertexts* $\mathsf{OT}_\alpha[1] = \mathsf{Enc}_{\text{PK}_a}(x_1)$ *and* $\mathsf{OT}_{\widetilde{\alpha}}[1] = \mathsf{Enc}_{\text{PK}_{\widetilde{a}}}(x_1)$ *and sends* $((\text{PK}_a, \mathsf{OT}_\alpha[1]), (\text{PK}_{\widetilde{a}}, \mathsf{OT}_{\widetilde{\alpha}}[1]))$ *to* P_2.
- *Party P_3 runs the key generation algorithm* Gen *twice and samples* $(\text{PK}_\beta, \text{SK}_\beta)$ *and* $(\text{PK}_{\widetilde{\beta}}, \text{SK}_{\widetilde{\beta}})$. *Next it computes ciphertexts* $\mathsf{OT}_\beta[1] = \mathsf{Enc}_{\text{PK}_\beta}(x_3)$ *and* $\mathsf{OT}_{\widetilde{\beta}}[1] = \mathsf{Enc}_{\text{PK}_{\widetilde{\beta}}}(x_3)$ *and sends* $((\text{PK}_\beta, \mathsf{OT}_\beta[1]), (\text{PK}_{\widetilde{\beta}}, \mathsf{OT}_{\widetilde{\beta}}[1]))$ *to* P_2.
- *Party P_3 runs the key generation algorithm* Gen *twice and samples* $(\text{PK}_\gamma, \text{SK}_\gamma)$ *and* $(\text{PK}_{\widetilde{\gamma}}, \text{SK}_{\widetilde{\gamma}})$. *Next it broadcasts* PK_γ *and* $\text{PK}_{\widetilde{\gamma}}$.

ROUND 2:

- *Party P_2 samples $x_\alpha, x_{\widetilde{\alpha}}$ such that $x_\alpha + x_{\widetilde{\alpha}} = x_2$ and $s_\alpha, s_{\widetilde{\alpha}}$ such that $s_\alpha + s_{\widetilde{\alpha}} = s_2$. It computes* $\mathsf{OT}_\alpha[2] = x_\alpha \cdot \mathsf{OT}_\alpha[1] + \mathsf{Enc}_{\text{PK}_a}(s_\alpha)$ *and* $\mathsf{OT}_{\widetilde{\alpha}}[2] = x_{\widetilde{\alpha}} \cdot \mathsf{OT}_{\widetilde{\alpha}}[1] + \mathsf{Enc}_{\text{PK}_{\widetilde{a}}}(s_{\widetilde{\alpha}})$ *then responds with* $(\mathsf{OT}_\alpha[2], \mathsf{OT}_{\widetilde{\alpha}}[2])$. *$P_1$ computes* $u = \mathsf{Dec}_{\text{SK}_a}(\mathsf{OT}_\alpha[2]) + \mathsf{Dec}_{\text{SK}_{\widetilde{a}}}(\mathsf{OT}_{\widetilde{\alpha}}[2])$.
- *Party P_2 samples $r_\beta, r_{\widetilde{\beta}}$ such that $r_\beta + r_{\widetilde{\beta}} = r$ and $s_\beta, s_{\widetilde{\beta}}$ such that $s_\beta + s_{\widetilde{\beta}} = s_2$. It computes* $\mathsf{OT}_\beta[2] = r_\beta \cdot \mathsf{OT}_\beta[1] + \mathsf{Enc}_{\text{PK}_\beta}(s_\beta)$ *and* $\mathsf{OT}_{\widetilde{\beta}}[2] = r_{\widetilde{\beta}} \cdot \mathsf{OT}_{\widetilde{\beta}}[1] + \mathsf{Enc}_{\text{PK}_{\widetilde{\beta}}}(s_{\widetilde{\beta}})$ *then responds with* $(\mathsf{OT}_\beta[2], \mathsf{OT}_{\widetilde{\beta}}[2])$. *$P_3$ computes* $v = \mathsf{Dec}_{\text{SK}_\beta}(\mathsf{OT}_\beta[2]) + \mathsf{Dec}_{\text{SK}_{\widetilde{\beta}}}(\mathsf{OT}_{\widetilde{\beta}}[2])$

Validity Conditions for Defenses

- Defense for π_{OT}: The receiver must provide randomness for one of the two parallel OTs, while the sender must provide randomness for both parallel executions

$$\begin{aligned}&\mathsf{valid}(\mathsf{OTRecv}, (\mathsf{trans}, \widetilde{\mathsf{trans}}), \mathsf{def}) := \\ &\quad \big(((\mathrm{PK}, \mathrm{SK}) = \mathsf{Gen}(1^\kappa; r_{\mathsf{Gen}})) \wedge \ (\mathsf{OT}[1] = \mathsf{Enc}_{\mathrm{PK}}(x; r_{\mathsf{Enc}}))\big) \\ &\qquad \vee \ \Big((\mathsf{Gen}(\widetilde{r}_{\mathsf{Gen}}) = (\widetilde{\mathrm{PK}}, \widetilde{\mathrm{SK}})) \ \wedge \ (\widetilde{\mathsf{OT}}[1] = \mathsf{Enc}_{\widetilde{\mathrm{PK}}}(x; \widetilde{r}_{\mathsf{Enc}}))\Big)\end{aligned}$$

where $\mathsf{def} = (x, \mathrm{SK}, r_{\mathsf{Gen}}, r_{\mathsf{Enc}}, \widetilde{\mathrm{SK}}, \widetilde{r}_{\mathsf{Enc}}, \widetilde{r}_{\mathsf{Gen}})$ and $\mathsf{trans} = ((\mathrm{PK}, \mathsf{OT}[1]), \mathsf{OT}[2]), \widetilde{\mathsf{trans}} = ((\widetilde{\mathrm{PK}}, \widetilde{\mathsf{OT}}[1]), \widetilde{\mathsf{OT}}[2])$.

$$\begin{aligned}&\mathsf{valid}(\mathsf{OTSen}, (\mathsf{trans}, \widetilde{\mathsf{trans}}), \mathsf{def}) := \\ &\quad \big(\ (ct = \mathsf{Enc}_{\mathrm{PK}}(s, r_{\mathsf{Enc}})) \wedge \ (\mathsf{OT}[2] = r \cdot \mathsf{OT}[1] + ct)\big) \\ &\qquad \wedge \ \big((\widetilde{ct} = \mathsf{Enc}_{\widetilde{\mathrm{PK}}}(\widetilde{s}, \widetilde{r}_{\mathsf{Enc}})) \wedge \ (\widetilde{\mathsf{OT}}[2] = \widetilde{r} \cdot \widetilde{\mathsf{OT}}[1] + \widetilde{ct})\big)\end{aligned}$$

where $\mathsf{def} = (r, s, ct, r_{\mathsf{Enc}}, \widetilde{r}, \widetilde{s}, \widetilde{ct}, \widetilde{r}_{\mathsf{Enc}})$ and $\mathsf{trans} = ((\mathrm{PK}, \mathsf{OT}[1]), \mathsf{OT}[2]), \widetilde{\mathsf{trans}} = ((\widetilde{\mathrm{PK}}, \mathsf{OT}[1]), \mathsf{OT}[2])$.

- Defense for $\mathsf{3MUL} = (P_1, P_2, P_3)$: Every party participates in two executions of π_{OT} and need to provide defenses corresponding to their role in the protocol for the two instances.

$$\begin{aligned}&\mathsf{valid}(\mathsf{3MUL}_1, \tau, w) := \\ &\quad \big(\mathsf{valid}(\mathsf{OTRecv}, \mathsf{trans}_1, \mathsf{def}_1) \ \vee \ \mathsf{valid}(\mathsf{OTRecv}, \widetilde{\mathsf{trans}}_1, \mathsf{def}_1)\big) \\ &\qquad\qquad \wedge \ \mathsf{valid}(\mathsf{OTSen}, (\mathsf{trans}_2, \widetilde{\mathsf{trans}}_2), \mathsf{def}_2) \\ &\mathsf{valid}(\mathsf{3MUL}_2, \tau, w) := \mathsf{valid}(\mathsf{OTSen}, (\mathsf{trans}_1, \widetilde{\mathsf{trans}}_1), \mathsf{def}_1) \\ &\qquad\qquad \wedge \ \mathsf{valid}(\mathsf{OTSen}, (\mathsf{trans}_2, \widetilde{\mathsf{trans}}_2), \mathsf{def}_2) \\ &\mathsf{valid}(\mathsf{3MUL}_3, \tau, w) := \\ &\quad \big(\mathsf{valid}(\mathsf{OTRecv}, \mathsf{trans}_1, \mathsf{def}_1) \ \vee \ \mathsf{valid}(\mathsf{OTRecv}, \widetilde{\mathsf{trans}}_1, \mathsf{def}_1)\big) \\ &\qquad \wedge \ \big(\mathsf{valid}(\mathsf{OTRecv}, \mathsf{trans}_2, \mathsf{def}_2) \ \vee \ \mathsf{valid}(\mathsf{OTRecv}, \widetilde{\mathsf{trans}}_2, \mathsf{def}_2)\big)\end{aligned}$$

where $w = (\mathsf{def}_1, \mathsf{def}_2)$ and $\tau = ((\mathsf{trans}_1, \widetilde{\mathsf{trans}}_1), (\mathsf{trans}_2, \widetilde{\mathsf{trans}}_2))$.

Figure 5: Defining the validity condition for defenses.

– *Party P_3 computes ciphertexts $\mathsf{OT}_\gamma[1] = \mathsf{Enc}_{\mathrm{PK}_\gamma}(x_3)$ and $\mathsf{OT}_{\tilde{\gamma}}[1] = \mathsf{Enc}_{\mathrm{PK}_{\tilde{\gamma}}}(x_3)$ and sends $(\mathsf{OT}_\gamma[1], \mathsf{OT}_{\tilde{\gamma}}[1])$ to P_1.*

ROUND 3:

– *Party P_1 samples $u_\gamma, u_{\tilde{\gamma}}$ such that $u_\gamma + u_{\tilde{\gamma}} = u$ and $s_\gamma, s_{\tilde{\gamma}}$ such that $s_\gamma + s_{\tilde{\gamma}} = s_1$. It computes $\mathsf{OT}_\gamma[2] = u_\gamma \cdot \mathsf{OT}_\gamma[1] + \mathsf{Enc}_{\mathrm{PK}_\gamma}(s_\gamma)$ and $\mathsf{OT}_{\tilde{\gamma}}[2] = u_{\tilde{\gamma}} \cdot \mathsf{OT}_{\tilde{\gamma}}[1] + \mathsf{Enc}_{\mathrm{PK}_{\tilde{\gamma}}}(s_{\tilde{\gamma}})$ then responds with $(\mathsf{OT}_\gamma[2], \mathsf{OT}_{\tilde{\gamma}}[2])$. P_3 computes $w = \mathsf{Dec}_{\mathrm{SK}_\gamma}(\mathsf{OT}_\gamma[2]) + \mathsf{Dec}_{\mathrm{SK}_{\tilde{\gamma}}}(\mathsf{OT}_{\tilde{\gamma}}[2])$*

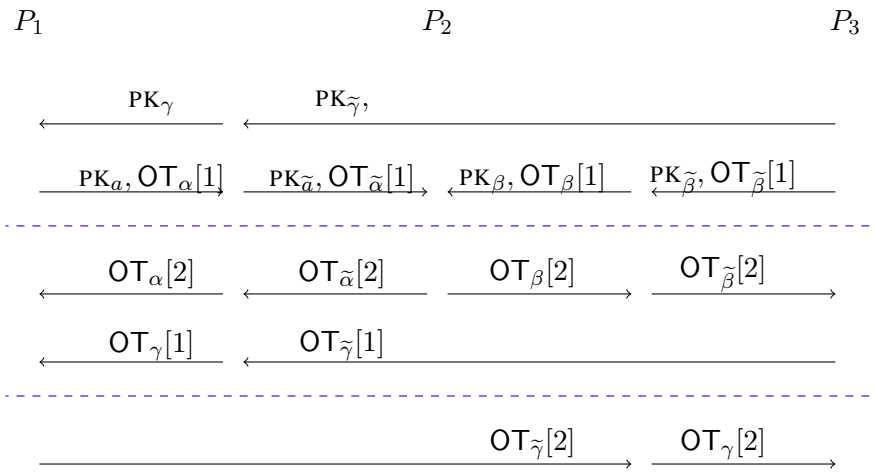

Figure 6: Π_{R3MUL} protocol. Note that in this protocol each OT protocol is executed twice i.e., $OT_\alpha, OT_{\widetilde{\alpha}}$ etc.

5.2 4-Round Actively Secure Protocol for $\mathcal{F}_{\text{ppoly}}$

In this protocol, besides the affine homomorphic encryption scheme (cf. Definition 2.5), we will require the following primitives. (1) A three-round weak one-many non-malleable commitment scheme against synchronizing adversaries (cf. Definition 2.10) where we denote the messages by $\mathsf{nm} = (\mathsf{nm}[1], \mathsf{nm}[2], \mathsf{nm}[3])$. (2) A two-round resettable reusable witness indistinguishable proof (cf. Definition 2.11) for which we rely on ZAPs. We follow the same conventions as we did in Section 4.2 for Π_{dpoly}. In addition, we will recall the notation Exec_j to be the set of indices t corresponding to the terms M_t that P_j participates in. We are now ready to describe our complete protocol.

Protocol 7 (Parallel Polynomial Protocol Π_{ppoly})

Input: *Parties $\overline{P}_1, \ldots, \overline{P}_n$ are given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ each of length κ, respectively.*

- ROUND 1: *Each party $\overline{P}_j$ samples M random secret shares of 0, $\{(z^1_{j,\ell}, \ldots, z^n_{j,\ell})\}_{\ell \in [M]}$ using $\mathsf{Share}(0, n)$ and sends $(z^i_{j,1}, \ldots, z^i_{j,M})$ to party P_i.*
- ROUNDS 1,2,3: *For each monomial $M_t = (x_{t,i}, x_{t,j}, x_{t,k})$, $t \in [q]$, parties $\overline{P}_i, \overline{P}_j, \overline{P}_k$ execute Π_{R3MUL} until the end of the 3^{rd} round. Let $s_{t,i}, s_{t,j}, s_{t,k}$ be the messages parties $\overline{P}_i, \overline{P}_j, \overline{P}_k$ respectively are set to broadcast in the 4^{th}-round as part of the Π_{R3MUL} protocol.*
- ROUNDS 1,2,3: *For every (ordered) pair (i, j), $i, j \in [n]$, $\overline{P}_i$ and $\overline{P}_j$ engage in two non-malleable commitments where $\overline{P}_i$ chooses two random strings w_1, w_2 of appropriate length and commits to them. Denote the transcript of the two interactions by $\mathsf{nm}^0_{i,j}, \mathsf{nm}^1_{i,j}$. In Round 3, $\overline{P}_i$ additionally sends $\tilde{w}_0$ and $\tilde{w}_1$ such that $w_0 + \tilde{w}_0 = w_1 + \tilde{w}_1 = (\boldsymbol{x}_i, \boldsymbol{y}_i)$, where a valid defense $\mathsf{def}_{\mathsf{nm}}$ for one of the two non-malleable commitments and for every $t \in \mathsf{Exec}_i$, a valid defense def_t for Π_{R3MUL} can be obtained from $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ that will serve as the defense for party P_i in the t^{th} Π_{R3MUL} instance.*
- ROUNDS 2,3: *For every (ordered) pair (i, j), $i, j \in [n]$, $\overline{P}_i$ and P_j engage in a ZAP where $\overline{P}_i$ proves to $\overline{P}_j$, the NP relation,*

$$\exists\, w \in \{0,1\}^*, b \in \{0,1\} \text{ such that } \mathsf{valid}(\mathsf{nmcom}, \mathsf{nm}^b_{i,j}, (w, \mathsf{def}_{\mathsf{nm}})) = 1 \text{ and } (\boldsymbol{x}_i, \boldsymbol{y}_i) = w + \widetilde{w}_b$$
$$\text{and } \forall t \in \mathsf{Exec}_i, \mathsf{valid}(\mathsf{3MUL}_{\mathsf{Role}(t,i)}, \tau_t, \mathsf{def}_t) = 1$$

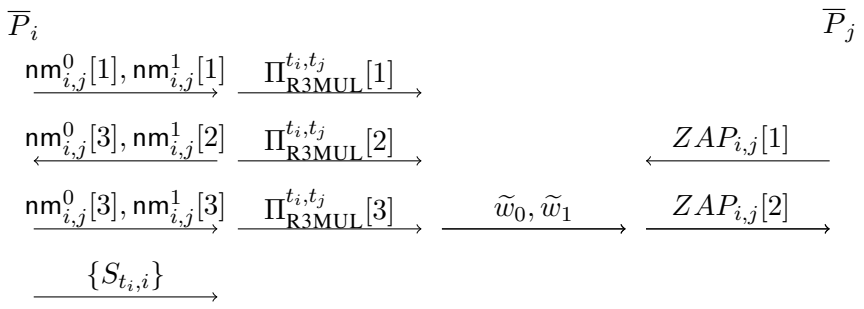

Figure 7: Communication flow from party $\overline{P}_i$ to $\overline{P}_j$ in protocol Π_{ppoly} for $t_i \in \mathsf{Exec}_i, t_j \in \mathsf{Exec}_j$.

where $(\widetilde{w}_0, \widetilde{w}_1)$ is part of Round 3 interaction between $\overline{P}_i$ and $\overline{P}_j$, $\tau = (\tau_1, \ldots, \tau_t)$ is the transcript of the interaction between all parties, τ_t is the interaction for the t^{th} Π_{R3MUL} instance and $\mathsf{def}_{\mathsf{nm}}, \mathsf{def}_t$ can be obtained from $(\boldsymbol{x}_i, \boldsymbol{y}_i)$

- ROUND 4: *In the 4^{th} round, for every $\ell \in \mathsf{Exec}_j$, $\overline{P}_j$ broadcasts $S_{\ell,j} = \sum_{t \in \mathsf{Terms}_\ell} s_{t,j} + \sum_{i=1}^{n} z_i^j$.*
- OUTPUT: *All parties output $Z_1, \ldots, Z_M$ where $Z_\ell = \sum_{j \in [n]} S_{\ell,j}$.*

Proof overview: We briefly describe our simulation and the proof approach. $\mathcal{S}$ samples random inputs $\{\mathbf{x}'_i\}_{i \in \bar{I}}$ for the honest parties and completes the first three rounds of the interactions following the honest strategy. If the adversary aborts before completing the third round, the simulator halts outputting the adversary's view. Otherwise it rewinds to extract a valid defense from the adversary. Upon extracting the defense, $\mathcal{S}$ follows the simulator $\mathcal{S}_{\text{dpoly}}$ strategy of Π_{dpoly} from Section 4.2 and extracts the adversary's input and the errors introduced in the computation from the defense and sends it to $\mathcal{F}_{\text{ppoly}}$. Upon receiving the response from the functionality, it continues the strategy of $\mathcal{S}_{\text{dpoly}}$ and generates the shares that the honest parties need to send in the fourth round which it feeds to the adversary. If the adversary sends its fourth round message, $\mathcal{S}$ follows the strategy $\mathcal{S}_{\text{dpoly}}$ to first extract the errors introduced in the output and then sends it to the ideal functionality and instructs it to deliver the outputs to the honest parties.

The main sequence of hybrids modifies the inputs of honest parties in the different Π_{R3MUL} instances so that they can equivocated to an arbitrary input. In these hybrids, first, we consider a hybrid where we decouple the fourth round message generated by an honest party from its actions in the first three rounds. The goal is that at the end of the hybrids the fourth round message is generated using the original inputs $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$ chosen for the honest parties, but the actions of the honest parties in the first three rounds will be according to randomly chosen inputs $\{\mathbf{x}'_i\}_{\overline{P}_i \in \bar{I}}$ which are randomly chosen. On a high-level the strategy is as follows:

- First, we consider a sequence of hybrids where in each Π_{R3MUL} instance where P_1 is controlled by an honest party, we will modify its action in $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\widetilde{\gamma}}[2]$ so that instead of using u as the input (that it received as output from $\mathsf{OT}_\alpha[2], \mathsf{OT}_{\widetilde{\alpha}}[2]$, we will switch to using a random u'. This will additionally involve switching the ZAP witness to use u' instead of u.
- Next, we consider a sequence of hybrids where in each Π_{R3MUL} instance where we switch P_1's input in $\mathsf{OT}_{\widetilde{\alpha}}[1]$ from x_1 to its value according to $\mathbf{x}'_1$ that we denote by x'_1.
- Following, this we consider a sequence of hybrids we do an anologous change to P_3's input in $\mathsf{OT}_{\widetilde{\beta}}[1]$ and $\mathsf{OT}_{\widetilde{\gamma}}[1]$ when controlled by an honest party from x_3 to x'_3.

- Finally, we generate P_2's messages when controlled by an honest party in such a way that the simulator will possess both witnesses for demonstrating its actions according to x_2 and x_2'.
- Now, we will be in a position to switch the ZAP witness generated using the inputs $\{\mathbf{x}_i\}_{P_i \in \bar{I}}$ to being generated according to $\{\mathbf{x}_i'\}_{P_i \in \bar{I}}$.
- Then we go through the hybrids in reverse where we switch the other witness to correspond to $\mathbf{x}_i'$. Now, the actions of all honest parties in the first three rounds are consistent with the honest strategy on input $\{\mathbf{x}_i'\}_{P_i \in \bar{I}}$.

5.3 4-Round Actively Secure Multi-Party Computation for Arbitrary Functionalities

Our final protocol to realize arbitrary functionalities will on a high-level rely on the protocol from Section 4.3 where instead of relying on Π_{dpoly} to realize $\mathsf{BMR}_{\text{Offline}}$ we will rely on Π_{ppoly} described in the previous section.

We recall first that we rely on the same preprocessing steps using MAC_k, AMD codes and the [GIW16] transformation as in Section 4.3. We repeat it here for completeness:

1. We modify the function f to $f'(x, k_1, \ldots, k_n)$ defined as $(f(x), \mathsf{MAC}_{k_1}(f(x)), \ldots, \mathsf{MAC}_{k_n}(f(x)))$. Let C be the boolean circuit that computes f'.
2. We modify C to obtain C' that takes as input x and computes $\mathsf{Enc}(\mathrm{C}(\mathsf{Dec}(x)))$ where $\mathsf{Enc}, \mathsf{Dec}$ are the encoding and decoding functions for an AMD code.
3. Let $\widehat{\mathrm{C}}'$ be the result of applying the transformtion of [GIW16] on C'.

Description of Protocol. We first formally describe our protocol. We remark that the protocol is identical to protocol 8 with the only exception that we rely on Π_{ppoly} instead of Π_{dpoly} as the sub-protocol to realize $\mathsf{BMR}_{\text{Offline}}$.

Protocol 8 (Protocol Π_f)

INPUT: *Parties $P_1, \ldots, P_n$ are given input $x_1, \ldots, x_n$ each of length κ, respectively, and a circuit $\widehat{\mathrm{C}}'$ as specified above. We fix the notation $[x_i]_j$ as the j^{th} bit of string x_i.*

- ROUNDS 1,2,3: *For each $i \in [M]$, parties $P_1, \ldots, P_n$ execute Π_{ppoly} for the polynomial p_i up until the 3^{rd} round of the protocol with random inputs for the $\mathcal{F}_{\text{ppoly}} = \mathsf{BMR}_{\text{Offline}}$. Along with the message transmitted in the 3^{rd} round of Π_{ppoly}, party P_j broadcasts the following:*
  - *For every input wire $w \in W$ that carries some input bit $[x_j]_k$ from P_j's input, P_j broadcasts $\Lambda_w = \lambda_w \oplus [x_j]_k$.*

  For every $j \in [n]$, let $\{S_{\ell,j}\}_{\ell \in M}$ be the output of party P_j for the M polynomials. It reassembles the output shares to obtain $\widetilde{S}^{g,j}_{r_1,r_2}$ for every garbled row.
- ROUND 4: *Finally for every gate $g \in G$ and $r_1, r_2 \in \{0,1\}$, P_j ($j \in [n]$) broadcasts the following:*
  - $\widetilde{R}^{g,i}_{r_1,r_2} = \mathsf{F}_{k^j_{a,r_1}}(g, j, r_1, r_2) \oplus \mathsf{F}_{k^j_{b,r_2}}(g, i, r_1, r_2) \oplus \widetilde{S}^{g,i}_{r_1,r_2}$ *for every $i \in [n]$.*
  - k^j_{w,Λ_w} *for every input wire w.*
  - λ^j_w *for every output wire w.*
  - $(\Gamma^j_{w,0}, \Gamma^j_{w,1}) = (h(T^j_{w,0}) \oplus k^j_{w,0}, h(T^j_{w,1}) \oplus k^j_{w,1})$ *for every wire w.*

- OUTPUT: *Upon collecting* $\{\widetilde{R}^{g,j}_{r_1r_2}\}_{j\in[n],g\in[G],r_1,r_2\in\{0,1\}}$, *the parties compute each garbled row by* $R^{g,j}_{r_1,r_2} = \bigoplus_{j=1}^{n} \widetilde{R}^{g,j}_{r_1,r_2}$. *Then using the keys corresponding to the input wires* w, *they evaluate the circuit to obtain* Λ_w *for every output wire as follows:*

  Consider a standard (arbitrary) topological ordering of the gates. The parties will evaluate the circuit according to the topological order. Let g *be a gate in this order with input wires* a, b *and output wire* c. *If a party does not have masks* Λ_a, Λ_b *or keys* (k_a, k_b) *corresponding to the input wires when processing gate* g *it aborts. Otherwise, it will compute*

$$T^j_c = R^{g,j}_{r_1,r_2} \oplus \bigoplus_{i=1}^{n} \left(\mathsf{F}_{k^i_{a,\Lambda_a}}(g, j, \Lambda_a, \Lambda_b) \oplus \mathsf{F}_{k^i_{b,\Lambda_b}}(g, j, \Lambda_a, \Lambda_b)\right)$$

  where $k_a = (k^1_a, \ldots, k^n_a)$ *and* $k_b = (k^1_b, \ldots, k^n_b)$. *Party* P_j *identifies* Λ_c *such that* $T^j_c = T^j_{c,\Lambda_c}$. *If no such* Λ_c *exists the party aborts. Otherwise, each party defines* $k^i_c = \Gamma^i_{c,\Lambda_c} \oplus h(T^j_c)$. *Let* $k_c = (k^1_c, \ldots, k^n_c)$. *The evaluation is completed when all gates in the topological order are processed. Finally given* Λ_w *for every output wire, the parties compute the output carried in wire* w *as* $\Lambda_w \oplus \left(\bigoplus_{j=1}^{n} \lambda^j_w\right)$ *and decode the outcome using* Dec.

This concludes the description of our protocol. Now, we have the following theorem:

Theorem 5.2 *Let* C *be an* n*-party circuit that computes* f *and assume the existence of a 2-round OT* Π_{OT} *(as specified in Section 2.4). Then Protocol* Π_f *securely computes* f *in the presence of an active adversary that corrupts at most* $n-1$ *parties.*

Proof Sketch: Correctness follows from the description of the protocol and essentially the same as in Section 4.3. We describe the simulation next. We remark that the simulation on a high-level follows the simulation presented in Section 4.3 with the exception where the simulator in Section 4.3 relies on a defense provided by the adversary, the simulator here will obtain a defense by rewinding the non-malleable commitments provided by the corrupted parties. Now, we proceed to formal description of the simulator.

Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $I \subset [n]$, then our simulator $\mathcal{S}$ is defined below.

- Recall that the parties engage in an instance of Π_{ppoly} to realized the $\mathsf{BMR}_{\mathrm{Offline}}$ functionality in the first three rounds. The simulator samples random inputs for honest parties and generates the messages of the honest parties honestly using the random inputs. For every input wire that is associated with an honest party's input, the simulator chooses a random Λ_w and sends these bits to the adversary as part of the 3^{rd} message.

- At this point, in our simulation in Section 4.3 we relied on the defense provided by the adversary. The simulator for the current protocol will obtain the defense by extracting the messages from the non-malleable commitment provided by the corrupted parties. More precisely, if the adversary fails to complete the third round for all parties by providing valid proofs using the ZAPs the simulator halts outputting $\perp$. Otherwise, it will rewind the parties from third to the second round repeatedly until it obtains a valid defense. If it fails to obtain a valid defense, the simulator halts outputting $\perp$. Otherwise, from this defense, the simulator obtains λ^j_w and $k^j_{w,0}, k^j_{w,0} \oplus k^j_{w,1}$ for every corrupted party P_j and wire w that is an output of some NAND gate, as well as the PRF values. Finally, it obtains the error $e^{g,j}_{r_1,r_2}$ for every gate g, $r_1, r_2 \in \{0,1\}$ and $j \in I$, where $e^{g,j}_{r_1,r_2}$ is a vector of errors introduced by the adversary for the plaintext encrypted in row (r_1, r_2) in the garbling of gate g.[9]

[9]The errors are bits and are extracted for each monomial where the corrupted party plays the role of P_1. For simplicity, we can collect all the errors for the polynomial representing each bit in the garbled row and then express it as a vector corresponding to each key.

- Next, the simulator chooses a random $\Lambda_w \leftarrow \{0,1\}$ for every internal wire $w \in W$ that is an output of some NAND gate. It further samples a single key k_w^j every honest party $P_j \in \bar{I}$ and wire $w \in W$.

- Upon receiving the adversary's public values for its input bits, the simulator extracts the adversary's input. Namely, for each input wire $w \in W$ that is associated with a corrupted party's input, the simulator computes $\rho_w = \Lambda_w \oplus \lambda_w$ and the errors in the input wires and fixes the adversary's input $\{\mathbf{x}_I\}$ to be the concatenation of these bits incorporating the errors. $\mathcal{S}$ sends $\mathsf{Dec}(\mathbf{x}_I)$ to the trusted party computing f, receiving the output $\tilde{y}$. $\mathcal{S}$ fixes $y = \mathsf{Enc}(\tilde{y})$ where recall $\mathsf{Enc}, \mathsf{Dec}$ are the encoding and decoding functions corresponding to an AMD code. Let $y = (y_1, \ldots, y_m)$.

- For every gate g, given Λ_a, Λ_b, we define $\widetilde{\Lambda}_c$, where a, b are the input wires and c is the output wire of g. Pick a random honest party P_j. If majority of the bits of $e^{g,j}_{r_1,r_2} = 1$ then we set $\widetilde{\Lambda}_c = 1 \oplus \Lambda_c$, and otherwise set $\widetilde{\Lambda}_c = \Lambda_c$. To simulate the output of the garbled row given the errors, we sample T^j_{c,Λ_c} at random and construct T^j_c such that it is equal to the bits of T^j_{c,Λ_c} where $e^{g,j}_{r_1,r_2} = 0$ and at random if $e^{g,j}_{r_1,r_2} = 1$. For every gate g, and honest party P_j, $\mathcal{S}$ constructs $Y^{g,j}_{\Lambda_a,\Lambda_b} = (e^{g,j}_{\Lambda_a,\Lambda_b} \oplus T^j_c)$. For corrupted parties P_j, $\mathcal{S}$ sets $Y^{g,j}_{\Lambda_a,\Lambda_b} = (e^{g,j}_{\Lambda_a,\Lambda_b} \oplus T^j_{c,\widetilde{\Lambda}_c})$. For the t^{th} output wire w, $\mathcal{S}$ defines $(\bigoplus_{i=1}^n \lambda_w^i) = \Lambda_w \oplus y_t$.

  Denote the simulated garbled circuit by $\mathrm{GC}_{\mathcal{S}}$. On behalf of every honest party P_j, $\mathcal{S}$ broadcasts $(r, h(T^j_{w,\Lambda_w}) \oplus k^j_w)$ if $\Lambda_w = 1$ and $(h(T^j_{w,\Lambda_w}) \oplus k^j_w, r)$ if $\Lambda_w = 0$ where r is sampled randomly.

- Next, the simulator provides the fourth message on behalf of the honest parties to the adversary. Namely, for every active row, i.e. for every gate g, the row Λ_a, Λ_b, the shares of the honest parties are computed assuming the output of the polynomials defined in $\mathsf{BMR}_{\mathrm{Offline}}$ are $Y^{g,i}_{\Lambda_a,\Lambda_b}$ for every i masked with the PRF under the keys k^j_a, k^j_b as $\tilde{R}^{g,j}_{\Lambda_a,\Lambda_b}$. For the remaining three rows the simulator sends random strings.

- If the adversary provides its fourth message $\{\widetilde{R}^{g,j}_{r_1 r_2}\}_{j\in[n],g\in[G],r_1,r_2\in\{0,1\}}$, the simulator reconstructs the garbling $\mathrm{GC}_{\mathcal{A}}$ and evaluates it on behalf of the honest parties. If one of the following events does not occur then the simulator sends $\perp$ to the trusted party computing f. First, the simulator checks that the output key of every key obtained during the evaluation is the active key k^j_{c,Λ_c} encrypted by the simulator. In addition, the simulator checks that the outcome of $\mathrm{GC}_{\mathcal{A}}$ is y. Otherwise, the simulator sends an OK message to the trusted party to deliver $\tilde{y}$ to the honest parties.

Arguing indistinguishability of simulation follows a sequence of intermediate hybrids where first we rely on the same hybrids as in the previous section (formally specified in Section 6) where we change the inputs supplied by the honest parties for Π_{ppoly} to randomly sampled inputs. Then we rely on the intermediate hybrids identically as in Section 4.3 where we change the garbled circuit computed from the real garbling to the fake garbling computed only using the output provided by the functionality.

6 Proof of Theorem 5.1

Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $I \subset [n]$, then we prove that there exists PPT simulator $\mathcal{S}$ with access to an ideal functionality $\mathcal{F}$ that implements f, that simulates the adversary's view, namely:

$$\{\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa,\cdot)\}_{\kappa} \overset{c}{\approx} \{\mathbf{REAL}_{\Pi,\mathcal{A}}(\kappa,\cdot)\}_{\kappa}$$

Denoting the set of honest parties by $\bar{I}$, our simulator $\mathcal{S}$ is defined below.

The description of the simulation.

- $\mathcal{S}$ internally incorporates $\mathcal{A}$ and proceeds as follows. It samples random inputs for the honest parties and completes the first three rounds of the interactions following the honest strategy. If the adversary aborts before completing the third round, the simulator halts outputting the adversary's view. Otherwise it proceeds.
- Then it stalls the main thread and proceeds to extract the defense from the adversary by rewinding the adversary from the third to the second round. In particular, it uses the extractor for the non-malleable commitment scheme and tries to extract the value committed to by every corrupted to party $\overline{P}_i$ to some honest party $\overline{P}_j$. If the extraction fails it halts outputting $\perp$ and otherwise proceeds.
- Upon extracting the defense, the simulation $\mathcal{S}$ follows the simulator $\mathcal{S}_{\text{ppoly}}$ strategy of Π_{ppoly} from the Section 4.2 and extracts the adversary's input and the errors introduced in the computation from the defense and sends it to $\mathcal{F}_{\text{ppoly}}$. Upon receiving the response from the functionality, it continues the strategy of $\mathcal{S}_{\text{ppoly}}$ and generates the shares that the honest parties need to send in the fourth round which it feeds to the adversary.
- If the adversary $\mathcal{A}$ sends its fourth round message, it again follows the strategy $\mathcal{S}_{\text{ppoly}}$ to first extract the errors introduced in the output and then sends it to the ideal functionality and instructs it to deliver the outputs to the honest parties.

We now proceed to proving formally.

Lemma 6.1 *The following distributions are indistinguishable:*

- $\left\{\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)\right\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$, *and*
- $\left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)\right\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$.

Proof: Let $\mathcal{P} = \{\overline{P}_1, \ldots, \overline{P}_n\}$ be the set of parties, let $\mathcal{A}$ be a malicious, static adversary in the plain model, and let $I \subseteq \mathcal{P}$ be the set of parties corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ (the ideal world adversary) with access to the ideal functionality $\mathcal{F}_{\text{dpoly}}$, such that the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}$ is indistinguishable from a real execution of Π_{ppoly} with $\mathcal{A}$.

Assume for contradiction, there exists an adversary $\mathcal{A}$, distinguisher D and polynomial $p(\cdot)$ such that the probability with which D distinguishers $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)$ and $\mathbf{REAL}_{\Pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)$ for infinitely many κ is $\frac{1}{p(\kappa)}$. Fix an n and inputs for parties for which this happens.

We design a sequence of intermediate hybrid experiments starting from the real world leading to the ideal world and argue correctness via a standard hybrid argument. More precisely, we design $q(n)$ hybrids below and there must be a mapping $i(\kappa)$ such that D distinguishes the outputs of i^{th} and $(i+1)^{st}$ intermediate experiments with probability at least $\frac{1}{p(\kappa)q(\kappa)}$. In experiment H_i below, we denote the output of the experiment by $\mathbf{Hybrid}_i(\kappa, z, x_1, \ldots, x_n)$.

Hybrid H_0: This experiment proceeds identically to the real execution. More precisely, in H_0 consider a simulator $\mathcal{S}_0$ that has all the honest parties real inputs and starts an execution with $\mathcal{A}$ providing it fresh randomness and input $\{x_j^*\}_{\overline{P}_j\in I}$ and emulating the actions of the honest parties using the real inputs. The output of the experiment is $\mathbf{REAL}_{\Pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)$ which consists of $\mathcal{A}$'s view and output of honest parties.

Hybrid H_1: This experiment proceeds identically to H_0 with the following exception: the simulator will try to extract the adversary's defenses $\{\mathbf{x}_i^*, \mathbf{y}_i^*\}_{i\in[I]}$ by rewinding the non-malleable commitment. In more detail, the simulator $\mathcal{S}_1$ proceeds as follows:

- It completes the first three rounds exactly as in H_0. If $\mathcal{A}$ aborts before delivering the third message for some corrupted party, then the simulator halts. Otherwise it proceeds to extraction.
- The simulator will extract the inputs and defenses from corrupted parties by rewinding the non-malleable commitment made by corrupted parties (to honest parties). Recall that the non-malleable commitment is executed from the first round and completes in the third round.

  In more detail, $\mathcal{S}_1$ constructs a committer for nmcom, C^* that internally incorporates $\mathcal{A}$ and simulates all messages for $\mathcal{A}$, except those corresponding to (each execution of) nmcom where the adversary controls the committer, which it forwards to an external party. Treating each commitment made by the corrupted party as an honest commitment, using the weak one-many non-malleability property, the simulator rewinds from third to second round to extract the message in the commitment. We are able to apply the weak one-many non-malleability property since the protocal demands a ZAP proof from the committer ensuring that one of the two commitments is well-formed. Recall that between every pair of parties $\overline{P}_i$ and $\overline{P}_j$, two non-malleable commitments are made by $\overline{P}_i$ to $\overline{P}_j$. For every corrupted party $\overline{P}_i$, the simulator chooses an honest party $\overline{P}_j$ and tries to extract one of the two commitments made by $\overline{P}_i$ to $\overline{P}_j$ in parallel until one of them succeeds (If it runs too long, say $2^{\kappa/2}$ time steps, it aborts). If extraction succeeds, let nm^b be the well-formed commitment, upon receiving the third round message $\{\widetilde{w}_{b,i}\}_{i\in[I]}$ and leveraging the extracted values $\{w_{b,i}\}_{i\in[I]}$ $\mathcal{S}_1$ defines $\{\mathbf{x}_i^*, \mathbf{y}_i^*\}_{i\in I}$ as follows:

  $$(\mathbf{x}_i^*, \mathbf{y}_i^*) = w_b + \widetilde{w}_b$$

  If two valid commitments were obtained, the simulator tries to obtain a defense from both messages and chooses the valid one (if one exists and at random if both are valid). Using the $(\mathbf{x}_i^*, \mathbf{y}_i^*)$ extracted from the valid message, $\mathcal{S}_1$ obtains $\mathsf{def}_{\mathsf{nm}}, \mathsf{def}_f$.
- H_1 now completes the final round as in H_0.

It follows from the proceeding argument that the outputs of H_0 and H_1 are identically distributed conditioned on the extraction procedure not failing. From the soundness of the ZAP, we know that except with negligible probability, at least one of the two non-malleable commitments will be well-formed and therefore the extractor will succeed in expected polynomial time. Furthermore, the soundness of ZAP will also ensure that the extracted defense is valid. This implies that $\mathcal{S}_1$ aborts only with negligible probability which in turn means that the outputs of H_0 and H_1 are statistically close. Therefore, we have the following claim:

Claim 6.2 *The following distributions are statistically close:*

- $\{\mathbf{Hybrid}_0(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$,
- $\{\mathbf{Hybrid}_1(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$.

It suffices to argue that $\mathcal{S}_1$ runs in expected polynomial time. Let p be the probability with which $\mathcal{A}$ completes in the third round. The number of rewinding executions is bounded by $p \cdot \frac{1}{p}$ and the number

of times before another non-aborting rewinding happens is $1/p$. That said, the expected number of rewinding executions till another non-aborting rewinding happens is constant. The extractor for nmcom takes expected time $\mathsf{poly}(\kappa)/p$ and succeeds with probability $1 - \mu(\kappa)$.

Hybrid H_2: This experiment proceeds identically to the previous experiment with the exception of how the fourth round messages of the honest parties are generated. In H_1, the fourth round messages were computed following the honest strategy. In H_2 we will rely on the defense extracted from the adversary and the real inputs for the honest parties. In more detail, the simulation will follow the simulator strategy $\mathcal{S}_{\text{dpoly}}$ of Π_{dpoly} from the Section 4.2 and extracts the adversary's input, output shares and the errors introduced in the computation from the defense. Using the defense of the adversary and the inputs and randomness of the honest parties for Π_{ppoly}, $\mathcal{S}$ will generate the fourth round messages of the honest parties. The output of the honest parties in this hybrid and previous hybrid will be statistiaclly close and this follows analogously as in Section 4.2. This hybrid is introduced so that the fourth round messages from all honest parties are generated using only the inputs and randomnss chosen for the honest parties and the adversary's defense. In particular, the intermediate outputs part of the protocol will not be used. We will generate the fourth round message this way in all the remaining hybrids until the last one. Therefore, we have the following claim.

Claim 6.3 *The following distributions are statistically close.*

- $\{\mathbf{Hybrid}_1(\kappa, z, x_1, \ldots, x_n)\}_{\kappa \in \mathbb{N}, z, x_1, \ldots, x_n \in \{0,1\}^*}$,
- $\{\mathbf{Hybrid}_2(\kappa, z, x_1, \ldots, x_n)\}_{\kappa \in \mathbb{N}, z, x_1, \ldots, x_n \in \{0,1\}^*}$.

In the next sequence of hybrids, we will modify the inputs of honest parties in the different Π_{R3MUL} instances. In these hybrids, we continue to generate the fourth round message using the original inputs $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$ chosen for the honest parties. At the end of the next set of hybrids, the simulator will have modified the inputs to $\{\mathbf{x}'_i\}_{\overline{P}_i \in \bar{I}}$ which are randomly chosen. On a high-level the strategy is as follows:

- First, we consider a sequence of hybrids where in each Π_{R3MUL} instance where P_1 is controlled by an honest party, we will modify its action in $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\widetilde{\gamma}}[2]$ so that instead of using u as the input (that it received as output from $\mathsf{OT}_\alpha[2], \mathsf{OT}_{\widetilde{\alpha}}[2]$, we will switch to using a random u'. This will additionally involve switching the ZAP witness to use u' instead of u.
- Next, we consider a sequence of hybrids where in each Π_{R3MUL} instance where we switch P_1's input in $\mathsf{OT}_{\widetilde{\alpha}}[1]$ from x_1 to its value according to $\mathbf{x}'_1$ that we denote by x'_1.
- Following, this we consider a sequence of hybrids we do an anologous change to P_3's input in $\mathsf{OT}_{\widetilde{\beta}}[1]$ and $\mathsf{OT}_{\widetilde{\gamma}}[1]$ when controlled by an honest party from x_3 to x'_3.
- Finally, we generate P_2's messages when controlled by an honest party in such a way that the simulator will possess both witnesses for demonstrating its actions according to x_2 and x'_2.
- Now, we will be in a position to switch the ZAP witness generated using the inputs $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$ to being generated according to $\{\mathbf{x}'_i\}_{\overline{P}_i \in \bar{I}}$.
- Then we go through the hybrids in reverse where we switch the other witness to correspond to $\mathbf{x}'_i$. Now, the actions of all honest parties in the first three rounds are consistent with the honest strategy on input $\{\mathbf{x}'_i\}_{P_i \in \bar{I}}$.

For every Π_{R3MUL} instance where the party controlling P_1 is honest, we consider the following sequence of intermediate experiments where we replace the inputs of the honest parties from the chosen one to a random input. We continue to generate the fourth round message using the original inputs chosen for the honest parties, namely $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$. We will do this in sequence, where we first replace all the inputs for party P_1 in the individual Π_{R3MUL} instances, when an uncorrupted party controls P_1. Then we move to P_2 and P_3.

We remark that the simulation in the next sequence of hybrids will be slightly different. The simulation after generating the first message of the protocol will stall the main thread and proceed to rewinding. In other words, the simulator proceeds to rewinding whether or not it aborts in the main thread. It might worrisome that this could affect the output distribution as we need to simulate the abort probability correctly. However, we will use the fact that towards arriving at a contradiction, we can restrict ourselves to distinguishers that distinguish $H_3^{2,j}$ from $H_3^{1,j}$ with probability at least $\frac{1}{q(\kappa)p(\kappa)}$. Therefore, it suffices for us to simulate the hybrids except with probability $\frac{1}{4q(\kappa)p(\kappa)}$. Then we can conclude using an union bound over the failure of extraction that the distinguisher can still distinguish with probability at least $\frac{1}{2q(\kappa)p(\kappa)}$ between every two successive hybrids. We call this strategy *premature rewinding*. In slight more detail, suppose that the adversary does not abort with probability p. Then we have two cases depending on whether p is bigger than $\frac{1}{4q(\kappa)p(\kappa)}$ or not. If $p < \frac{1}{4q(\kappa)p(\kappa)}$, then simply outputting aborting transcripts simulates the hybrid with the required probability. If $p > \frac{1}{4q(\kappa)p(\kappa)}$, then we will make sure that the extractor will succeed with high probability. In other words, our simulator will rewind a fixed polynomial number of times namely some polyonmial in $4q(\kappa)p(\kappa)$ to guarantee high success probability whenever $p > \frac{1}{4q(\kappa)p(\kappa)}$. Then if the simulator fails, it proceeds to the main thread and completes the execution until either the adversary aborts or it reaches the third round at which point the simulator halts outputting $\perp$. Now we proceed to these intermediate hybrid experiments.

For each honest party $\overline{P}_i$ in $\bar{I}$ and every monomial M_t such that $\mathsf{Role}(t, i) = 1$, consider the following intermediate experiments.

Hybrid $H_3^{1,j}$: This experiment proceeds identically to H_2 with the exception that the non-malleable commitment $\mathsf{nm}_{i,j}^1$ is simulated differently. First the value committed by the honest party playing P_1, say $\overline{P}_i$ to any (corrupted) party $\overline{P}_j$, in $\mathsf{nm}_{i,j}^1$ is switched from w_1 to a random value R. (in particular the message sent in the third round $\widetilde{w}_1$ will not satisfy that $R + \widetilde{w_1}$ is a valid defense). We will perform a premature rewinding to obtain a valid defense. We remark that this defense will be used to compute a second witness for the honest party but not used to generate the fourth round message. In more detail, consider a simulator $\mathcal{S}_3^{1,j}$ that proceeds as follows. After the first message is generated in the main thread, the simulator stalls the main thread and proceeds to rewinding. As mentioned above in the premature rewinding we will invoke the extractor to rewind and guarantee correct simulation with probabilty at least $1 - \frac{1}{4q(\kappa)p(\kappa)}$. In the rewinding, we will generate the third message of $\mathsf{nm}_{i,j}^1$ internally by assuming that the value committed in the first message is according to w_1. We remark that the success probability of the rewinding will not be affected (with more than negligible probability) whether the first message was generated according to w_1 or R because otherwise the hiding of the commitment in the first round would be violated.

After the premature extraction, we proceed to the main thread and complete the execution until the third round. Now, we can use the non-malleability reduction to extract the message committed to by the adversary, which in turn, is used to extract another defense and complete the fourth message as in the previous hybrid.

Let the output of the experiment be denoted as $\mathbf{Hybrid}_3^{1,j}(\kappa, z, x_1, \ldots, x_n)$. The indistinguishability

of the experiments $H_3^{1,j}$ and H_2 follows directly from the weak one-many non-malleability against synchronizing adversaries of the underlying non-malleable commitment scheme. The premature extraction strategy can cause the simulation to fail with probability at most $1/4p(\kappa)q(\kappa)$. Applying a union bound, we still have a distinguishing probability of $1/2p(\kappa)q(\kappa)$.

Claim 6.4 *The following distributions are indistinguishable.*

- $\{\mathbf{Hybrid}_2(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$,
- $\{\mathbf{Hybrid}_3^{1,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$.

Hybrid $H_3^{2,j}$: This experiment proceeds identically to $H_3^{1,j}$ with the exception that the $\tilde{w}_1$ generated in the third round of the main thread will be such that w_1 (that is in the simulator's head) and $\tilde{w}_1$ add up to a second defense for the oblivious transfer messages $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\tilde{\gamma}}[2]$ involving u and s_1 (see Protocol 6). First, we remark that a second defense cannot be obtained without rewinding. Since we are performing premature rewinding we will be able to extract a defense that will be sufficient to obtain the second defense.

Given the extracted defense, first a second witness corresponding to u, s_1 is obtained. Given that u, s_1 are the inputs used by P_1 in the main execution, sample random u' and compute $s_1' = (u - u')x_3 + s_1$. Now, given any second message, we can generate $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\tilde{\gamma}}[2]$ using the defense provided for P_3 (namely the public and secret keys for the encryption schemes) such that valid defense with both u, s_1 and u', s_1' can be established (i.e. obtain randomness that shown both $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\tilde{\gamma}}[2]$ could have been obtained from either pair of inputs). This is possible using the equivocation property of the underlying encryption scheme.

After the premature extraction and obtaining the second defense, the simulator $\mathcal{S}_3$ will resume the main thread and complete the execution by setting $\tilde{w}_1$ such that $w_1 + \tilde{w}_1$ has the second defense. Indistinguishability follows from the fact that since w_1 has been decoupled from the commitment made in $\mathsf{nm}_{i,j}^1$ (that is made to a random R), the two distributions will be identically distributed (conditioned on extraction). We will rely again on the non-malleability reduction for obtaining the defense in the main thread. Indistinguishability of the main thread follows again using weak one-one non-malleability. Therefore, we have the following claim.

Claim 6.5 *The following distributions are statistically close.*

- $\{\mathbf{Hybrid}_3^{1,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$,
- $\{\mathbf{Hybrid}_3^{2,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N}, z, x_1, \ldots, x_n\in\{0,1\}^*}$.

Hybrid $H_3^{3,j}$: In this experiment the simulator proceeds identically as in the previous hybrid, with the exception that it reverts the change made in $H_3^{1,j}$, namely, $\mathcal{S}_3^{3,j}$ will follow the honest strategy by committing to the value w_1 (instead of random R). Indistinguishability follows from the weak non-malleability property of the commitment scheme. Recall that premature rewinding will provide the second witness. In the main thread, the adversary will receives a commitment to R in the previous hybrid and w_1 in the current hybrid.

We arrive at a contradiction by relying on the weak non-malleability property of the underlying non-malleability commitment scheme. Therefore, we have the following claim.

Claim 6.6 *The following distributions are computationally indistinguishable:*

- $\{\mathbf{Hybrid}_3^{2,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$,
- $\{\mathbf{Hybrid}_3^{3,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$.

Hybrid $H_3^{4,j}$: This experiment proceeds identically to $H_3^{3,j}$ with the exception that the witness used in the ZAP proof is changed from the witness with u to the witness with u'. Indistinguishability will follow directly from the resettable reusable witness indistinguishabilty of the ZAP. Just as in the previous two hybrids, we will rely on premature rewinding to extract the second defense. Therefore, we have the following claim:

Claim 6.7 *The following distributions are computationally indistinguishable:*

- $\{\mathbf{Hybrid}_3^{3,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$,
- $\{\mathbf{Hybrid}_3^{4,j}(\kappa, z, x_1, \ldots, x_n)\}_{\kappa\in\mathbb{N},z,x_1,\ldots,x_n\in\{0,1\}^*}$.

We are now at a hybrid where the witness used in the ZAP contains a defense for u' which is a random value, rather than the actual value u obtained from $\mathsf{OT}_\alpha[2], \mathsf{OT}_{\tilde{\alpha}}[2]$.

Hybrid $H_3^{5,j} - H_3^{7,j}$: We consider a set of experiments analogous to $H_3^{1,j}$-$H_3^{3,j}$, where we will switch the first non-malleable commitment $\mathsf{nm}_{i,j}^0$ and $\tilde{w}_0$ such that $w_0 + \tilde{w}_0$ is equal to the second witness.

At this point we have that both $w_0 + \tilde{w}_0 = w_1 + \tilde{w}_1$ provide a defense for P_1's actions in $\mathsf{OT}_\gamma[2]$ and $\mathsf{OT}_{\tilde{\gamma}}[2]$ according to the randomly chosen value u'.

Hybrid$H_3^{8,j}$: This experiment is identical to $H_3^{7,j}$ with the exception that the witness used in the ZAP proof is changed from using $\mathsf{nm}_{i,j}^1$ to $\mathsf{nm}_{i,j}^0$. Indistinguishability will follow directly from the resettable reusable witness indistinguishabilty of the ZAP as in $H_3^{4,j}$.

In the next set of H_4 hybrids we remove the input of honest party $\overline{P}_i$ from the oblivious transfer. In particular:

Hybrid $H_4^{1,j}$: This experiment is identical to hybrid $H_3^{8,j}$ with the exception that the non-malleable commitment $\mathsf{nm}_{i,j}^1$ is simulated differently. First the value committed by the honest party in $\mathsf{nm}_{i,j}^1$ is switched from w_1 to a random value R. We will perform premature rewinding where internally we use the value w_1 to simulate the third message of $\mathsf{nm}_{i,j}^1$. We run the extractor for the non-malleable commitment to obtain a valid defense from the adversary to simulate the fourth round. Indistinguishability follows directly from the weak non-malleability property of the commitment scheme.

Hybrid $H_4^{2,j}$: Identical to $H_4^{1,j}$ except that we remove the inputs on behalf of the honest party $\overline{P}_i$ (acting as P_1) from the first duplicate oblivious transfer message $OT_{\tilde{\alpha}}[1]$ from x_1 to a x_1'. Indistdinghuisability follows due to privacy against defensible receivers. Note that the sequence of H_3 hybrids were needed to address the "u" problem. Namely, the effect of the current hybrid will lead to an incorrect (unknown) value for u, and by changing the correct u to a random u' in the previous hybrids we avoided this problem.

Hybrid $H_4^{3,j}$: This experiment proceeds identically to $H_4^{2,j}$ with the exception that $\tilde{w}_1$ generated in the third round will be such that $w_1 + \tilde{w}_1$ (where w_1 is in the simulator's head) equal to the second defense for the oblivious transfer messages $\mathsf{OT}_{\alpha}[1]$ and $\mathsf{OT}_{\widetilde{\alpha}}[1]$, namely x_1' used in $\mathsf{OT}_{\widetilde{\alpha}}[1]$. Indistinguishability follows essentially as in hybrid $H_3^{2,j}$.

Hybrid $H_4^{4,j}$: This experiment is identical to hybrid $H_4^{3,j}$ with the exception that it reverts the change made in $H_4^{1,j}$, namely, $\mathcal{S}_4^{4,j}$ will follow the honest strategy by committing to the value w_1 in $\mathsf{nm}_{i,j}^1$ (instead of R). Indistinguishability follows as in hybrid $H_4^{1,j}$.

At this point for every multiplication instance where P_1 is controlled by an honest party, the input is switched from x_1, the original chosen input, to a random input x_1' as in the simulation.

Next, we consider the scenario where P_2 is controlled by an honest party. The sequence of hybrids involved in replacing its input from x_2 to x_2' will be analogous to the sequence $H_3^{1,j}$ to $H_3^{3,j}$ where we set up $\mathsf{nm}_{i,j}^1$ and $\tilde{w}_1$ such that $w_1 + \tilde{w}_1$ contains a defense for x_2'.

Next, we consider the scenario where P_3 is controlled by an honest party. The sequence hybrids involved in replacing its input from x_3 to x_3' will be analogous to the sequence $H_4^{1,j}$ to $H_4^{4,j}$.

At this point, we have that $w_1 + \tilde{w}_1$ is a defense according to $\{\mathbf{x}_i'\}_{\overline{P}_i \in \bar{I}}$ and $w_0 + \tilde{w}_0$ is a defense according to $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$. Now we consider a hybrid where we switch the witness in the ZAP statement from a defense according to $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$ to a defense according to $\{\mathbf{x}_i'\}_{\overline{P}_i \in \bar{I}}$. Indistinguishability will rely on the reusable resetting WI property of the ZAP.

Next, we consider a sequence of hybrids in reverse order so that $w_0 + \tilde{w}_0$ is also a defense for $\{\mathbf{x}_i'\}_{\overline{P}_i \in \bar{I}}$ followed by switching the ZAP witness using $w_0, \tilde{w}_0$. This completes our hybrids as we have switched the inputs and actions of the honest party in the first three rounds from according to $\{\mathbf{x}_i\}_{\overline{P}_i \in \bar{I}}$ to $\{\mathbf{x}_i'\}_{\overline{P}_i \in \bar{I}}$ and this is our simulation strategy. Therefore, this completes our proof. □

References

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO*, pages 468–499, 2017.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001.

[BGJ+17] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Dakshita Khurana, and Amit Sahai. Round optimal concurrent MPC via strong simulation. *To Appear TCC*, 2017.

[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.

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. *To Appear TCC*, 2017.

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

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

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

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *EUROCRYPT*, pages 471–488, 2008.

[COSVa] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In *TCC 2017*.

[COSVb] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In *TCC 2017*.

[COSV16] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. In *CRYPTO*, pages 270–299, 2016.

[COSV17] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Four-round concurrent non-malleable commitments from one-way functions. In *CRYPTO*, pages 127–157, 2017.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

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

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[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.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In *CRYPTO*, pages 721–741, 2015.

[GIW16] Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary amd circuits from secure multiparty computation. In *TCC*, pages 336–366, 2016.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60, 2012.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT*, pages 448–476, 2016.

[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.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 695–704. ACM Press, June 2011.

[GPR16] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In *STOC*, pages 1128–1141, 2016.

[GRRV14] Vipul Goyal, Silas Richelson, Alon Rosen, and Margarita Vald. An algebraic approach to non-malleability. In *FOCS*, pages 41–50, 2014.

[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.

[HPV16] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper-proof hardware model under minimal complexity. In *TCC*, pages 367–399, 2016.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *To Appear ASIACRYPT*, 2017.

[Khu17] Dakshita Khurana. Round optimal concurrent non-malleability from polynomial hardness. *To Appear TCC*, 2017.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 578–595. Springer, Heidelberg, May 2003.

[LP11] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 705–714. ACM Press, June 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.

[LPV08] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 571–588. Springer, Heidelberg, March 2008.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, pages 735–763, 2016.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

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

A Secure Multi-Party Computation

We briefly present the standard definition for secure multi-party computation and refer to [Gol04, Chapter 7] for more details and motivating discussions. A multi-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f : \{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^* \times \cdots \times \{0,1\}^*$, where $f = (f_1, \ldots, f_n)$. That is, for every tuple of inputs $(x_1, \ldots, x_n)$, the output-vector is a random variable $(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$ ranging over tuples of strings where P_i receives $f_i(x_1, \ldots, x_n)$. We use the notation $(x_1, \ldots, x_n) \mapsto (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$ to describe a functionality.

We prove the security of our protocols in the settings of *honest-but-curious* and *malicious* computationally bounded adversaries. Security is analyzed by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario. In the ideal scenario, the computation involves an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Informally, the protocol is secure if any adversary interacting in the real protocol (i.e., where no trusted third party exists) can do no more harm than what it could do in the ideal scenario.

A.1 The Honest-but-Curious Setting

In this model the adversary controls one of the parties and follows the protocol specification. However, it may try to learn more information than allowed by looking at the transcript of messages that it received and its internal state. Let $f = (f_1, \ldots, f_n)$ be a multi-party functionality and let π be a multi-party protocol for computing f. The *view* of the ith party in an execution of π on inputs $(x_1, \ldots, x_n)$ is

$$\mathbf{View}_{\pi,i}(x_1, \ldots, x_n) = (x_i, r_i, m_1^i, \ldots, m_t^i),$$

where r_i is the content of the first party's internal random tape, and m_j^i represents the jth message that it received. The *output* of the ith party in an execution of π on $(x_1, \ldots, x_n)$ is denoted $\mathbf{Output}_{\pi,i}(x_1, \ldots, x_n)$ and can be computed from $\mathbf{View}_{\pi,i}(x_1, \ldots, x_n)$. We denote the set of corrupted parties by $I \subset [n]$ and the set of honest parties by $\bar{I}$. We extend the above view notation to capture any subset of parties, denoting by $\mathbf{View}_{\pi,T}(\kappa, x_1, \ldots, x_n)$ the joint views of all parties in T on $(\kappa, x_1, \ldots, x_n)$.

Definition A.1 *Let f and π be as above. Protocol π is said to securely compute f in the presence of* honest-but-curious *adversaries if for every $I \subset [n]$ there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that*

$$\begin{aligned}&(\mathcal{S}(\{x_i, f_i(\kappa, x_1, \ldots, x_n)\}_{i\in I}), \{f_i(\kappa, x_1, \ldots, x_n)\}_{i\notin I})_{\kappa\in\mathbb{N}, x_i\in\{0,1\}^*} \\ &\quad \stackrel{c}{\approx} \quad \{(\mathbf{View}_{\pi,I}(\kappa, x_1, \ldots, x_n), \mathbf{Output}_{\pi,\bar{I}}(\kappa, x_1, \ldots, x_n))\}_{\kappa\in\mathbb{N}, x_i\in\{0,1\}^*}\end{aligned}$$

where κ is the security parameter.

A.2 The Malicious Setting

Execution in the ideal model. In an ideal execution, the parties submit inputs to a trusted party, that computes the output. An honest party receives its input for the computation and just directs it to the trusted party, whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the outputs of the corrupted parties to the adversary, and the adversary then decides whether the honest parties would receive their outputs from the trusted party or an *abort* symbol $\perp$. Let f be a multi-party functionality where $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subset [n]$ be the set of corrupted parties. Then, the *ideal execution of* f on inputs $(\kappa, x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{IDEAL}_{f,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of the specified protocol π.

Let f be as above and let π be a multi-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the *real execution of* π on inputs $(\kappa, x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(\kappa, x_1, \ldots, x_n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure protocol (in

the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

Definition A.2 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [n]$,*

$$\left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(\kappa, x_1, \ldots, x_n)\right\}_{\kappa\in\mathbb{N},x_i,z\in\{0,1\}^*} \stackrel{c}{\approx} \left\{\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(\kappa, x, y)\right\}_{\kappa\in\mathbb{N},x_i,z\in\{0,1\}^*}$$

where κ is the security parameter.

The $\mathcal{F}$-hybrid model. In order to construct some of our protocols, we will use secure multi-party protocols as subprotocols. The standard way of doing this is to work in a "*hybrid model*" where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, when constructing a protocol π that uses a subprotocol for securely computing some functionality $\mathcal{F}$, we consider the case that the parties run π and use "ideal calls" to a trusted party for computing $\mathcal{F}$. Upon receiving the inputs from the parties, the trusted party computes $\mathcal{F}$ and sends all parties their output. Then, after receiving these outputs back from the trusted party the protocol π continues.

Let $\mathcal{F}$ be a functionality and let π be a multi-party protocol that uses ideal calls to a trusted party computing $\mathcal{F}$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine. Then, the *$\mathcal{F}$-hybrid execution of* π on inputs $(x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{H}_{\pi^{\mathcal{F}},\mathcal{A}(z)}(\kappa, x_1, \ldots, x_n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $\mathcal{F}$. By the composition theorem of [Can00] any protocol that securely implements $\mathcal{F}$ can replace the ideal calls to $\mathcal{F}$.