J Cryptol (2021) 34:19
https://doi.org/10.1007/s00145-021-09382-3

Journal of
CRYPTOLOGY

Round-Optimal Secure Multi-party Computation*

Shai Halevi
Algorand Foundation, Yorktown Heights, USA
shaih@alum.mit.edu

Carmit Hazay
Bar-Ilan University, Ramat Gan, Israel
Carmit.hazay@biu.ac.il

Antigoni Polychroniadou
J.P. Morgan AI Research, New York, USA
antigoni.poly@jpmorgan.com

Muthuramakrishnan Venkitasubramaniam
University of Rochester, Rochester, USA
muthuv@cs.rochester.edu

Communicated by Nigel Smart.

Received 31 March 2019 / Revised 10 March 2021 / Accepted 10 March 2021

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 an active (i.e. malicious) adversary that can corrupt any number of parties. Despite extensive research, the precise round complexity of this "standard-bearer" cryptographic primitive, under polynomial-time hardness assumptions, 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

*S. Halevi: Research supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236. C. Hazay: 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. A. Polychroniadou: This work was supported in part by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. M. Venkitasubramaniam: Research supported by Google Faculty Research Grant and NSF Award CNS-1526377.

Published online: 13 May 2021

and Brakerski, Halevi, and Polychroniadou, TCC 2017 made progress towards solving this question and constructed four-round protocols based on the DDH and LWE assumptions, respectively, albeit with super-polynomial hardness. More recently, Ciampi, Ostrovsky, Siniscalchi and Visconti in TCC 2017 closed the gap for two-party protocols by constructing a four-round protocol from polynomial-time assumptions, concretely, trapdoor permutations. In another work, Ciampi, Ostrovsky, Siniscalchi and Visconti TCC 2017 showed how to design a four-round multi-party protocol for the specific case of multi-party coin-tossing based on one-way functions. In this work, we resolve this question by designing a four-round actively secure multi-party (two or more parties) protocol for general functionalities under standard polynomial-time hardness assumptions with a black-box proof of security, specifically, under the assumptions LWE, DDH, QR and DCR.

Keywords. Secure multi-party computation, Garbled circuits, Round complexity, Additive errors.

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 [9,13,31,51].

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 [30], Garg, et al., proved a lower bound of four rounds for MPC protocols that rely on black-box simulation. Following this work, in independent works, Ananth, Choudhuri and Jain [2] and Brakerski, Halevi and Polychroniadou, [10] showed a matching upper bound by constructing four-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 [18] closed the gap for two-party protocols by constructing a four-round protocol from standard polynomial-time assumptions. The same authors in another work [17] showed how to design a four-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:

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

We remark that tight answers have been obtained in prior works where one or more of the requirements in the motivating question are relaxed. In the two-party setting, the recent work of Ciampi et al. [18] showed how to obtain a four-round protocol based on trapdoor permutations. Assuming trusted setup, namely, a common reference string, two-round constructions can be obtained [24,45] or three-round assuming tamper-proof

hardware tokens [38].[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. [10]. Ananth et al. gave a five-round protocol based on DDH [2]. Under subexponential hardness assumptions, four-round constructions were demonstrated in [2,10]. Under some relaxations of superpolynomial simulation, the work of Badrinarayanan et al. [7] shows how to obtain three-round MPC assuming subexponentially secure LWE and DDH. For specific multi-party functionalities four-round constructions have been obtained, e.g., coin-tossing by Ciampi et al. [17]. Finally, if we assume an honest majority, the work of Damgard and Ishai [19] provided a three-round MPC protocol for $t < n/5$ corruptions whereas Ananth et al. [1] dealt with the case of $t < n/2$.

1.1. *Our Results*

The main result we establish is a four-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.

Theorem 1.1. (Informal) *Assuming the existence of injective one-way functions, ZAPs and a certain affine homomorphic encryption scheme with an equivocation property, there exists a four-round multi-party protocol that securely realizes arbitrary functionalities in the presence of active adversaries corrupting any number of parties.*

This theorem addresses our motivating question and resolves the round complexity of secure multiparty computation protocols under polynomial-time hardness assumptions. The encryption scheme that we need admits a homomorphic affine transformation

$$c = \mathsf{Enc}(m) \mapsto c' = \mathsf{Enc}(a \cdot m + b) \text{ for plaintext } a, b,$$

as well as some equivocation property. Roughly, given the secret key and encryption randomness, it should be possible to "explain" the result c' as coming from $c' = \mathsf{Enc}(a' \cdot m + b')$, for any a', b' satisfying $am + b = a'm + b'$. 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, Quadratic Residuosity (QR) and Decisional Composite Residuosity (DCR) hardness assumptions. ZAPs on the other hand can be instantiated using the QR assumption or any (doubly) enhanced trapdoor permutation such as RSA or bilinear maps. Injective one-way functions can be instantiated using the QR assumption. In summary, all our primitives can be instantiated by the single QR assumption and therefore we have the following corollary

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

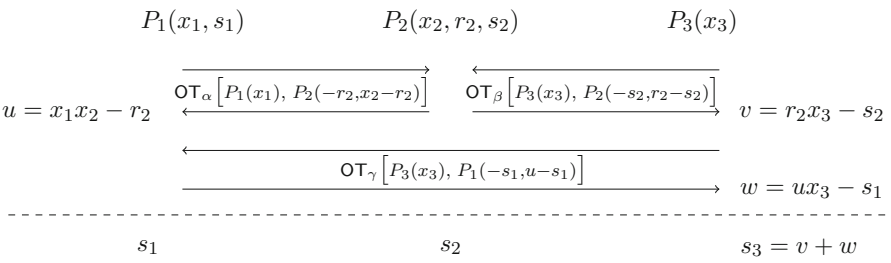

Fig. 1. The three-bit multiplication protocol from [2], using two-round oblivious transfer. The OT subprotocols are denoted by $\mathsf{OT}[\text{Receiver}(b), \text{Sender}(m_0, m_1)]$, and u, v, w are the receivers' outputs in the three OT protocols. The outputs of P_1, P_2, P_3 are s_1, s_2, s_3, respectively. The first message in OT_γ can be sent in the second round, together with the sender messages in OT_α and OT_β .

Corollary 1.2. *Assuming QR, there exists a four-round multi-party protocol that securely realizes arbitrary functionalities in the presence of active adversaries corrupting any number of parties.*

All our results assume a broadcast channel and a private channel between every pair of parties.

1.2. *Our Techniques*

Starting point: the [2] **protocol.** We begin from the beautiful work of Ananth, Choudhuri and Jain [2], where they used randomized encoding [3] to reduce the task of securely computing an arbitrary functionality to securely computing the sum of many three-bit multiplications. To implement the required three-bit multiplications, Ananth et al. used an elegant three-round protocol, consisting of three instances of a two-round semi-honest oblivious-transfer (OT) subprotocol, as illustrated in Fig. 1.

In more detail, the construction in [2] proceeds in three steps:

1. Design a 3-round 3-party protocol tolerating semi-honest adversaries for computing the share of the product of three input elements. Namely, if P_i holds x_i for $i \in \{1, 2, 3\}$, the parties learn additive shares of $x_1 \cdot x_2 \cdot x_3$. Furthermore, this protocol satisfies a robustness feature where an active adversary can at most influence the "correctness" of the computation but cannot affect privacy.
2. Next, they reduce an arbitrary n-party MPC task to computing degree-3 polynomially relying on the randomizing polynomial construction of [4]. In essence, this step allows for securely computing any MPC functionality by computing several degree-3 terms in parallel. This extends to a four-round protocol to securely compute arbitrary functionalities against semi-honest adversaries.
3. Finally, they amplify semi-honest to full security by incorporating "witness encryption" where parties share their outputs in the 4th round in a way that the result can be "decrypted" only if they can provide a witness to demonstrate honest behavior in the first three rounds.

Next, enforced correctness in the third and fourth rounds using zero-knowledge proofs to get security against an active adversary. In particular, the proof of correct behavior in the third round required a special three-round non-malleable zero-knowledge proofs, for which they had to rely on super-polynomial hardness assumptions. (A four-round proof to enforce correctness in the last round can be done based on standard assumptions.) To eliminate the need for super-polynomial assumptions, our very high level approach is to weaken the correctness guarantees needed in the third round, so that we can use simpler proofs. Specifically we would like to be able to use two-round (resettable) witness indistinguishable (WI) proofs (aka ZAPs [22]).

WI using the Naor-Yung approach. To replace zero-knowledge proofs by ZAPs, we must be able to use the honest prover strategy (since ZAPs have no simulator), even as we slowly remove the honest parties' input from the game. We achieve this using the Naor-Yung approach: We modify the three-bit multiplication protocol by repeating each OT instance twice, with the receiver using the same choice bit in both copies and the sender secret-sharing its input bits between the two. (Thus we have a total of six OT instances in the modified protocol.) Crucially, while we require that the sender proves correct behavior relative to its inputs in both instances, we only ask the receiver to prove that it behaves correctly in *at least one of the two*.

In the security proof, this change allows us to switch in two steps from the real world where honest parties use their real inputs as the choice bit, to a simulated world where they are simulated using random inputs. In each step we change the choice bit in just one of the two OT instances, and use the other bit that we did not switch to generate the ZAP proofs on behalf of the honest parties.[3]

We note that intuitively, this change does not add much power to a real-world adversary: Although an adversarial receiver can use different bits in the two OT instances, this will only result in that receiver getting random bits from the protocol, since the sender secret-shares its input bits between the two instances.

Extraction via rewinding. While the adversary cannot gain much by using different bits in different OT instances, we crucially rely on the challenger in our hybrid games to use that option. Hence we must compensate somehow for the fact that the received bits in those OT protocols are meaningless. To that end, the challenger (as well as the simulator in the ideal model) will use rewinding to extract the necessary information from the adversary.

But rewinding takes rounds, so the challenger/simulator can only extract this information at the end of the third round.[4] Thus we must rearrange the simulator so that it does not need the extracted information—in particular the bits received in the OT protocols—until after the third round. Looking at the protocol in Fig. 1, there is only one place where a value received in one of the OTs is used before the end of the third round. Specifically, the value u received in the second round by P_1 in OT_α is used in the third round when P_1 plays the sender in OT_γ.

[3] We do not need to apply a similar trick to the sender role in the OT subprotocols, since the sender bits are always random.

[4] To get it by then, the ZAPs are performed in parallel to the second and third rounds of the three-bit multiplication protocol.

This causes a real problem in the security proof: Consider the case where P_2 is an adversarial sender and P_1 an honest receiver. In some hybrid we would want to switch the choice bit of P_1 from its real input to an arbitrary bit, and argue that these hybrids are close by a reduction to the OT receiver privacy. Inside the reduction, we cannot use the knowledge of the value received in the OT, so we cannot ensure that it is consistent with the value that P_1 uses as the sender in OT_γ (with P_3 as the receiver). We would like to extract the value of u from the adversary, but we are at a bind: we must send to the adversary the last message of OT_γ before we can extract u, but we cannot compute that message without knowing u.

Relaxing the correctness guarantees. To overcome the difficulty from above, we relax the correctness guarantees of the three-bit multiplication protocol, allowing the value that P_1 sends in OT_γ (which we denote by u') to differ from the value that it received in OT_α (denoted u). The honest parties will still use $u' = u$, but the protocol no longer includes a proof for that fact (so the adversary can use $u' \neq u$, and so can the challenger). This modification lets us introduce into the proof an earlier hybrid in which the challenger uses $u' \neq u$, even on behalf of an honest P_1. (That hybrid is justified by the sender privacy of OT_γ.) Then, we can switch the choice bit of P_1 in OT_α from real to random, and the reduction to the OT receiver privacy in OT_α will not need to use the value u, see Claim 3.3 and the hybrids that precede it.[5]

Dealing with additive errors. Since the modified protocol no longer requires proofs that $u' = u$, an adversarial P_1 is free to use $u' \neq u$, thereby introducing an error into the three-bit multiplication protocol. Namely, instead of computing the product $x_1x_2x_3$, an adversarial P_1 can cause the result of the protocol to be $(x_1x_2 + (u' - u))x_3$. Importantly, the error term $e = u' - u$ cannot depend on the input of the honest parties. (The reason is that the value u received by P_1 in OT_α is masked by r_2 and hence independent of P_2's input x_2, so any change made by P_1 must also be independent of x_2.), see Claim 3.5.

To deal with this adversarial error, we want to use a randomized encoding scheme which is resilient to such additive attacks. Indeed, Genkin et al. presented transformations that do exactly this in [25–27]. Namely, they described a compiler that transforms an arbitrary circuit C to another circuit C′ that is resilient to additive attacks. Unfortunately, using these transformations do not work out of the box, since they do not preserve the degree of the circuit. So even if after using randomized encoding we get a degree-three function, making it resilient to additive attacks will blow up the degree, and we will not be able to use the three-bit multiplication protocol as before.

What we would like, instead, is to first transform the original function f that we want to compute into a resilient form $\hat{f}$, then apply randomized encoding to $\hat{f}$ to get a degree-three function g that we can use in our protocol. But this too does not work out of the box: The adversary can introduce additive errors in the circuit of g, but we only know that $\hat{f}$ is resilient to additive attacks, not its randomized encoding g.

BMR to the rescue. To tackle this last problem, we forgo "generic" randomized encoding, relying instead on the specific multiparty garbling due to Beaver, Micali and

[5] The reduction will still need to use u in the fourth round of the simulation, but by then we have already extracted the information that we need from the adversary.

Rogaway [12] (referred to as "BMR encoding"). For this specific encoding, we carefully align the roles in the BMR protocol to those in the three-bit multiplication protocol, and show that the errors in the three-bit multiplication instances with a corrupted P_1 can be effectively translated to an additive attack against the underlying computation of $\hat{f}$, see Lemma 3.6. Our final protocol, therefore, precompiles the original function f to $\hat{f}$ using the transformations of Genkin et al., then applies the BMR encoding to get $\hat{f}'$ which is of degree-three and still resilient to the additive errors by a corrupted P_1. We remark here that another advantage of relying on BMR encoding as opposed to the randomized encoding from [4] is that it can be instantiated based on any one-way function. In contrast the randomized encoding of [4] requires the assumption of PRGs in NC^1.

1.2.1. *Other Technical Issues*

Non-malleable commitments. Recall that we need a mechanism to extract information from the adversary before the fourth round, while simultaneously providing proofs of correct behavior for honest parties via ZAPs. In fact, we need the stronger property of *non-malleability*, namely the extracted information must not change when the witness in the ZAP proofs changes.

Ideally, we would want to use standard non-malleable commitments and recent work of Khurana [41] shows how to construct such commitments in three rounds. However, our proof approach demands additional properties of the underlying non-malleable commitment, and we do not know how to construct such commitments in three rounds. Hence we relax the conditions of standard non-malleable commitments. Specifically, we allow for the non-malleable commitment scheme to admit invalid commitments. (Such weaker commitments are often used as the main tool in constructing full-fledged non-malleable commitments, see [35,41] for few examples.)

A consequence of this relaxation is the problem of "over-extraction" where an extractor extracts the wrong message from an invalid commitment. We resolve this in our setting by making each party provide two independent commitments to its witness, and modify the ZAP proofs to show that at least one of these two commitments is a valid commitment to a valid witness.

This still falls short of yielding full-fledged non-malleable commitments, but it ensures that the witness extracted in at least one of the two commitments is valid. Since the witness in our case includes the input and randomness of the OT subprotocols, the challenger in our hybrids can compare the extracted witness against the transcript of the relevant OT instances and discard invalid witnesses.

Another obstacle is that in some intermediate hybrids, some of the information that the challenger should commit to is only known in later rounds of the protocol, hence we need the commitments to be *input-delayed*. For this we rely on a technique of Ciampi et al. [16] for making non-malleable commitments into input-delayed ones. Finally, we observe that we can instantiate the "weak non-malleable commitments" that we need from the three-round non-malleable commitment scheme by using a scheme that is implicit in the work of Goyal et al. [35].

Equivocable oblivious transfer. In some hybrids in the security proof, we need to switch the sender bits in the OT subprotocols. For example in one step we switch the P_2 sender

inputs in OT_α from $(-r_2, x_2 - r_2)$ to $(-r_2, \tilde{x}_2 - r_2)$ where x_2 is the real input of P_2 and $\tilde{x}_2$ is a random bit. (We also have a similarly step for P_1's input in OT_γ.)

For every instance of OT, the challenger needs to commit to the OT randomness on behalf of the honest party and prove via ZAP that it behaved correctly in the protocol. Since ZAPs are not simulatable, the challenger can only provide these proofs by following the honest prover strategy, so it needs to actually have the sender randomness for these OT protocols. Recalling that we commit twice to the randomness, our security proof goes through some hybrids where in one commitment we have the OT sender randomness for one set of values and in the other we have the randomness for another set. (This is used to switch the ZAP proof from one witness to another, see Claim 4.7.)

But how can there be two sets of randomness values that explain *the same OT transcript*? To this end, we use an *equivocable* oblivious transfer protocol. Namely, given the receiver's randomness, it is possible to explain the OT transcript after the fact, in such a way that the "other sender bit" (the one that the receiver does not get) can be opened both ways. In all these hybrids, the OT receiver gets a random output bit. So the challenger first runs the protocol according to the values in one hybrid, then rewinds the adversary to extract the randomness of the receiver, where it can then explain (and hence prove) the sender's actions in any way that it needs, while keeping the OT transcript fixed.

We show how to instantiate the equivocable OT that we need from (a slightly weak variant of) additive homomorphic encryption, with an additional equivocation property. Such encryption schemes can in turn be constructed under standard (polynomial) hardness assumptions such as LWE, DDH, Quadratic Residuosity (QR) and Decisional Composite Residuosity (DCR).

Premature rewinding. One subtle issue with relying on equivocable OT is that equivocation requires knowing the randomness of the OT receiver. To get this randomness, the challenger in our hybrids must rewind the receiver, so we introduce in some of the hybrids another phase of rewinding, which we call "premature rewinding." This phase has nothing to do with the adversary's input, and it has no effect on the transcript used in the main thread. All it does is extract some keys and randomness, which are needed to equivocate. For more details, see the discussion in Sect. 4.1.

No four-round proofs. A side benefit of using BMR garbling is that the authentication properties of BMR let us do away completely with the four-round proofs from [2]. In our protocol, at the end of the third round the parties hold a secret sharing of the garbled circuit, its input labels, and the translation table to interpret the results of the garbled evaluation. Then in the last round they just broadcast their shares and input labels, then reconstruct the circuit, evaluate the circuit, and recover the result.

Absent a proof in the fourth round, the adversary can report arbitrary values as its shares, even after seeing the shares of the honest parties, but we argue that it still cannot violate privacy or correctness. It was observed in prior work [44] that faulty shares for the garbled circuit itself or the input labels can at worst cause an honest party to abort, and such an event will be independent of the inputs of the honest parties. Roughly speaking, this is because the so-called active path in the evaluation is randomized by masks from each party. Furthermore, if an honest party does not abort and completes evaluation, then the result is correct. This was further strengthened in [39], and was shown to hold

even when the adversary is rushing. One course of action still available to the adversary is to modify the translation tables, arbitrarily making the honest party output the wrong answer. This can be fixed by a standard technique of precompiling f to additionally receive a MAC key from each party and output the MACs of the output under all keys along with the output. Each honest party can then verify the garbled-circuit result using its private MAC key.

A modular presentation with "defensible" simulation. In order to make our presentation more modular, we separate the issues of extraction and non-malleability from the overall structure of the protocol by introducing the notion of a "defensible" simulation. Specifically, we first prove security in a simpler model in which the adversary voluntarily provides the simulator with some extra information. In a few more details, we consider an "explaining adversary" that at the end of the third round outputs a "defense" (or explanation) for its actions so far. [6]

This model is somewhat similar to the semi-malicious adversary model of Asharov et al. [5] where the adversary outputs its internal randomness with every message. The main difference is that here we (the protocol designers) get to decide what information the adversary needs to provide and when. We suspect that our model is also somewhat related to the notion of robust semi-honest security defined in [2], where, if a protocol admits defensible simulation and a defense is required after the k^{th} round of the protocol, then it is plausible that the first k rounds admits robust semi-honest security.

Once we have a secure protocol in this weaker model, we add to it commitment and proofs that would let us extract from the adversary the same information that was provided in the "defense". As we hinted above, this is done by having the adversary commit to that information using (a weaker variant of) simulation extractable commitments, and also prove that the committed values are indeed a valid "defense" for its actions. While in this work we introduce "defensible" simulation merely as a convenience to make the presentation more modular, we believe that it is a useful tool for obtaining round-efficient protocols.

1.2.2. *Our Final Protocol*

Combining all these ideas, our final protocol proceeds as follows:

Circuit Preprocessing: Let C be a circuit that we want to evaluate securely. First, we apply the Genkin et al. transformation to get resilience against additive attacks. Next, we apply the distributed garbling procedure of BMR to gives us a randomized encoding for our original circuit $\mathsf{C}_{\mathsf{BMR}}(\mathbf{x}; (\lambda, \rho)) = (x \oplus \lambda, g(\lambda, \rho))$ where $\mathbf{x}$ are the inputs, λ are the mask bits and ρ are the keys used in the garbling. Furthermore, each output bit of g has degree three (or less) in (λ, ρ). We consider a slight variant of $\mathsf{C}_{\mathsf{BMR}}$ where view it as a n-party functionality, where the inputs $x = (x_1, \ldots, x_n)$ are distributed across the parties, we have the parties choose respective pieces of the BMR randomness λ^i, ρ^i, and engage in our modified three-bit multiplication protocol Π' (with a pair of OT's for each one in Fig. 1), to compute the outputs of $g(\lambda, \rho)$. Additionally, the inputs are used only in the third round message of Π',

[6] The name "defensible" simulation is adapted from the "defensible adversaries" of Haitner et al. [37].

where each party P_i broadcasts its masked input $x_i \oplus \lambda^i$, see more details regarding the BMR encoding in Sect. 2.9.

Rounds 1-3: The parties engage in several instances of three-bit multiplication protocol in the first three rounds to compute shares of the distributed garbling of the circuit. Additionally, in the third round, parties use their real inputs by broadcasting $x_i \oplus \lambda^i$. In parallel, the parties engage in the following interactions:

1. In Rounds 1 through 3, for every i and j, party P_i runs two executions of non-malleable commitments $\mathsf{nmcom}^0_{i,j}$, $\mathsf{nmcom}^1_{i,j}$ with every other party P_j in the first three rounds. Let wit_i be a witness for P_i's actions in 3-bit multiplication protocols, that constitute the defense in this protocol. P_i chooses two random strings $w^0_{i,j}, w^1_{i,j}$, and uses specifically the GRRV protocol [35] to commit to them separately, in two commitment protocols. In the third round, together with the last messages of $\mathsf{nmcom}^0_{i,j}$, $\mathsf{nmcom}^1_{i,j}$, P_i also sends $\tilde{w}^0_{i,j} := \mathsf{wit}_i \oplus w^0_{i,j}$ and $\tilde{w}^1_{i,j} := \mathsf{wit}_i \oplus w^1_{i,j}$, respectively.
2. Every party P_i proves using ZAPs to every other party P_j in the first two rounds that it generated correctly all the public keys and ciphertexts in the first round of Π_{DMPC}.
3. Finally, every party P_i proves using ZAPs to every other party P_j from Round 2 to Round 3 that at least one of $\mathsf{nmcom}^0_{i,j}$, $\mathsf{nmcom}^1_{i,j}$ is a valid commitment to a valid witness. Namely for some $b \in \{0, 1\}$, $\mathsf{nmcom}^b_{i,j}$ is a valid commitment to a value $w^b_{i,j}$ such that $w^b_{i,j} \oplus \tilde{w}^b_{i,j}$ is a valid defense.

Round 4: Parties broadcast their shares for the distributed garbling and the input encodings with which all parties evaluate the garbled circuit to learn the result.

1.3. *Concurrent and Followup Work*

The earliest MPC protocol is due to Goldreich, Micali and Wigderson [31]. 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. Although [31] requires order of circuit depth round complexity, using randomized encoding the depth of the circuit can be constant. The blowup of the round complexity in [31] comes from the ZK proofs that are run sequentially, making the round complexity be dependent on the number of parties.

A different approach was taken in [12], extending the celebrated garbled circuits technique of [51] 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). Katz, Ostrovsky and Smith [42] provided a constant-round MPC protocol secure against active adversaries in the dishonest majority setting while relying on non-black-box simulation [6] and superpolynomial assumptions. This was later improved by Pass [49] to obtain the first constant-round protocol based on standard polynomial hardness assumptions. The first constant-round MPC protocols that relied on black-box simulation were obtained by Goyal [34] and Lin and Pass [43].

In concurrent work, Badrinarayanan et al. [8] establish the main feasibility result presented in this work, albeit with a different abstraction and slightly different assumptions. In particular, their main idea is to consider promise zero-knowledge against adversarial verifiers that promise not to abort the protocol early with noticeable probability. To ensure full security, this primitive is combined with a three-round delayed input WI proof. We remark that the premature rewinding technique employed in our work bears similarities to the promise zero-knowledge and partitioned simulation strategy in their work. In another concurrent work, Benhamouda and Lin [11] construct a five-round MPC protocol based on minimal assumptions. While this protocol relies on the minimal assumption of four-round OT protocol, it requires an additional round to construct an MPC protocol. Simultaneously, Garg and Srinivasan [36] construct a two-round MPC protocol assuming a two-round OT in the CRS model. Finally, in a recent followup work by Choudhuri et al. [14], the authors extends the protocols from [8] and the techniques of promise zero-knowledge, building a four-round MPC protocol based on oblivious-transfer and resolving the question of the minimal assumption required for four-round MPC. In more details, they use garbled circuits and OT for designing promise zero-knowledge under the hood, where the witness is not given in the clear but encrypted inside the OT receiver message.

1.4. *A Roadmap*

In Sect. 3.2 we provide our four-round three-bit multiplication protocol Π_{DMULT} with defensible simulation. In Sect. 3.3 we extend the four-round protocol from the previous section that computes an arbitrarily degree-3 polynomials. In Sect. 3.4 we compile this protocol into a defensible simulatable protocol which securely computes arbitrary functionalities (via the BMR garbling). Finally, in Sect. 4 we compile protocol Π_f that is secure against defensible adversaries to a four-round actively 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 additive shares of which their sum 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 $s_{\mathrm{N}} = s - \bigoplus_{i=1}^{\mathrm{N}-1} s_i$.
  2. Output $(s_1, \ldots, s_{\mathrm{N}})$ where s_i is the share of the i^{th} party.
- The recovery algorithm $\mathsf{Recover}$ on input $(s_1, \ldots, s_{\mathrm{N}})$, outputs $\bigoplus_{i=1}^{\mathrm{N}} s_i$.

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 $[s] = (s_1, \ldots, s_{\mathrm{N}})$. For better readability, we use throughout the paper the notation $s_1 + s_2$ to denote $s_1 \oplus s_2$.

2.2. *Pseudorandom Functions*

Informally, a pseudorandom function (PRF) is an efficiently computable function that looks like a truly random function to any PPT observer. The [12] garbling technique from [44], 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 $F : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ 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 $\{0,1\}^n \to \{0,1\}^n$. 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. *Commitment Schemes*

Commitment schemes are used to enable a party, known as the *sender* Sen, to commit itself to a value while keeping it secret from the *receiver* Rec (this property is called *hiding*). Furthermore, in a later stage when the commitment is opened, it is guaranteed that the "opening" can yield only a single value determined in the committing phase (this property is called *binding*). In this work, we consider commitment schemes that are *statistically binding*, namely while the hiding property only holds against computationally bounded (non-uniform) adversaries, the binding property is required to hold against unbounded adversaries. Formally,

Definition 2.5. (*Commitment schemes*) A PPT machine $\mathsf{Com} = \langle S, R \rangle$ is said to be a non-interactive commitment scheme if the following two properties hold.

Computational hiding: For every (expected) PPT machine Rec*, it holds that the following ensembles are computationally indistinguishable.

- $\{\mathbf{View}^{\mathrm{Rec}^*}_{\mathsf{Com}}(m_1, z)\}_{\kappa \in \mathbb{N}, m_1, m_2 \in \{0,1\}^\kappa, z \in \{0,1\}^*}$
- $\{\mathbf{View}^{\mathrm{Rec}^*}_{\mathsf{Com}}(m_2, z)\}_{\kappa \in \mathbb{N}, m_1, m_2 \in \{0,1\}^\kappa, z \in \{0,1\}^*}$

where $\mathbf{View}^{R^*}_{\mathsf{Com}}(m, z)$ denotes the random variable describing the output of Rec* after receiving a commitment to m using Com.

Statistical binding: For any (computationally unbounded) adversarial sender Sen* and auxiliary input z, it holds that the probability that there exist valid decommitments to two different values for a view v, generated with an honest receiver while interacting with $\mathrm{Sen}^*(z)$ using Com, is negligible.

We refer the reader to [32] for more details. We recall that non-interactive perfectly binding commitment schemes can be constructed based on one-way permutations, whereas two-round statistically binding commitment schemes can be constructed based on one-way functions [46]. We further consider *pseudorandom* commitments for which the honest sender's messages in the commitment phase are pseudorandom, i.e. indistinguishable from a uniform string of the same length. We note that such commitment schemes with statistical binding can be constructed based on one-way functions due to [46] and with perfect binding based on one-way permutations.

A delayed-input commitment scheme is a commitment scheme that retains its security even if the adversary can decide one of the challenge messages in the last round. In [COSV16] it is showed that any commitment scheme can be made delayed-input.

2.4. *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 Learning With Errors (LWE), Decisional Diffie–Hellman (DDH), Quadratic Residuosity (QR) and Decisional Composite Residuosity (DCR) hardness assumptions.

Definition 2.6. (*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 a probabilistic 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 $c = \mathsf{Enc}_{\mathrm{PK}}(m; r_c)$, $\mathcal{D}_{rand}(1^\kappa)$ is the randomness distribution used by Enc and $am + b$ is computed over $\mathbb{F}_2$.
- *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 2.6 under a variety of hardness assumptions.

2.4.1. *An Instantiation Based on LWE*

Definition 2.7. (*LWE* [50]) 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 $t + a_{1-\sigma}(1 - 2 \cdot b_{1-\sigma}) \cdot r_c = r + a_\sigma(1 - 2 \cdot b_\sigma) \cdot r_c$.

2.4.2. *An Instantiation Based on DDH*

Definition 2.8. (*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 [23] 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 $t + r_c \cdot a_{1-\sigma}(1 - 2 \cdot b_{1-\sigma}) = r + r_c \cdot a_\sigma (1 - 2 \cdot b_\sigma) \mod p$.

2.4.3. *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 2.9. (*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 [29] 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 $c = z^m \cdot r_c^2 \mod N$.
- **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 $t = r_c^{a_\sigma} \cdot r / r_c^{a_{1-\sigma}}$.

2.4.4. *An Instantiation Based on DCR*

Definition 2.10. (*The Decisional Composite Residuosity (DCR) Problem*) Let N is a random κ-bit RSA composite. Let r and y be sampled uniformly at random from $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$, respectively, (i.e., $r \leftarrow \mathbb{Z}_N$ and $y \leftarrow \mathbb{Z}_N^*$). The DCR problem asks to distinguish the distributions $(N, r^N \mod N^2)$ and $(N, (N+1)^y \cdot r^N \mod N^2)$.

We next describe a variant of the Paillier encryption scheme [48] over characteristic-2 fields, as defined in [21]. As for the El-Gamal encryption scheme, we compute the encryption of $a \oplus b$, we compute $a + b - 2ab = a \oplus b$.

- **Key generation:** Choose a random RSA composite $\mathrm{PK} = N$ that is a product of two random κ-bit primes (p, q) and fix $\mathrm{SK} = (N, \phi(N))$.
- **Encryption:** Given the public key PK and a plaintext $m \in \{0, 1\}$, choose $r_c \leftarrow \mathbb{Z}_N^*$ and output the ciphertext $c = (1+N)^m \cdot r_c^N \mod N^2$.
- **Decryption:** Given the secret key SK and the ciphertext c, compute $\frac{[c^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N$.
- **Affine transformation:** Given PK, c and $a, b \in \{0, 1\}$, choose $r \leftarrow \mathbb{Z}_N^*$ and set $c^* = (c^{a(1-2\cdot b)}(1+N)^b \cdot r^N$.
- **Equivocation:** Note that the ciphertext $c^* = (1+N)^{b_\sigma + m \cdot a_\sigma (1-2\cdot b_\sigma)}$ 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 $t \cdot r_c^{a_{1-\sigma}(1-2\cdot b_{1-\sigma})} = r \cdot r_c^{a_\sigma (1-2\cdot b_\sigma)} \mod N$.

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 $u \in \{0,1\}^\kappa$, 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 (chosen by M); 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 2.11. (*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), the following two distributions are computationally indistinguishable:

- $\{\mathsf{nmc}^M_{\langle C,R\rangle}(u^0, z)\}_{\kappa\in\mathbb{N}, u^0, u^1\in\{0,1\}^\kappa, z\in\{0,1\}^*}$.
- $\{\mathsf{nmc}^M_{\langle C,R\rangle}(u^1, z)\}_{\kappa\in\mathbb{N}, u^0, u^1\in\{0,1\}^\kappa, z\in\{0,1\}^*}$.

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. each round in the left and right execution proceed in lock-step where round $i+1$ is executed only after round i messages are transmitted from all executions). We produce the following definitions verbatim from [41].

Definition 2.12. (*One-Many weak non-malleable commitments with respect to synchronizing adversaries* [41]) 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 $\mathsf{E}_{\mathsf{nmcom}}$ parameterized by ε, that given a PPT synchronizing MIM which participates in one left session and $p = \mathsf{poly}(\kappa)$ right ses-

sions, and given the transcript of a main-thread interaction τ, outputs a set of values $m_1, m_2, \ldots m_p$ in time $\mathsf{poly}(\kappa, 1/\varepsilon)$. 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 2.13. (*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 $m(\kappa) = \mathsf{poly}(\kappa)$ 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.

2.5.1. *Non-malleable Commitments for Premature Rewinding*

For completeness, we repeat the non-malleable commitment subprotocol of Goyal et al. [35]. Consider the following protocol between a committer C with input message m and a receiver R and an identity id. The protocol employs an instance of a non-interactive commitment protocol Com.

Round 1: $C \rightarrow R$: Select $r_i \in \mathbb{F}_q$ $(i \in [n])$ uniformly at random. Send commitments $\mathsf{Com}(m), \mathsf{Com}(r_1), \ldots, \mathsf{Com}(r_n)$.

Round 2: $R \rightarrow C$: Select a random challenge $\alpha_i \in [2^{t_i}] \subset \mathbb{F}_q$ $(i \in [n])$ where $t_1, \ldots, t_n$ are deterministically chosen based on the identity id.

Round 3: $C \rightarrow R$: Send $a_i = r_i \alpha_i + m$ $(i \in [n])$.

Observe that there is no proof of validity in this subprotocol, so the response in the third message from an adversarial committer can be incorrect. The full protocol of [35] includes a zero-knowledge proof at the end where the committer proves that the response was correctly computed. In our work, the first three rounds will suffice. More precisely, as stated in Khurana [41], this three round sub-protocol satisfies the weak one-many non-malleability against synchronizing adversaries (cf. Definition 2.12).

We present a high-level intuition of why the GRRV subprotocol satisfies this definition. We need an extractor that can extract values from the (multiple) commitments made by the adversary A while receiving (a valid) commitment on the left (made honestly). The extractor can be instantiated using the extractor E provided in GRRV (cf. [35] eprint 2014/586, Figure 3, page 15). We apply this procedure to each commitment made by A on the right. Theorem 2 from GRRV ensures that when the commitment on the right is valid, the extractor succeeds in extracting the correct message (except with small probability). In our protocol, between every party P_i and P_j, P_i commits using two instances of GRRV, and from the soundness of the ZAP at least one of the two is a valid commitment, so the extractor from that copy will succeed. Moreover, the extracted value in our case is a "defense", i.e., an explanation of some other parts of the protocol, and it is enough for the overall "challenger" to get a single explanation in order to continue the protocol. Hence we don't care if we get over-extraction for invalid commitments, as long as we can extract a valid "defense" from at least one of the commitments.

Additionally, we require that our non-malleable commitment satisfies two properties

Simulatability Property: The simulatable property requires that the following games are indistinguishable for $b = 0$ and $b = 1$, for any pair of messages m_0 and m_1:

Game(b): Sample $r_1, \ldots, r_n \in \mathbb{F}_q$. Challenger generates first message of GRRV subprotocol using m_b and $r_1, \ldots, r_n$. The adversary repeatedly sends challenge messages for the GRRV subprotocol and the challenger responds according to $m_0, r_1, \ldots, r_n$.

Indistinguishability of the games follows directly from the hiding property of the non-interactive commitment scheme used to generate the first message. We remark that this is similar to the property described (and required) in the work of Garg et al. [28].

Special-Soundness Property: We say that the commitment is special sound if there exist a probabilistic polynomial-time extractor machine E, such that for all pairs of valid transcripts for proving $T_1 = (a, b_1, c_1)$, $T_2 = (a, b_2, c_2)$, where $b_1 \neq b_2$, $E(T_1, T_2)$ is the message (statistically)-bound in the first message. This follows directly from the construction as the committer reveals two linear combinations involving the randomness and the message in two distinct transcripts and the message can be extracted.

2.6. *Interactive Argument Systems*

An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in \mathcal{R}$ for some input x, and learns nothing beyond that. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w.

Definition 2.14. (*Interactive proof system*) A pair of PPT interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is called an *interactive proof system for a language* $\mathcal{L}$ if there exists a negligible function negl such that the following two conditions hold:

1. Completeness: For every $x \in \mathcal{L}$,

$$\Pr[\langle \mathcal{P}(w), \mathcal{V} \rangle(x) = 1] \geq 1 - \mathsf{negl}(|x|).$$

2. Soundness: For every $x \notin L$ and every interactive PPT machine B,

$$\Pr[\langle B, \mathcal{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$

2.7. *Witness Indistinguishability*

Definition 2.15. (*witness indistinguishable system*) A pair of PPT interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is called a *witness indistinguishable proof system for a language* $\mathcal{L}$ if for every PPT verifier $\mathcal{V}^*$ and every two sequences $\{w_x^1\}_{x \in \mathcal{L}}$ and $\{w_x^2\}_{x \in \mathcal{L}}$ such that w_x^1 and w_x^2 are both witnesses for x, the following ensembles are computationally indistinguishable:

1. $\{\langle \mathcal{P}(w_x^1), \mathcal{V}(z) \rangle(x)\}_{x \in \mathcal{L}, z \in \{0,1\}^*}$
2. $\{\langle \mathcal{P}(w_x^2), \mathcal{V}(z) \rangle(x)\}_{x \in \mathcal{L}, z \in \{0,1\}^*}$.

In this work, we will rely on witness-indistinguishable proof systems that additionally satisfy properties such as *delayed-input*, *reusability*, and resettability. These properties are presented in Definition 2.13.

2.8. *Additive Attacks and AMD Circuits*

In what follows we borrow the terminology and definitions verbatim from [25,27]. We recall that in this paper we work with the binary field $\mathbb{F}_2$.

Definition 2.16. (*AMD code* [15]) An (n, k, ε)-AMD code is a pair of circuits $(\mathsf{Encode}, \mathsf{Decode})$ where $\mathsf{Encode} : \mathbb{F}^n \to \mathbb{F}^k$ is randomized and $\mathsf{Decode} : \mathbb{F}^k \to \mathbb{F}^{n+1}$ is deterministic such that the following properties hold:

- *Perfect completeness.* For all $x \in \mathbb{F}^n$,

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(\mathbf{x})) = (0, \mathbf{x})] = 1.$$

- *Additive robustness.* For any $\mathbf{a} \in \mathbb{F}^k$, $\mathbf{a} \neq 0$, and for any $\mathbf{x} \in \mathbb{F}^n$ it holds that

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(\mathbf{x}) + \mathbf{a}) \notin \mathsf{ERR}] \leq \varepsilon$$

  where ERR is an error message.

Definition 2.17. (*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 2.18. (*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 2.19. (*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 $\mathbf{x} \in \mathbb{F}^n$, $\Pr[\widehat{\mathrm{C}}(\mathbf{x}) = f(\mathbf{x})] = 1$.
- *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 $\mathbf{x} \in \mathbb{F}^n$,

$$SD(\mathrm{C}^{\mathbf{A}}(\mathbf{x}),\ f(\mathbf{x} + a^{\mathrm{In}}) + \mathbf{A}^{\mathrm{Out}}) \leq \varepsilon$$

  where SD denotes statistical distance between two distributions.

Theorem 2.20. ([27], 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* $|\widehat{\mathrm{C}}| = \mathsf{poly}(|\mathrm{C}|, n, \kappa)$*. Moreover, given any additive attack* $\mathbf{A}$ *and input* $\mathbf{x}$*, it is possible to identify* a^{In} *such that* $\widehat{\mathrm{C}}^{\mathbf{A}}(\mathbf{x}) = f(\mathbf{x} + a^{\mathrm{In}}) + \mathbf{A}^{\mathrm{Out}}$.

Remark 2.1. Genkin et al. [27] present a transformation that achieves tighter parameters, namely, better overhead than what is reported in the preceding theorem. We state this theorem in weaker form as it is sufficient for our work.

Remark 2.2. Genkin et al. [27] do not claim the stronger version where the equivalent a^{In} is identifiable. However their transformation directly yields a procedure to identify a^{In}. Namely each bit of the input to the function f needs to be preprocessed via an AMD code before feeding it to $\widehat{\mathrm{C}}$. a^{In} can be computed as $\mathsf{Decode}(\mathbf{x}_{\mathsf{Encode}} + \mathbf{A}_{\mathrm{In}}) - x$ where x_{Encode} is the encoded input $\mathbf{x}$ via the AMD code and $\mathbf{A}_{\mathrm{In}}$ is the additive attack $\mathbf{A}$ restricted to the input wires. In other words, either the equivalent input is $\mathbf{x}$ or the output of $\widehat{\mathrm{C}}$ will be ERR.

2.9. *The [12] Garbling*

An extension of Yao garbled circuits approach [51] for any number of parties n, was introduced by Beaver, Micali and Rogaway in [12], 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 [12] 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 [44] 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 [44] is a modification of the garbling functionality introduced in [12], 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 Eq. 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 input wire w that is associated with party P_i's input 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 $1 \le a, b \le W$ and output wire c, the garbled row $r_1, r_2 \in \{0,1\}$ in gate g is expressed as the concatenation of $\mathbf{R}_{r_1,r_2} = \{R^{g,j}_{g,r_1 r_2}\}_{j=1}^n$, where

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

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 $k^j_{c,1} \oplus k^j_{c,0}$). In this work we consider a functionality that captures an additive error that the adversary can embed into the garbling; see more discussion in Sect. 3.4.

3. Warmup MPC: The Case of Defensible Adversaries

For the sake of gradual introduction of our technical ideas, we begin with a warm-up, we present a protocol and prove security in an easier model, in which the adversary volunteers a "defense" of its actions, consisting of some of its inputs and randomness. Specifically, instead of asking the adversary to prove an action, in this model we just assume that the adversary reveals all its inputs and randomness for that action.

The goal of presenting a protocol in this easier model is to show that it is sufficient to prove correct behavior in some *but not all* of the "OT subprotocols". Later in Sect. 4 we will rely on our non-malleability and zero-knowledge machinery to achieve similar results. Namely, the adversary will be required to prove correct behavior, and we will use rewinding to extract from it the "defense" that our final simulator will need.

3.1. *Defensible Simulation*

We begin by sketching the definition of our "defensible adversary" model. This can be seen as a generalization of *defensible privacy* from the work of Haitner et al. [37], and *semi-malicious security* from the work of Asharov et al. [5]. In both of these notions, corrupted parties are expected to provide a *defense* of their actions, consisting of their inputs and randomness. A defense is valid if it explains the actions of the corrupted party in the protocol execution, and it can be viewed as a "proof of honest behavior." The notions in these two works differ in when the defense is given (at the end in [37] vs. after each message in [5]), and the type of security provided (full security vs. privacy only).

Our notion of *defensible simulation* extends these two notions, in that we do not require that the adversary gives all its coins and inputs. Rather the protocol designer can choose what parts of the adversary state should be included in the defense, and what should be the validity condition. (Indeed, the whole point of presenting this notion is to show that it is enough to require that the adversary proves some of its actions, not all of them.) We also let the protocol designer decide when the defense should be given, rather than always have it at the end of the protocol execution.

Our security guarantee here is simulation security, namely any defensible adversary can be simulated by a simulator that sees the defense (but of course does not get to see the honest parties input). The definition below uses the notations from Appendix A.2.

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 $\mathbf{REAL}$ only if the adversary outputs a valid defense def at the end of the $(r')^{th}$-round and $\widetilde{\mathbf{REAL}} = \perp$

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, and P_1 also sends another bit e_{In}.
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 $(\text{deliver}, e_{\text{Out}})$ from $\mathcal{S}$, the functionality sends $y + e_{\text{Out}}$ to all parties.

Fig. 2. Additively corruptible 3-bit multiplication functionality .

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 [25,27]. Recall from Definition 2.18 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 Sect. 2.8.

3.2. *Step 1: 3-Bit Multiplication with Additive Errors*

The functionality that we realize in this section, $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ is an additively corruptible version of the 3-bit multiplication functionality $\mathcal{F}_{\text{MULT}}$. In addition to the three bits x_1, x_2, x_3, $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ also takes as input an additive "error bit" e_{In} from P_1, and e_{Out} from the adversary, and computes the function $(x_1 x_2 + e_{\text{In}})x_3 + e_{\text{Out}}$. The description of $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$ can be found in Fig. 2.

We recall that our protocol requires an equivocable OT. Namely, given the receiver's randomness, it is possible to explain the OT transcript after the fact, in such a way that the "other sender bit" (the one that the receiver does not get) can be opened both ways. To this end, we rely on an equivocable affine-homomorphic-encryption scheme (Gen, Enc, Dec, AT, Explain) (over $\mathbb{F}_2$) as per Definition 2.6, and an additive secret sharing scheme (Share, Recover) for sharing 0 as per Definition 2.3. The details of our protocol are as follows. We usually assume that randomness is implicit in the encryption scheme, unless specified explicitly. Note that the OT protocol is implicit in our protocol and can be abstracted by a two-message protocol, where the receiver encrypts its input bits and sends it to the sender, that homomorphically evaluates the ciphertext.

Protocol 1. (3-bit Multiplication protocol Π_{DMULT})

Input & Randomness: *Parties P_1, P_2, P_3 are given inputs (x_1, e_{In}), x_2, x_3, respectively. P_1 chooses a random bit s_1 and P_2 chooses two random bits s_2, r_2 (in addition to the randomness needed for the sub-protocols below).*

Notation: *The $\boxminus$ and $\boxdot$ notations denote the respective operations of addition and scalar multiplication in the plaintext group.*

Round 1:

- *Party P_1 runs key generation twice,* $(\mathrm{PK}_a^1, \mathrm{SK}_a^1), (\mathrm{PK}_a^2, \mathrm{SK}_a^2) \leftarrow \mathsf{Gen}$*, encrypts* $\mathsf{C}_\alpha^1[1] := \mathsf{Enc}_{\mathrm{PK}_a^1}(x_1)$ *and* $\mathsf{C}_\alpha^2[1] := \mathsf{Enc}_{\mathrm{PK}_a^2}(x_1)$*, and broadcasts* $((\mathrm{PK}_a^1, \mathsf{C}_\alpha^1[1]), (\mathrm{PK}_a^2, \mathsf{C}_\alpha^2[1]))$ *(to be used by P_2).*
- *P_3 runs key generation four times,* $(\mathrm{PK}_\beta^1, \mathrm{SK}_\beta^1), (\mathrm{PK}_\beta^2, \mathrm{SK}_\beta^2), (\mathrm{PK}_\gamma^1, \mathrm{SK}_\gamma^1), (\mathrm{PK}_\gamma^2, \mathrm{SK}_\gamma^2) \leftarrow \mathsf{Gen}(1^\kappa)$.
  Next it encrypts using the first two keys, $\mathsf{C}_\beta^1[1] := \mathsf{Enc}_{\mathrm{PK}_\beta^1}(x_3)$ *and* $\mathsf{C}_\beta^2[1] := \mathsf{Enc}_{\mathrm{PK}_\beta^2}(x_3)$*, and broadcasts* $\left((\mathrm{PK}_\beta^1, \mathsf{C}_\beta^1[1]), (\mathrm{PK}_\beta^2, \mathsf{C}_\beta^2[1])\right)$ *(to be used by P_2), and* $(\mathrm{PK}_\gamma^1, \mathrm{PK}_\gamma^2)$ *(to be used in round 3 by P_1).*

Round 2:

- *Party P_2 samples x_α^1, x_α^2 such that $x_\alpha^1 + x_\alpha^2 = x_2$ and r_α^1, r_α^2 such that $r_\alpha^1 + r_\alpha^2 = r_2$. It uses affine homomorphism to compute* $\mathsf{C}_\alpha^1[2] := (x_\alpha^1 \boxdot \mathsf{C}_\alpha^1[1]) \boxminus r_\alpha^1$ *and* $\mathsf{C}_\alpha^2[2] := (x_\alpha^2 \boxdot \mathsf{C}_\alpha^2[1]) \boxminus r_\alpha^2$.

Party P_2 also samples r_β^1, r_β^2 such that $r_\beta^1 + r_\beta^2 = r_2$ and s_β^1, s_β^2 such that $s_\beta^1 + s_\beta^2 = s_2$, and uses affine homomorphism to compute $\mathsf{C}_\beta^1[2] := (r_\beta^1 \boxdot \mathsf{C}_\beta^1[1]) \boxminus s_\beta^1$ *and* $\mathsf{C}_\beta^2[2] := (r_\beta^2 \boxdot \mathsf{C}_\beta^2[1]) \boxminus s_\beta^2$*. P_2 broadcasts* $(\mathsf{C}_\alpha^1[2], \mathsf{C}_\alpha^2[2])$ *(to be used by P_1) and* $(\mathsf{C}_\beta^1[2], \mathsf{C}_\beta^2[2])$ *(to be used by P_3).*

- *Party P_3 encrypts* $\mathsf{C}_\gamma^1[1] := \mathsf{Enc}_{\mathrm{PK}_\gamma^1}(x_3)$ *and* $\mathsf{C}_\gamma^2[1] := \mathsf{Enc}_{\mathrm{PK}_\gamma^2}(x_3)$ *and broadcasts* $(\mathsf{C}_\gamma^1[1], \mathsf{C}_\gamma^2[1])$ *(to be used by 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 over a private channel.*

Round 3:

- *Party P_1 computes* $u := \mathsf{Dec}_{\mathrm{SK}_a^1}(\mathsf{C}_\alpha^1[2]) + \mathsf{Dec}_{\mathrm{SK}_a^2}(\mathsf{C}_\alpha^2[2])$ *and $u' = u + e_{\mathrm{In}}$. Then P_1 samples u_γ^1, u_γ^2 such that $u_\gamma^1 + u_\gamma^2 = u'$ and s_γ^1, s_γ^2 such that $s_\gamma^1 + s_\gamma^2 = s_1$. It uses affine homomorphism to compute* $\mathsf{C}_\gamma^1[2] := (u_\gamma^1 \boxdot \mathsf{C}_\gamma^1[1]) \boxminus s_\gamma^1$ *and* $\mathsf{C}_\gamma^2[2] := (u_\gamma^2 \boxdot \mathsf{C}_\gamma^2[1]) \boxminus s_\gamma^2$.

P_1 broadcasts $(\mathsf{C}_\gamma^1[2], \mathsf{C}_\gamma^2[2])$ *(to be used by P_3).*

Defense: *At this point, the adversary broadcasts its "defense:" It gives an input for the protocol, namely $x_\star$. For every "OT protocol instance" where the adversary was the sender (the one sending $\mathsf{C}_\star^\star[2]$), it gives all the inputs and randomness that it used to generate these messages (i.e., the values and randomness used in the affine-homomorphic computation). For instances where it was the receiver, the adversary chooses* one message of each pair *(either $\mathsf{C}_\star^1[1]$ or $\mathsf{C}_\star^2[1]$) and gives the*

inputs and randomness for it (i.e., the plaintext, keys, and encryption randomness). Formally, let trans *be a transcript of the protocol up to and including the* 3^{rd} *round*

$$\mathsf{trans} \stackrel{\text{def}}{=} \left(\begin{array}{l}\mathrm{PK}_a^1, \mathsf{C}_\alpha^1[1], \mathsf{C}_\alpha^1[2], \mathrm{PK}_a^2, \mathsf{C}_\alpha^2[1], \mathsf{C}_\alpha^2[2],\ \mathrm{PK}_\beta^1, \mathsf{C}_\beta^1[1], \mathsf{C}_\beta^1[2], \mathrm{PK}_\beta^2, \mathsf{C}_\beta^2[1], \mathsf{C}_\beta^2[2],\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{PK}_\gamma^1, \mathsf{C}_\gamma^1[1], \mathsf{C}_\gamma^1[2], \mathrm{PK}_\gamma^2, \mathsf{C}_\gamma^2[1], \mathsf{C}_\gamma^2[2]\end{array}\right)$$

$$\mathsf{trans}^\star_{P_1} \stackrel{\text{def}}{=} \left(\mathrm{PK}_a^\star, \mathsf{C}_\alpha^\star[1],\ \mathsf{C}_\gamma^1[2], \mathsf{C}_\gamma^2[2]\right)$$

$$\mathsf{trans}^\star_{P_3} \stackrel{\text{def}}{=} \left(\mathrm{PK}_\beta^\star, \mathsf{C}_\beta^\star[1],\ \mathrm{PK}_\gamma^\star, \mathsf{C}_\gamma^\star[1]\right)$$

we have three NP languages, one per party, with the defense for that party being the witness:

$$\mathcal{L}_{P_1} = \left\{ \mathsf{trans} \;\middle|\; \begin{array}{l}\exists\,(x_1, e_{\mathrm{In}}, \rho_\alpha, \mathrm{SK}_a, \sigma_\alpha, u_\gamma^1, u_\gamma^2, s_\gamma^1, s_\gamma^2)\\ s.t.\ \left(\begin{array}{l}(\mathrm{PK}_a^1, \mathrm{SK}_a = \mathsf{Gen}(\rho_\alpha) \wedge \mathsf{C}_\alpha^1[1] = \mathsf{Enc}_{\mathrm{PK}_a^1}(x_1; \sigma_\alpha))\\ \vee\ (\mathrm{PK}_a^2, \mathrm{SK}_a = \mathsf{Gen}(\rho_\alpha) \wedge \mathsf{C}_\alpha^2[1] = \mathsf{Enc}_{\mathrm{PK}_a^2}(x_1; \sigma_\alpha))\end{array}\right)\\ \quad \wedge\ \mathsf{C}_\gamma^1[2] = u_\gamma^1 \boxdot \mathsf{C}_\gamma^1[1] \boxminus s_\gamma^1\ \wedge\ \mathsf{C}_\gamma^2[2] = u_\gamma^2 \boxdot \mathsf{C}_\gamma^2[1] \boxminus s_\gamma^2\end{array} \right\} \quad (2)$$

$$\mathcal{L}_{P_2} = \left\{ \mathsf{trans} \;\middle|\; \begin{array}{l}\exists\,(x_\alpha^1, x_\alpha^2, s_\beta^1, s_\beta^2, r_\alpha^1, r_\alpha^2, r_\gamma^1, r_\gamma^2)\ \ s.t.\ \ r_\alpha^1 + r_\alpha^2 = r_\gamma^1 + r_\gamma^2\\ \quad \wedge\ \mathsf{C}_\alpha^1[2] = x_\alpha^1 \boxdot \mathsf{C}_\alpha^1[1] \boxminus r_\alpha^1\ \wedge\ \mathsf{C}_\alpha^2[2] = x_\alpha^2 \boxdot \mathsf{C}_\alpha^2[1] \boxminus r_\alpha^2\\ \quad \wedge\ \mathsf{C}_\beta^1[2] = r_\beta^1 \boxdot \mathsf{C}_\beta^1[1] \boxminus s_\beta^1\ \wedge\ \mathsf{C}_\beta^2[2] = r_\beta^2 \boxdot \mathsf{C}_\beta^2[1] \boxminus r_\beta^2\end{array} \right\} \quad (3)$$

$$\mathcal{L}_{P_3} = \left\{ \mathsf{trans} \;\middle|\; \begin{array}{l}\exists\,(x_3, \rho_\beta, \mathrm{SK}_\beta, \sigma_\beta, \rho_\gamma, \mathrm{SK}_\gamma, \sigma_\gamma)\\ s.t.\ \left(\begin{array}{l}(\mathrm{PK}_\beta^1, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}_\beta^1[1] = \mathsf{Enc}_{\mathrm{PK}_\beta^1}(x_3; \sigma_\beta))\\ \vee\ (\mathrm{PK}_\beta^2, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}_\beta^2[1] = \mathsf{Enc}_{\mathrm{PK}_\beta^2}(x_3; \sigma_\beta))\end{array}\right)\\ \quad \wedge \left(\begin{array}{l}(\mathrm{PK}_\gamma^1, \mathrm{SK}_\gamma = \mathsf{Gen}(\rho_\gamma) \wedge \mathsf{C}_\gamma^1[1] = \mathsf{Enc}_{\mathrm{PK}_\gamma^1}(x_3; \sigma_\gamma))\\ \vee\ (\mathrm{PK}_\gamma^2, \mathrm{SK}_\gamma = \mathsf{Gen}(\rho_\gamma) \wedge \mathsf{C}_\gamma^2[1] = \mathsf{Enc}_{\mathrm{PK}_\gamma^2}(x_3; \sigma_\gamma))\end{array}\right)\end{array} \right\} \quad (4)$$

ROUND 4:

- P_3 *computes* $v := \mathsf{Dec}_{\mathrm{SK}_\beta^1}(\mathsf{C}_\beta^1[2]) + \mathsf{Dec}_{\mathrm{SK}_\beta^2}(\mathsf{C}_\beta^2[2])$, $w := \mathsf{Dec}_{\mathrm{SK}_\gamma^1}(\mathsf{C}_\gamma^1[2]) + \mathsf{Dec}_{\mathrm{SK}_\gamma^2}(\mathsf{C}_\gamma^2[2])$, *and* $s_3 := v + w$.
- *Every party* P_j *adds the zero shares to* s_j, *broadcasting* $S_j := s_j + \sum_{i=1}^3 z_i^j$.

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

Lemma 3.1. *Assuming either LWE, DDH, QR or DCR, protocol* Π_{DMULT} *securely realizes functionality* $\mathcal{F}_{\mathrm{MULT}}^{\mathbf{A}}$ *(cf. Fig. 2) in the presence of a "defensible adversary" that always broadcasts a valid defense at the end of the third round and actively corrupts at most two parties. The protocol uses private channels after the first round and a broadcast channel in all rounds.*

Proof. We first show that the protocol is correct with a benign adversary. Observe that $u' = e_{\mathrm{In}} + x_1(x_\alpha^1 + x_\alpha^2) - (r_\alpha^1 + r_\alpha^2) = e_{\mathrm{In}} + x_1x_2 - r_2$, and similarly $v = x_3r_2 - s_2$ and

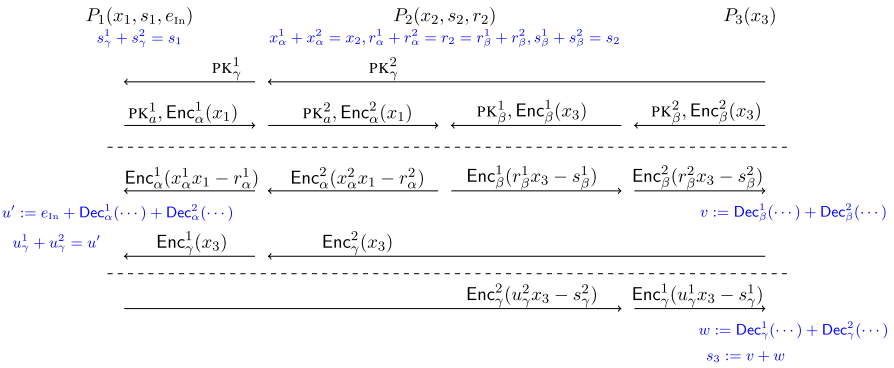

Fig. 3. Round 1, 2 and 3 of Π_{DMULT} protocol. In the fourth round each party P_i adds the zero shares to s_j and broadcasts the result .

$w = x_3 u' - s_1$. Therefore,

$$\begin{aligned} S_1 + S_2 + S_3 &= s_1 + s_2 + s_3 = s_1 + s_2 + (v + w) \\ &= s_1 + s_2 + (x_3 r_2 - s_2) + (x_3 u' - s_1) \\ &= x_3 r_2 + x_3(x_1 x_2 - r_2 + e_{\text{In}}) \;=\; (x_1 x_2 + e_{\text{In}}) x_3 \end{aligned}$$

as required. We continue with the security proof.

To argue security we need to describe a simulator and prove that the simulated view is indistinguishable from the real one. Below fix inputs x_1, e_{In}, x_2, x_3, and a defensible PPT adversary $\mathcal{A}$ controlling a fixed subset of parties $I \subseteq [3]$ (and also an auxiliary input z).

The simulator $\mathcal{S}$ chooses arbitrary inputs for each honest party (denote these values by $\widehat{x_i}$), and then follows the honest protocol execution using these inputs until the end of the 3^{rd} round. Upon receiving a valid "defense" that includes the inputs and randomness that the adversary used to generate (some of) the messages $\mathsf{C}_\star^i[j]$, the simulator extracts from that defense the effective inputs of the adversary to send to the functionality, and other values to help with the rest of the simulation. Specifically:

- If P_3 is corrupted then its defense (for one of the $\mathsf{C}_\beta^i[1]$'s and one of the $\mathsf{C}_\gamma^i[1]$'s) includes a value for x_3, that we denote x_3^*. (A defensible adversary is guaranteed to use the same value in the defense for $\mathsf{C}_\beta^\star[1]$ and in the defense for $\mathsf{C}_\gamma^\star[1]$'s.)
- If P_2 is corrupted then the defense that it provides includes all of its inputs and randomness (since it always plays the "OT sender"), hence the simulator learns a value for x_2 that we denote x_2^*, and also some values r_2, s_2. (If P_2 is honest then by r_2, s_2 we denote below the values that the simulator chose for it.)
- If P_1 is corrupted then its defense (for either of the $\mathsf{C}_\alpha^i[1]$'s) includes a value for x_1 that we denote x_1^*.

From the defense for both $\mathsf{C}_\gamma^1[2]$, $\mathsf{C}_\gamma^2[2]$ the simulator learns the u_i^γ's and s_i^γ's, and it sets $u' := u_\gamma^1 + u_\gamma^2$ and $s_1 := s_\gamma^1 + s_\gamma^2$.

The simulator sets $u := x_1^* x_2^* - r_2$ if P_2 is corrupted and $u := x_1^* \widehat{x_2} - r_2$ if P_2 is honest, and then computes the effective value $e_{\text{In}}^* := u' - u$. (If P_1 is honest then by s_1, u, u' we denote below the values that the simulator used for it.)

Let x_i^* and e_{In}^* be the values received by the functionality. (These are computed as above if the corresponding party is corrupted, and are equal to x_i, e_{In} if it is honest.) The simulator gets back from the functionality the answer $y = (x_1^* x_2^* + e_{\text{In}}^*) x_3^*$.

Having values for s_1, s_2 as described above, the simulator computes $s_3 := y - s_1 - s_2$ if P_3 is honest, and if P_3 is corrupted then the simulator sets $v := r_2 x_3^* - s_2$, $w := u x_3^* - s_1$ and $s_3 := v + w$. It then proceeds to compute the values S_j that the honest parties broadcast in the last round.

Let s be the sum of the s_i values for all the corrupted parties, and let z be the sum of the zero-shares that the simulator sent to the adversary (on behalf of all the honest parties), and z' be the sum of zero-shared that the simulator received from the adversary. The values that the simulator broadcasts for the honest parties in the fourth round are chosen at random, subject to them summing up to $y - (s + z - z')$.

If the adversary sends its fourth round messages, an additive output error is computed as $e_{\text{Out}} := y - \sum_j \tilde{S}_j$ where $\tilde{S}_j$ are the values that were broadcast in the fourth round. The simulator finally sends $(\mathsf{deliver}, e_{\text{Out}})$ to the ideal functionality.

This concludes the description of the simulator, it remains to prove indistinguishability. Namely, we need to show that for the simulator $\mathcal{S}$ above, the distributions $\widetilde{\mathbf{REAL}}_{\Pi_{\text{DMULT}}, \mathcal{A}(z), I}(\kappa, (x_1, e_{\text{In}}), x_2, x_3)$ and $\mathbf{IDEAL}_{\mathcal{F}_{\text{MULT}}^{\mathbf{A}}, \mathcal{S}(z), I}(\kappa, (x_1, e_{\text{In}}), x_2, x_3)$ are indistinguishable. We proceed in several hybrid games, beginning from H_0 that has view identical to the real game, and ending at H_8 whose view is distributed identically to the simulation.

Roughly, in the hybrid H_3 below we modify the messages S_i that are broadcast in the last round, in the hybrids H_4, H_6, H_7, we modify P_3, P_1, P_2, respectively, to use fake inputs instead of their real inputs, and finally in H_8 we modify the S_i's again to use the output from the functionality $\mathcal{F}_{\text{MULT}}^{\mathbf{A}}$.

Hybrid H_0. This experiment is the execution in the real world. As usual, we postulate a centralized "challenger" that plays the role of the honest parties in H_0, namely it gets all their inputs and just follows the protocol on their behalf (and ignores the "defense" provided by the adversary).

Hybrid H_1. In the next few hybrids we make some changes to the internal computations of the challenger without affecting what the adversary sees. In this hybrid, if P_1 is honest then instead of choosing s_1 at random and then choosing random $s_\gamma^{1,2}$ that add up to s_1, the challenger chooses both $s_\gamma^{1,2}$ uniformly at random and sets $s_1 := s_\gamma^1 + s_\gamma^2$. Similarly, if P_2 is honest then the challenger chooses at random $s_\beta^{1,2}$, $r_\alpha^{1,2}$, and r_β^1, and sets $s_2 := s_\beta^1 + s_\beta^2$, $r_2 := r_\alpha^1 + r_\alpha^2$, and $r_\beta^2 := r_2 - r_\beta^1$.

Also, the challenger in this hybrid no longer ignores the adversary's defense, instead it uses it to compute some local variables (which are then ignored). Specifically, regardless of which party is corrupted, the challenger knows:

- A value for x_1 (either since P_1 is honest, or from the defense of one of the $\mathsf{C}_\alpha^{1,2}[1]$'s).
- The $u_\gamma^{1,2}$ and $s_\gamma^{1,2}$'s (either since P_1 is honest, or from the defense of both the $\mathsf{C}_\gamma^{1,2}[2]$'s).

- The $x_\beta^{1,2}, r_\beta^{1,2}, r_\gamma^{1,2}$, and, $s_\beta^{1,2}$'s (either since P_2 is honest, or from the defense of all the $\mathsf{C}_{\alpha,\beta}^{1,2}[2]$'s).
- A value for x_3 (either P_3 is honest, or from the defense of one of each of the $\mathsf{C}_\beta^{1,2}[1]$'s and $\mathsf{C}_\gamma^{1,2}[1]$'s).

Below we denote $s_1 := s_\gamma^1 + s_\gamma^2$ and $u' = u_\gamma^1 + u_\gamma^2$, whether or not P_1 is honest. Similarly, we denote $r_2 := r_\beta^1 + r_\beta^2 = r_\gamma^1 + r_\gamma^2$ and $s_2 := s_\beta^1 + s_\beta^2$, whether or not P_2 is honest.

The challenger in H_1 also chooses random "fake input bits" $\widehat{x_i}$ for the parties (in addition to the real inputs x_i). These inputs are never used in H_1, but will be used in some of the hybrids below.

Hybrid H_2. In this hybrid, the challenger forgoes decrypting the ciphertexts $\mathsf{C}_{\beta,\gamma}^{1,2}[2]$ even if P_3 is honest. Whether or not P_3 is honest, at the end of the 3^{rd} round the challenger uses the values above that it knows to set $v := r_2 x_3 - s_2$, $w_i := u_\gamma^i x_3 - s_\gamma^i$ $(i = 1, 2)$, and $w := w_1 + w_2$.

When P_3 is honest, $\mathsf{C}_{\beta,\gamma}^{1,2}[1]$ are all valid ciphertexts. Therefore so are $\mathsf{C}_{\beta,\gamma}^{1,2}[2]$, since they all have a valid defense. Moreover the valid defense implies that $r_\beta^{1,2}$, $u_\gamma^{1,2}$ and $s_{\beta,\gamma}^{1,2}$ are consistent with the plaintext values inside these ciphertexts. Hence the computed values v, w are identical to what was computed in H_1 (in the case that P_3 is honest).

Hybrid H_3. This hybrid changes the computation of the s_i's:

- If P_1 is honest then the challenger changes the way it computes s_1: Rather than $s_1 := s_\gamma^1 + s_\gamma^2$, the challenger waits until after the 3^{rd} round, then sets $s_1 := (x_1 x_2 + e_{\text{In}}) x_3 - s_2 - v - w$ (and broadcasts $S_1 = s_1 + \sum_j z_j^1$ on behalf of P_1). It is easy to check that s_1 in this hybrid is the same as in H_2: As $u' = u_\gamma^1 + u_\gamma^2 = x_1 x_2 + e_{\text{In}} - r_2$, then

$$\begin{aligned}(x_1 x_2 + e_{\text{In}}) x_3 - s_2 - v - w \;&=\; (x_1 x_2 + e_{\text{In}}) x_3 - s_2 - (r_2 x_3 - s_2) \\ &\quad -\big(u' x_3 - (s_\gamma^1 + s_\gamma^2)\big) \\ &= (x_1 x_2 + e_{\text{In}}) x_3 - r_2 x_3 - (x_1 x_2 + e_{\text{In}} - r_2) x_3 + s_\gamma^1 + s_\gamma^2 \\ &= s_\gamma^1 + s_\gamma^2.\end{aligned}$$

- If P_1 is corrupted and P_2 is honest, then the challenger changes the way it computes s_2: Rather than $s_2 := s_\beta^1 + s_\beta^2$, the challenger waits until after the 3^{rd} round, then sets $s_2 := (x_1 x_2 + e_{\text{In}}) x_3 - s_1 - v - w$ (and broadcasts $S_2 = s_2 + \sum_j z_j^2$ on behalf of P_2). The same argument as above implies that s_2 in this hybrid is the same as in H_2.
- If P_1 and P_2 are corrupted and P_3 is honest, the challenger changes the way it computes s_3: Rather than $s_3 := v + w$, the challenger sets after the 3^{rd} round $s_3 := (x_1 x_2 + e_{\text{In}}) x_3 - s_1 - s_2$ and broadcasts $S_3 = s_3 + \sum_j z_j^3$ on behalf of P_3. As in the other two cases, here too s_3 is the same as in H_2.

Hybrid H_4. In this hybrid, if P_3 is honest then the challenger encrypts the fake $\widehat{x_3}$ rather than the real x_3 in all the ciphertexts $\mathsf{C}_{\beta,\gamma}^{1,2}[1]$. The rest of the execution remains unchanged (including using the real x_3 in the computation of v, w, s_3 as above).

Recall that the challenger no longer decrypts any of the ciphertexts $\mathsf{C}^{1,2}_{\beta,\gamma}[2]$ and hence no longer uses the secret keys $\mathrm{sk}^{1,2}_{\beta}$, $\mathrm{sk}^{1,2}_{\gamma}$. We can therefore reduce indistinguishability between H_3 and H_4 to the semantic security of the encryption. The reduction algorithm plays the challenger, but instead of generating the keys itself and doing the encryption, it receives the public keys and ciphertexts from the CPA-security game, where the $\mathsf{C}^{1,2}_{\beta,\gamma}[1]$'s encrypt either x_3 or $\widehat{x}_3$. Hence we have:

Claim 3.2. *Assuming semantic security of* Enc*, the adversary's view in* H_4 *is indistinguishable from* H_3.

Hybrid H_5. If P_1 is honest, the challenger changes the way it chooses the $u^{1,2}_{\gamma}$'s: Instead of choosing them at random subject to $u^1_\gamma + u^2_\gamma = u'$, they are chosen uniformly and independently.

We note that in this hybrid the challenger no longer needs to compute u and u', and therefore it no longer needs to decrypt the ciphertexts $\mathsf{C}^1_\alpha[2]$, $\mathsf{C}^2_\alpha[2]$.

Claim 3.3. *If* Enc *satisfies the equivocation property from Definition 2.6, then the adversary's view in hybrid* H_5 *is statistically close to the hybrid* H_4.

Proof. Below we prove Claim 3.3 for the case where $\mathsf{C}^1_\gamma[1]$ is a valid ciphertext, encrypting the x_3 that the challenger knows. The proof for the case where $\mathsf{C}^2_\gamma[1]$ is a valid encryption of x_3 is symmetric, and we know that at least one of these cases hold since the adversary is defensible.

Below we fix all the inputs and randomness of all the parties *except the choices of* u^1_γ *and* s^1_γ and whatever randomness is involved in computing $\mathsf{C}^1_\gamma[2] := (u^1_\gamma \boxdot \mathsf{C}^1_\gamma[1]) \boxminus s^1_\gamma$. We show that when $\mathsf{C}^1_\gamma[1]$ is a valid encryption of x_3, the adversary's views in the two hybrids are statistically close, even conditioned on all these fixed values.

We note that the fixed values include in particular u^2_γ, and s^2_γ, and conditioned on all those values the adversary's view is uniquely determined by $\mathsf{C}^1_\gamma[2]$ and s_1. It is therefore sufficient to show that the residual distributions on $(\mathsf{C}^1_\gamma[2], s_1)$ are close between these hybrids. We next show that both s_1 and $\mathsf{C}^1_\gamma[2]$ depend only on the single value $w_1 := u^1_\gamma \cdot x_3 - s^1_\gamma$.

For s_1, since everything except u^1_γ and s^1_γ is fixed, then in particular $\Delta := (x_1x_2 + e_{\mathrm{In}})x_3 - s_2 - v - w_2$ is fixed, so $s_1 := \Delta - w_1$ is uniquely determined by w_1. For $\mathsf{C}^1_\gamma[2]$, the equivocation property of the encryption scheme implies in particular that for any valid $\mathsf{C}^1_\gamma[1] = \mathsf{Enc}(x_3)$, the distribution of $\mathsf{C}^1_\gamma[2] := u^1_\gamma \boxdot \mathsf{C}^1_\gamma[1] \boxminus s^1_\gamma$ depends (up to negligible difference) on just the value encrypted in it, namely by $w_1 := u^1_\gamma \cdot x_3 - s^1_\gamma$.[7]

We conclude that it is sufficient to show that w_1 has the same distribution in both hybrids. But this is obvious, as in both hybrids it is set as $w_1 := u^1_\gamma \cdot x_3 - s^1_\gamma$ for a uniformly random s^1_γ, and only the choice of u^1_γ differs between them. □

[7] This is where we use the assumption about the validity of $\mathsf{C}^1_\gamma[1] = \mathsf{Enc}(x_3)$, for the same value x_3 that the challenger use to compute $w_1 := u^1_\gamma \cdot x_3 + s^1_\gamma$.

Hybrid H_6. If P_1 is honest, the challenger encrypts the fake $\widehat{x_1}$ rather than the real x_1 in the ciphertexts $\mathsf{C}_\alpha^{1,2}[1]$. The rest of the execution remains unchanged (including using the real x_1 in the computation of s_1). As the challenger no longer decrypts the ciphertexts $\mathsf{C}_\alpha^{1,2}[2]$ and hence no longer uses the secret keys $\mathsf{sk}_a^{1,2}$, we can reduce indistinguishability between H_5 and H_6 to the semantic security of the encryption.

Claim 3.4. *Assuming semantic security of* Enc*, the adversary's view in* H_6 *is indistinguishable from* H_5.

Hybrid H_7. If P_2 is honest, the challenger uses the fake $\widehat{x_2}$ rather than the real x_2 in the homomorphic computation of the $\mathsf{C}_\alpha^{\mathsf{i}}[2]$'s. More specifically, instead of choosing $x_\beta^{1,2}$ subject to $x_\beta^1 + x_\beta^2 = x_2$ the challenger choose random and independent $x_\beta^{1,2}$ (and sets $\widehat{x_2} := x_\beta^1 + x_\beta^2$).

As before, statistical closeness between H_6 and H_7 follows from the equivocation property of the encryption scheme, the argument is similar to (but slightly more complicated than) Claim 3.3: We begin with a sub-hybrid in which $x_\beta^{1,2}$ still add up to x_2 but r_β^2 is chosen independently of the other $r_\star^\star$'s (rather than setting $r_\beta^2 := r_2 - r_\beta^1$). Statistical closeness of this sub-hybrid is almost identical to Claim 3.3. Then we switch the $x_\beta^{1,2}$, and repeat a similar argument.

Claim 3.5. *If* Enc *satisfies the equivocation property from Definition 2.6, then the adversary's view in hybrid* H_7 *is statistically close to the hybrid* H_6.

Hybrid H_8. In this hybrid, the challenger is given access to the functionality $\mathcal{F}_{\mathrm{MULT}}^{\mathbf{A}}$, that gets the honest parties' inputs. The challenger extracts as before the inputs of the corrupted parties from the "defense" and gives them to the functionality after the third round, getting back the output $y = (x_1 x_2 + e_{\mathrm{ln}})x_3$.

The challenger computes the s_i values for the corrupted parties, let s be their sum, and also let z be the sum of the zero-shares that it sent to the adversary (on behalf of all the honest parties) and z' be the sum of zero-shared that it received from the adversary. The challenger broadcasts on behalf of the honest parties messages S_i chosen at random, subject to them summing up to $y - (s + z - z')$.

We argue that the distribution of the S_i values for the honest parties is identical between to the previous hybrid. To see this, we first note that as long as there are any honest parties, in both hybrids we have $\sum_{i \notin I} S_i + \sum_{i \in I} s_i = y - z + z'$ (where I is the set of corrupted parties). In H_8 the S_i values for the honest parties are chosen at random subject to that constraint, so it is enough to show that also in H_7 these values are random subject to that constraint.

This is clearly true when there is just one honest party (since in that case there is only one value for the honest S_i that satisfies the constraint). When there are more honest parties, then their S_i's are randomized by the zero shares z_j^i, rendering them random subject to their sum.

Epilogue. To complete the proof, we note that in H_8 the challenger does not use at all the input of the honest parties, and indeed the view of that hybrid is identical to the simulated game with the simulator $\mathcal{S}$. □

3.2.1. *Between Defensible and Real Security*

In Sect. 4 below we show how to augment the protocol above to provide security against general adversaries, not just defensible ones, by adding proofs of correct behavior and using rewinding for extraction.

There is, however, one difference between having a defensible adversary and having a general adversary that proves correct behavior: Having a proof in the protocol cannot ensure correct behavior, it only ensures that deviation from the protocol will be detected (since the adversary cannot complete the proof). So we still must worry about the deviation causing information to be leaked to the adversary before it is caught.

Specifically for the protocol above, in Claims 3.3 and 3.5 we relied on at least one in each pair of ciphertexts being valid. Indeed for an invalid ciphertext C, it could be the case that $C' := (u \boxdot C) \boxplus s$ reveals both u and s. If that was the case, then (for example) a corrupt P_1 could send invalid ciphertexts $\mathsf{C}_\alpha^{1,2}[1]$ to P_2, then learn both $x_\alpha^{1,2}$ (and hence x_2) from P_2's reply.

One way of addressing this concern would be to rely on an actively secure encryption (as defined in [47]), but this is a strong requirement, much harder to realize than our Definition 2.6. Instead, in our BMR-based protocol we ensure that all the inputs to the multiplication gates are just random bits, and have parties broadcast their real inputs masked by these random bits later in the protocol. We then use ZAP proofs of correct ciphertexts *before the parties broadcast their masked real inputs*. Hence, an adversary that sends two invalid ciphertexts can indeed learn the input of (say) P_2 in the multiplication protocol, but this is just a random bit, and P_2 will abort before outputting anything related to its real input in the big protocol. For that, we consider the following two NP languages:

$$\mathcal{L}'_{P_1} = \left\{ \mathsf{trans}_2 \;\middle|\; \begin{array}{l} \exists\,(x_1, \rho_\alpha, \mathrm{SK}_a, \sigma_\alpha) \\ s.t. \left(\begin{array}{l} (\mathrm{PK}_a^1, \mathrm{SK}_a = \mathsf{Gen}(\rho_\alpha) \wedge \mathsf{C}_\alpha^1[1] = \mathsf{Enc}_{\mathrm{PK}_a^1}(x_1; \sigma_\alpha)) \\ \vee\ (\mathrm{PK}_a^2, \mathrm{SK}_a = \mathsf{Gen}(\rho_\alpha) \wedge \mathsf{C}_\alpha^2[1] = \mathsf{Enc}_{\mathrm{PK}_a^2}(x_1; \sigma_\alpha)) \end{array} \right) \end{array} \right\}$$

$$\mathcal{L}'_{P_3} = \left\{ \mathsf{trans}_2 \;\middle|\; \begin{array}{l} \exists\,(x_3, \rho_\beta, \mathrm{SK}_\beta, \sigma_\beta, \rho_\gamma, \mathrm{SK}_\gamma) \\ s.t. \left(\begin{array}{l} (\mathrm{PK}_\beta^1, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}_\beta^1[1] = \mathsf{Enc}_{\mathrm{PK}_\beta^1}(x_3; \sigma_\beta)) \\ \vee\ (\mathrm{PK}_\beta^2, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}_\beta^2[1] = \mathsf{Enc}_{\mathrm{PK}_\beta^2}(x_3; \sigma_\beta)) \\ \quad \wedge \left((\mathrm{PK}_\gamma^1, \mathrm{SK}_\gamma = \mathsf{Gen}(\rho_\gamma)) \right) \end{array} \right) \end{array} \right\}$$

where trans_2 is a transcript of the protocol up to and including the 2^{rd} round. Note that P_2 does not generate any public keys and thus need not prove anything.

3.3. *Step 2: Arbitrary Degree-3 Polynomials*

The protocol Π_{DMULT} from above can be directly used to securely compute any degree-3 polynomial for any number of parties in this "defensible" model, roughly by just expressing the polynomial as a sum of degree-3 monomials and running Π_{DMULT} to compute each one, with some added shares of zero so that only the sum is revealed.

Namely, party P_i chooses an n-of-n additive sharing of zero $\mathbf{z}_i = (z_i^1, \ldots, z_j^n) \leftarrow \mathsf{Share}(0, n)$, and sends z_i^j to party j. Then the parties run one instance of the protocol Π_{DMULT} for each monomial, up to the end of the third round. Let $s_{i,m}$ be the value that P_i would have computed in the m^{th} instance of Π_{DMULT} (where $s_{i,m} := 0$ if P_i's is not a party that participates in the protocol for computing the m^{th} monomial). Then P_i only broadcasts the single value

$$S_i = \sum_{m \in [M]} s_{i,m} + \sum_{j \in [n]} z_j^i.$$

To compute multiple degree-3 polynomials on the same input bits, the parties just repeat the same protocol for each output bit (of course using an independent sharing of zero for each output bit).

In terms of security, we add the requirement that a valid "defense" for the adversary is not only valid for each instance of Π_{DMULT} separately, but all these "defenses" are consistent: If some input bit is a part of multiple monomials (possibly in different polynomials), then we require that the same value for that bit is used in all the corresponding instances of Π_{DMULT}. We denote this modified protocol by Π_{DPOLY} and note that the proof of security is exactly the same as the proof in the previous section.

3.4. *Step 3: Arbitrary Functionalities*

We recall that from the works of [12,20,44] 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 can be reduced to computing many degree-3 polynomials. However, our protocol for realizing degree-3 polynomials from above lets the adversary introduce additive errors (cf. Functionality $\mathcal{F}_{\text{MULT}}^{\mathbf{A}}$), so we rely on a pre-processing step to make the BMR functionality resilient to such additive attacks. We will immunize the circuit to these attacks by relying on the following primitives and tools:

Information theoretic MAC $\{\mathsf{MAC}_k\}_k$: This will be required to protect the output translation tables from being manipulated by a rushing adversary. Namely, each party contributes a MAC key where the output of the function is authenticated under each of the parties' keys. The idea here is that an adversary cannot simply change the output without forging the authenticated values.

AMD codes (Definition 2.16): This will be required to protect the inputs and outputs of the computation from an additive attack by the adversary. Namely, each party encodes its input using an AMD code. The original computed circuit is then modified so that it first decodes these encoded inputs, then runs the original computation and finally, encodes the outcome.

Additive attack resilient circuits (i.e. AMD circuits, Section 2.8): This will be required to protect the computation of the internal wire values from an additive attack by the adversary. Recall from Sect. 3.2 that the adversary may introduce additive errors to the computed polynomials whenever corrupting party P_1. To combat with such errors we only evaluate circuits that are resilient to additive attacks.

Family of pairwise independent hash functions: We will need this to mask the key values. The parties broadcast all keys in a masked format, namely, $h, h(T) \oplus k$ for a random string T and key k. Then, when decrypting a garbled row, only T is revealed. T and h can be combined with the broadcast message to reveal k.

Next we explain how to embed these tools in the BMR garbling computation. Let $f(\hat{x}_1, \ldots, \hat{x}_n)$ be an n-party function that the parties want to compute securely. At the onset of the protocol, the parties locally apply the following transformation to the function f and their inputs:

1. Define

$$f_1\big((\hat{x}_1, \alpha_1), \ldots, (\hat{x}_n, \alpha_n)\big) = \big(f(\mathbf{x}), \mathsf{MAC}_{\alpha_1}(f(\mathbf{x})), \ldots, \mathsf{MAC}_{\alpha_n}(f(\mathbf{x}))\big)$$

   where $\mathbf{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ are the parties' inputs.

The MAC verification is meant to detect adversarial modifications to output wires (since our basic model allows arbitrary manipulation to the output wires).

2. Let $(\mathsf{Encode}, \mathsf{Decode})$ be the encoding and decoding functions for an AMD code, and define

$$\mathsf{Encode}'(\hat{x}_1, \ldots, \hat{x}_n) = (\mathsf{Encode}(\hat{x}_1), \ldots, \mathsf{Encode}(\hat{x}_n))$$

   and

$$\mathsf{Decode}'(y_1, \ldots, y_n) = (\mathsf{Decode}(y_1), \ldots, \mathsf{Decode}(y_n)).$$

   Then define a modified function

$$f_2(\mathbf{x}) = \mathsf{Encode}'(f_1(\mathsf{Decode}'(\mathbf{x}))).$$

   Let C be a Boolean circuit that computes f_2.
3. Next we apply the transformations of Genkin et al. [25,27] to circuit C to obtain $\widehat{\mathrm{C}}$ that is resilient to additive attacks on its internal wire values.
4. We denote by $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}((x_1, R_1), ..., (x_n, R_n))$ our modified BMR randomized encoding of circuit $\widehat{\mathrm{C}}$ with inputs x_i and randomness R_i, as described below. We denote by $\mathsf{BMR.Decode}$ the corresponding decoding function for the randomized encoding, where, for all i, we have

$$\mathsf{BMR.Decode}(\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}((x_1, R_1), ..., (x_n, R_n)), R_i) = \widehat{\mathrm{C}}(x_1, \ldots, x_n).$$

In the protocol for computing f, each honest party P_i with input $\hat{x}_i$ begins by locally encoding its input via an AMD code, $x_i := \mathsf{Encode}(\hat{x}_i; \$)$ (where $\$$ is some fresh randomness). P_i then engages in a protocol for evaluating the circuit $\widehat{\mathrm{C}}$ (as defined below), with local input x_i and a randomly chosen MAC key α_i. Upon receiving an output y_i from the protocol (which is supposed to be AMD encoded, as per the definition of f_2 above), P_i decodes and parses it to get $y_i' := \mathsf{Decode}(y_i) = (z, t_1, \ldots, t_n)$. Finally

P_i checks whether $t_i = \mathsf{MAC}_{\alpha_i}(z)$, outputting z if the verification succeeds, and $\perp$ otherwise.

A modified BMR encoding. We describe the modified BMR encoding for a general circuit D with n inputs $x_1, \ldots, x_n$. Without loss of generality, we assume D is a Boolean circuit comprising only of fan-in two NAND gates. Let W be the total number of wires and G the total number of gates in the circuit D. Let $F = \{\mathsf{F}_k : \{0,1\}^\kappa \to \{0,1\}^{4\kappa}\}_{k\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be a family of PRFs.

The encoding procedure takes the inputs $x_1, \ldots, x_n$ and additional random inputs $R_1, \ldots, R_n$. Each R_j comprises of PRF keys, key masks and hash functions from pairwise independent family for every wire. More precisely, R_j ($j \in [n]$) can be expressed as $\{\lambda_w^j, k_{w,0}^j, k_{w,0}^j, T_{w,0}^j, T_{w,1}^j, h_{w,0}^j, h_{w,1}^j\}_{w\in[W]}$ where λ_w^j are bits, $k_{w,b}^j$ are κ bit PRF keys, $T_{w,b}^j$ are 4κ bits key masks, and $h_{w,b}^j$ are hash functions from a pairwise independent family from 4κ to κ bits.

The encoding procedure $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$ on input $((x_1, R_1), ..., (x_n, R_n))$ outputs

$$\left\{\begin{array}{ll}(R_{00}^{g,j}, R_{01}^{g,j}, R_{10}^{g,j}, R_{11}^{g,j})_{g\in[G],j\in[n],r_1,r_2\in\{0,1\}} & \texttt{//GarbledTables}\\ (h_{w,b}^j, \Gamma_{w,b}^j)_{w\in[W],j\in[n],b\in\{0,1\}}, & \texttt{//maskedkeyvalues}\\ (\Lambda_w, k_{w,\Lambda_w}^1, \ldots, k_{w,\Lambda_w}^n)_{w\in\mathsf{Inp}}, & \texttt{//keysandmasksforinputwires}\\ (\lambda_w)_{w\in\mathsf{Out}} & \texttt{//Outputtranslationtable}\end{array}\right\}$$

where

$$\begin{aligned}
R_{r_1,r_2}^{g,j} &= \Big(\bigoplus_{i=1}^{n}\mathsf{F}_{k_{a,r_1}^i}(g,j,r_1,r_2)\Big)\oplus\Big(\bigoplus_{i=1}^{n}\mathsf{F}_{k_{b,r_2}^i}(g,j,r_1,r_2)\Big)\oplus S_{r_1,r_2}^{g,j}\\
S_{r_1,r_2}^{g,j} &= T_{c,0}^j \oplus \chi_{r_1,r_2}\cdot(T_{c,1}^j \oplus T_{c,0}^j)\\
\chi_{r_1,r_2} &= \mathsf{NAND}\big(\lambda_a\oplus r_1, \lambda_b\oplus r_2\big)\oplus\lambda_c \;=\; [(\lambda_a\oplus r_1)\cdot(\lambda_b\oplus r_2)\oplus 1]\oplus\lambda_c\\
\Gamma_{w,b}^j &= h_{w,b}^j(T_{w,b}^j)\oplus k_{w,b}^j\\
\lambda_w &= \begin{cases}\lambda_w^{j_w} & \text{if } w\in\mathsf{Inp} \quad \texttt{//inputwire}\\ \lambda_w^1\oplus\cdots\oplus\lambda_w^n & \text{if } w\in[W]/\mathsf{Inp} \quad \texttt{//internalwire}\end{cases}\\
\Lambda_w &= \lambda_w\oplus x_w \text{ for all } w\in\mathsf{Inp} \qquad \texttt{//maskedinputbit}
\end{aligned}$$

and wires a, b and $c \in [W]$ denote the input and output wires, respectively, for gate $g \in [G]$. $\mathsf{Inp} \subseteq [W]$ denotes the set of input wires to the circuit, $j_w \in [n]$ denotes the party whose input flows the wire w and x_w the corresponding input. $\mathsf{Out} \subseteq [W]$ denotes the set of output wires.

We remark that the main difference with standard BMR encoding is that when decrypting a garbled row, a value $T_{\star,\star}^\star$ is revealed and the key is obtained by unmasking the corresponding $h_{\star,\star}^\star, h_{\star,\star}^\star(T_{\star,\star}^\star)\oplus k_{\star,\star}^\star$ value that is part of the encoding. This additional level of indirection of receiving the mask T and then unmasking the key is required to tackle errors to individual bits of the plaintext encrypted in each garbled row.

The decoding procedure basically corresponds to the evaluation of the garbled circuit. More formally, the decoding procedure $\mathsf{BMR.Decode}$ is defined iteratively gate by gate according to some standard (arbitrary) topological ordering of the gates. In particular,

given an encoding information k^j_{w,Λ_w} for every input wire w and $j \in [n]$, of some input x, then for each gate g with input wires a and b and output wire c compute

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

Let Λ_c denote the bit for which $T^j_c = T^j_{c,\Lambda_c}$ and define $k^j_c = \Gamma^j_{c,\Lambda_c} \oplus h^j_{c,\Lambda_c}(T^j_c)$. Finally given Λ_w for every output wire w, compute the output carried in wire w as $\Lambda_w \oplus \left(\bigoplus_{j=1}^{n} \lambda^j_w\right)$.

Securely computing BMR.Encode **using** Π_{DPOLY}. We decompose the computation of BMR.Encode into an offline and online phase. The offline part of the computation will only involve computing the "plaintexts" in each garbled row, i.e. $S^{\star,\star}_{\star,\star}$ values and visible masks Λ_w values for input wires. More precisely, the parties compute

$$\{(S^{g,j}_{00}, S^{g,j}_{01}, S^{g,j}_{10}, S^{g,j}_{11})_{g\in[G], j\in[n], r_1,r_2\in\{0,1\}}, (\Lambda_w)_{w\in\mathsf{Inp}}\}.$$

Observe that the $S^{\star,\star}_{\star,\star}$ values are all degree-3 computations over the randomness $R_1, \ldots, R_n$ and therefore can be computed using Π_{DPOLY}. Since the Λ_w values for the input wires depend only on the inputs and internal randomness of party P_{j_w}, the Λ_w value can be broadcast by that party P_{j_w}. The offline phase comprises of executing all instances of Π_{DPOLY} in parallel in the first three rounds. Additionally, the Λ_w values are broadcast in the third round. At the end of the offline phase, in addition to the Λ_w values for the input wires, the parties obtain XOR shares of the $S^{\star,\star}_{\star,\star}$ values.

In the online phase which is carried out in rounds 3 and 4, each party P_j broadcasts the following values:

- $\widetilde{R}^{\star,j}_{\star,\star}$ values that correspond to the shares of the $S^{\star,j}_{\star,\star}$ values masked with P_j's local PRF computations.
- $h^j_{\star,\star}, \Gamma^j_{\star,\star} = h^j_{\star,\star}(T^j_{\star,\star}) \oplus k^j_{\star,\star}$ that are the masked key values.
- λ^j_w for each output wire w that are shares of the output translation table.

Handling errors. Recall that our Π_{DPOLY} protocol will allow an adversary to introduce errors into the computation, namely, for any degree-3 monomial $x_1x_2x_3$, if the party playing the role of P_1 in the multiplication sub-protocol is corrupted, it can introduce an error e_{In} and the product is modified to $(x_1x_2 + e_{\text{In}})x_3$. The adversary can also introduce an error e_{Out} that is simply added to the result of the computation, namely the $S^{\star,\star}_{\star,\star}$ values. Finally, the adversary can reveal arbitrary values for λ^j_w, which in turn means the output translation table can arbitrarily assign the keys to output values.

Our approach to tackle the "e_{In}" errors is to show that these errors can be translated to additive errors on the wires of $\widehat{\mathsf{C}}$ and then rely on the additive resilience property of $\widehat{\mathsf{C}}$. Importantly, to apply this property, we need to demonstrate the errors are independent of the actual wire value. We show this in two logical steps. First, by carefully assigning the roles of the parties in the multiplication subprotocols, we can show that the shares obtained by the parties combine to yield $S^{g,j}_{r_1,r_2} + e^{g,j}_{r_1,r_2} \cdot (T^j_{c,0} \oplus T^j_{c,1})$ where $e^{g,j}_{r_1,r_2}$ is a 4κ bit string (and '$\cdot$' is applied bitwise). In other words, by introducing an error, the adversary causes the decoding procedure of the randomized encoding to result in a string where

each bit comes from either $T^j_{c,b}$ or $T^j_{c,1-b}$. Since an adversary can incorporate different errors in each bit of $S^{\star,\star}_{\star,\star}$, it could get partial information from both the T values. We use a pairwise independent hash function family to mask the actual key, and by the left-over hash lemma, we can restrict the adversary from learning at most one key. As a result, if the majority of the bits in $e^{g,j}_{r_1,r_2}$ are 1 then the "value" on the wire flips, and otherwise "correct".[8] The second logical step is to rely on the fact that there is at least one mask bit λ^j_w chosen by an honest party to demonstrate that the flip event on any wire will be independent of the actual wire value.

To address the "e_{Out}" errors, following [39,44], we show that the BMR encoding is already resilient to such adaptive attacks (where the adversary may add errors to the garbled circuit even after seeing the complete garbling and then deciding on the error).

Finally, to tackle a rushing adversary that can modify the output of the translation table arbitrarily, we rely on the MACs to ensure that the output value revealed can be matched with the MACs revealed along with the output under each party's private MAC key.

Role assignment in the multiplication subprotocols. As described above, we carefully assign roles to parties to restrict the errors introduced in the multiplication protocol. Observe that χ_{r_1,r_2} is a degree-2 computation, which in turn means the expressions $T^j_{c,0} \oplus \chi_{r_1,r_2}(T^j_{c,1} \oplus T^j_{c,0})$ over all garbled rows is a collection of polynomials of degree at most 3. In particular, for every $j \in [n]$, every gate $g \in G$ with input wires a, b and an output wire c, $S^{g,j}_{r_1,r_2}$ involves the computation of one or more of the following monomials:

- $\lambda^{j_1}_a \lambda^{j_2}_b (T^j_{c,1} \oplus T^j_{c,0})$ for $j, j_1, j_2 \in [n]$.
- $\lambda^{j_1}_c (T^j_{c,1} \oplus T^j_{c,0})$ for $j, j_1 \in [n]$.
- $T^j_{c,0}$.

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} (Sect. 3.2), and what products are computed. Letting $\Delta^j_c = (T^j_{c,1} \oplus T^j_{c,0})$, we observe that every product always involves Δ^j_c as one of its operands. Moreover, every term can be expressed as a product of three operands, where the product $\lambda^{j_1}_c \Delta^j_c$ will be (canonically) expressed as $(\lambda^{j_1}_c)^2 \Delta^j_c$ 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 $\lambda^{j_1}_a, \lambda^{j_2}_b$ and Δ^j_c, 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^j_c, where it follows from our simulation that e_{In} will be independent of δ for honest P_j.

We now proceed to a formal description of our protocol.

[8] Even if a particular gate computation is correctly evaluated, it does not necessarily mean this is the correct wire value as the input wire values to the gate could themselves be incorrect due to additive errors that occur earlier in the circuit.

Protocol 2. (Defensibly-simulatable protocol Π_{DMPC})
INPUT: *Parties $P_1, \ldots, P_n$ are given input $\hat{x}_1, \ldots, \hat{x}_n$ of length κ', respectively, and a circuit $\widehat{\mathsf{C}}$ as specified above.*
LOCAL PRE-PROCESSING: *Each party P_i chooses a random MAC key α_i and sets $x_i = \mathsf{Encode}(\hat{x}_i, \alpha_i)$. Let κ be the length of the resulting x_i's, and we fix the notation $[x_i]_j$ as the j^{th} bit of x_i. Next P_i chooses all the randomness that is needed for the BMR encoding of the circuit $\widehat{\mathsf{C}}$. Namely, for each wire w, P_i chooses the masking bit $\lambda^i_w \in \{0,1\}$, random wire PRF keys $k^i_{w,0}, k^i_{w,1} \in \{0,1\}^\kappa$, random functions from a universal hash family $h^i_{w,0}, h^i_{w,1} : \{0,1\}^{4\kappa} \to \{0,1\}^\kappa$ and random hash inputs $T^i_{w,0}, T^i_{w,1} \in \{0,1\}^{4\kappa}$. Then, for every non-output wire w and every gate g for which w is one of the inputs, P_i computes all the PRF values $\Theta^{i,w,g}_{j,r_1,r_2} = \mathsf{F}_{k^i_{w,r_1}}(g, j, r_1, r_2)$ for $j = 1, \ldots, n$ and $r_1, r_2 \in \{0,1\}$. (The values λ^i_w, $T^i_{w,r}$, and $\Theta^{i,w,g}_{j,r_1,r_2}$, will play the role of P_i's inputs to the protocol that realizes the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$.)*
The parties identity the set of 3-monomials that should be computed by the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$ and index them by $1, 2, \ldots, M$. Each party P_i identifies the set of monomials, denoted by Set_i, that depends on any of its inputs (λ^i_w, $T^i_{w,r}$, or $\Theta^{i,w,g}_{j,r_1,r_2}$). As described above, each P_i also determines the role, denoted by $\mathsf{Role}(t,i) \in \{P_1, P_2, P_3\}$, that it plays in the computation of M_t (which is set to $\perp$ if P_i does not participate in the computation of M_t).

- 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 BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$. 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 degree-3 polynomials. It reassembles the output shares to obtain $S^{g,j}_{r_1,r_2}$ for every garbled row r_1, r_2 and gate g.
- DEFENSE: *At this point, the adversary broadcasts its "defense:" The defense for this protocol is a collection of defenses for every monomial M_t that assembles the BMR encoding. The defense for every monomial is as defined in protocol Π_{DMULT} from Sect. 3. Namely, for each party P_i there is an NP language*

$$\mathcal{L}^*_{P_i} = \left\{ (\mathsf{trans}^1, \ldots \mathsf{trans}^M) \;\middle|\; \begin{array}{l} \mathsf{trans}^j \in \mathcal{L}_{p_1}, \mathcal{L}_{p_2}, \mathcal{L}_{p_3} \;\; \textit{if} P_i \textit{ is assigned the role } P_1, P_2, P_3, \\ \qquad\qquad \textit{respectively, in the } j^{th} \textit{ instance of } \Pi_{\text{DMULT}} \\ \wedge \textit{ all the } \mathsf{trans}^j\textit{'s are consistent with the same value of } x_i \end{array} \right\}$$

- 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:*
  - $\widetilde{R}^{g,i}_{r_1,r_2} = \mathsf{F}_{k^j_{a,r_1}}(g, j, r_1, r_2) \oplus \mathsf{F}_{k^j_{b,r_2}}(g, i, r_1, r_2) \oplus S^{g,i}_{r_1,r_2}$ *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* $\{\widetilde{R}^{g,j}_{r_1,r_2}\}_{j\in[n],g\in[G],r_1,r_2\in\{0,1\}}$, *the parties compute each garbled row by* $R^{g,j}_{r_1,r_2} = \bigoplus_{j=1}^{n} \widetilde{R}^{g,j}_{r_1,r_2}$ *and run the decoding procedure* $\mathsf{BMR.Decode}$ *on some standard (arbitrary) topological ordering of the gates.Concretely, 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*

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

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* $k^i_c = \Gamma^i_{c,\Lambda_c} \oplus h(T^j_c)$. *The evaluation is completed when all the gates in the topological order are processed. Finally given* Λ_w *for every output wire* w, *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,

Lemma 3.6. (Defensible simulation MPC) *Assuming either LWE, DDH, QR or DCR, protocol* Π_{DMPC} *securely realizes any* n*-input function* f *in the presence of a "defensible adversary" that always broadcasts a valid defense at the end of the third round and actively corrupting any number of parties.*

Proof. Let $\mathcal{A}$ be a PPT defensible 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 $\overline{I}$, our simulator $\mathcal{S}$ is defined below.

Description of the simulator.

- *Simulating rounds 1-3.* Recall that the parties engage in an instance of Π_{DPOLY} to realize the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$ in the first three rounds. The simulator samples random inputs for the honest parties and generates their messages using these 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, a defensible adversary outputs a valid defense. Next the simulator executes the following procedure to compute the fourth round messages of the honest parties.

  SimGarble(defense):

  1. The simulator extracts from the defense λ^j_w and $T^j_{w,0}, T^j_{w,0} \oplus T^j_{w,1}$ for every corrupted party P_j and internal wire w. Finally, it obtains the vector of errors

$e^{g,j}_{r_1,r_2}$ for every gate g, $r_1, r_2 \in \{0,1\}$ and $j \in I$, introduced by the adversary for row (r_1, r_2) in the garbling of gate g.[9]

2. The simulator defines the inputs of the corrupted parties by using the Λ_w values revealed in round 3 corresponding to the wires w carrying inputs the corrupted parties. Namely, for each such input wire $w \in W$, 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 $\mathsf{Decode}(\mathbf{x}_I)$ to the trusted party computing f, receiving the output $\tilde{y}$. $\mathcal{S}$ fixes $y = \mathsf{Encode}(\tilde{y})$ (recall that Encode in the encoding of an AMD code). Let $y = (y_1, \ldots, y_m)$.
3. Next, the simulator defines the $S^{\star,\star}_{\star,\star}$ values, i.e the plaintexts in the garbled rows. Recall that the shares of the $S^{\star,\star}_{\star,\star}$ values are computed using the Π_{DPOLY} subprotocol. Then the simulator for the main protocol, uses the $S^{\star,\star}_{\star,\star}$ values that are defined by the simulation of Π_{DPOLY}. Next, $\mathcal{S}$ chooses a random $\Lambda_w \leftarrow \{0,1\}$ for every internal wire $w \in W$. Finally, it samples a single key k^j_w for every honest party $j \in \overline{I}$ and wire $w \in W$. We recall that in the standard BMR garbling, the simulator sets the garbled row so that for every gate g with input wires a, b and output wire c, only the row Λ_a, Λ_b is decryptable and decrypting this row gives the single key chosen for wire c (denoted by an active key). In our modified BMR garbling, we will essentially ensure the same, except that we also need to simulate the errors introduced in the computation. More formally, the simulator considers an arbitrary topological ordering on the gates. Fix some gate g in this sequence with a, b as input wires and c as the output wire. Then, for every honest party P_j and random values $T^j_{c,0}$ and $T^j_{c,1}$ that were involved in the computation of the $S^{\star,\star}_{\star,\star}$ values for this gate within the above simulation of Π_{DPOLY}, the simulator defines the bits of $S^{g,j}_{\Lambda_a,\Lambda_b}$ to be $(e^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\Lambda_c}) \oplus (\overline{e^{g,j}_{\Lambda_a,\Lambda_b}} \cdot T^j_{c,\overline{\Lambda_c}})$ if the majority of the bits in $e^{g,j}_{\Lambda_a,\Lambda_b}$ is 1 and $(\overline{e^{g,j}_{\Lambda_a,\Lambda_b}} \cdot T^j_{c,\Lambda_c}) \oplus (e^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\overline{\Lambda_c}})$ otherwise.
4. Next, it generates the fourth message on behalf of the honest parties. Namely, for every gate g and an active row Λ_a, Λ_b, the shares of the honest parties are computed assuming the output of the polynomials defined in the BMR encoding are $S^{g,j}_{\Lambda_a,\Lambda_b}$ for every j masked with the PRF under the keys k^j_a, k^j_b as defined by $\tilde{R}^{g,j}_{\Lambda_a,\Lambda_b}$. For the remaining three rows the simulator sends random strings. On behalf of every honest party P_j, in addition to the shares, the fourth round message is appended with a broadcast of the message $(r, h(T^j_{w,\Lambda_w}) \oplus k^j_w)$ 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. Intuitively, upon decrypting $S^{g,j}_{\Lambda_a,\Lambda_b}$ for any gate g, the adversary learns the majority of the bits of T^j_{c,Λ_c} with which it can learn only k^j_c.

[9] The errors are bits and are extracted for each monomial where the corrupted party plays the role of P_1. For simplicity of notation we lump them all in a single vector.

- The simulator sends the messages as indicated by the procedure above on behalf of the honest parties. 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 executes the following procedure that takes as input all the messages exchanged in the fourth round, the Λ_w values broadcast in the third round and the target output y. It determines whether the final output needs to be delivered to the honest parties in the ideal world.

  ReconGarble(4^{th} round messages, Λ_w for every input wire w,y):

  - The procedure reconstructs the garbling $\mathrm{GC}_{\mathcal{A}}$ using the shares and the keys provided. 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. If both events hold, the procedure outputs the OK message and otherwise outputs $\perp$.

- Finally, if the procedure outputs OK the simulator instructs the trusted party to deliver $\tilde{y}$ to the honest parties.

Claim 3.7. $\boldsymbol{REAL}_{\Pi_{\mathrm{DMPC}},\mathcal{A}(z),I}(\kappa,\hat{x}_1,\ldots,\hat{x}_n) \overset{\mathrm{c}}{\approx} \boldsymbol{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa,\hat{x}_1,\ldots,\hat{x}_n)$.

Proof. Assume for contradiction, there exists an adversary $\mathcal{A}$, distinguisher D and polynomial $p(\cdot)$ such that the probability with which D distinguishers the two distributions $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa,\hat{x}_1,\ldots,\hat{x}_n)$ and $\mathbf{REAL}_{\Pi,\mathcal{A}(z),I}(\kappa,\hat{x}_1,\ldots,\hat{x}_n)$ for infinitely many κ is $\frac{1}{p(\kappa)}$. Fix κ and a set of inputs for the 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 indistinguishability via standard hybrid arguments. 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 $\frac{1}{p(\kappa)q(\kappa)}$.

Hybrid H_0. This experiment proceeds identically to the real execution. More precisely, in H_0 we consider a simulator $\mathcal{S}_0$ that knows all honest parties' real inputs $\{\hat{x}_j\}_{j\in\bar{I}}$ and starts an execution with $\mathcal{A}$ providing it fresh randomness and inputs $\{\hat{x}_j\}_{j\in I}$ and emulating the actions of the honest parties using the real inputs. The output of the experiment is $\mathbf{REAL}_{\Pi_{\mathrm{DMPC}},\mathcal{A}(z),I}(\kappa,\hat{x}_1,\ldots,\hat{x}_n)$ which consists of $\mathcal{A}$'s view and the output of honest parties. Note that the Λ_w values are computed correctly in the third round using the real inputs $\{\hat{x}_j\}_{j\in\bar{I}}$ of the honest parties.

Hybrid H_1. In this hybrid, we follow the simulation strategy of Π_{DPOLY}. Namely, we define a simulator $\mathcal{S}_1$ that calls $\mathcal{S}_{\mathrm{DPOLY}}$. We recall that $\mathcal{S}_{\mathrm{DPOLY}}$ chooses random inputs for each honest party and then follows the honest protocol execution using these random inputs until the end of the 3^{rd} round. At this point, $\mathcal{S}_{\mathrm{DPOLY}}$ is expected to receive a valid defense which is identical to the defence in Π_{DMPC}. Then, upon receiving a valid "defense" that includes the inputs and randomness that the adversary used to generate its messages, $\mathcal{S}_{\mathrm{DPOLY}}$ extracts from that defense the effective inputs of the adversary to Π_{DPOLY}, and other values to help with the rest of the simulation and forward this information to $\mathcal{S}_1$. Upon creating the $S^{\star,\star}_{\star,\star}$ values of the honest parties that denote the garbled circuits shares, $\mathcal{S}_1$ completes the simulation of the third and fourth messages

using these values and the honest parties' real inputs (recall that these inputs are needed to compute the Λ_w values for every input wire that is associated with the honest parties' inputs). Then the following claim holds,

Claim 3.8. *Assuming affine homomorphic PKE with equivocation, the adversary's view in* H_1 *is indistinguishable from its view in* H_0.

Note that indistinguishability of H_1 from H_0 follows essentially from the simulation strategy for Π_{DPOLY}. This in turn relies on executing the hybrids for the Π_{DMULT} subprotocols.

In the next set of hybrids we change the simulation of the fourth round messages for the honest parties from using $\mathsf{BMR.Encode}^{\widehat{C}}$ to SimGarble. Next, we consider an arbitrary topological ordering on the gates $g \in G$ of the computed circuit and will inductively consider a sequence of hybrids in this ordering. We will maintain as an invariant that while at gate g, the simulation does not require the knowledge of any 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.

Fix gate g with input wires a and b and output wire c. Then, define the following two hybrids.

Hybrid H_2^g. Consider an arbitrary gate g in the topological order with input wires a, b and output wire c and fix an honest party P_j. 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. More formally,

Claim 3.9. *Assuming pseudorandomness of* F*, the adversary's view in* H_2^g *is indistinguishable from its view in* H_2^{g-1}*. Where* $g-1$ *is the gate that precedes gate* g *in the fixed topological ordering.*

Proof. We consider two cases.

g **is an input gate:** In this case we note that the inactive keys $k^j_{a,\bar{\Lambda}_a}$ and $k^j_{b,\bar{\Lambda}_b}$ are not involved in the construction of the garbled circuit other than in the garbled gates for which wires a or b are inputs to these gates. This means that we can replace the three inactive rows in gate g and reduce the security to the pseudorandomness of F. Specifically, assume by contradiction that the two views are distinguishable by an adversary $\mathcal{A}$. We construct an adversary $\mathcal{A}_{\mathsf{F}}$ that distinguishes the computation based on $k^j_{a,\bar{\Lambda}_a}, k^j_{b,\bar{\Lambda}_b}$ and the computation based on truly random functions f_1, f_2 in the sense of Definition 2.4.

Specifically, $\mathcal{A}_{\mathsf{F}}$ behaves like the simulator in H_2^g with the exception that it does not choose the inactive keys $k^j_{a,\bar{\Lambda}_a}$ and $k^j_{b,\bar{\Lambda}_b}$. In order to simulate the garbling of any gate g' that precedes g (by the topological ordering, such gates can only be input gates), $\mathcal{A}_{\mathsf{F}}$ honestly computes the active row of this gate and chooses the remaining three rows at random. To simulate the garbling of any gate g' that comes after g in the topological ordering, $\mathcal{A}_{\mathsf{F}}$ generates the garbling of g' as in H_2^{g-1}. Namely, it fixes the garbing shares as would have been generated by P_j in the real execution.

g is not an input gate: By the assumption that the inactive keys $k^j_{a,\bar{\Lambda}_a}$ and $k^j_{b,\bar{\Lambda}_b}$ are not in use, then the reduction follows as in the previous case. □

Hybrid H^g_3. In this hybrid we replace the inactive $T^j_{c,\beta}$ values within $\Gamma^j_{c,\beta}$ and $S^{g,j}_{r_1,r_2}$ for every honest party P_j and inactive row in the garbled circuit. More concretely, consider an honest party P_j. We recall that the PRF 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. Furthermore, 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 either $T^j_{c,0}$ or $T^j_{c,1}$, 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 [40] that except with probability $2^{-\kappa}$, the key $k^j_{c,\beta}$ is hidden. If a majority of the bits in $e^{g,j}_{r_1,r_2}$ is 0 then we set $\Lambda_c = \widetilde{\Lambda_c}$ and otherwise we set $\Lambda_c = 1 \oplus \widetilde{\Lambda_c}$ where

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

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

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 $\{0,1\}^\kappa$. We can conclude that,

Claim 3.10. *Based on the pairwise independent property of $h^j_{c,\bar{\Lambda}_c}$. The adversary's view in H^g_3 is statistically indistinguishable from its view in H^{g-1}_3.*

Note that at this point, the adversary's view is independent of the honest parties' inputs. This is because the inactive keys are not part of the view and it is impossible to determine the values of which the active keys are associated with.

Hybrid H_4. In this hybrid, we modify the λ^j_w shares of the honest parties P_j for every $j \in \bar{I}$ 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 $\widehat{\mathsf{C}}'$ 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 $\widehat{\mathsf{C}}'$,

the simulator computes $e_w = \widetilde{\Lambda_w} \oplus \Lambda_w$ where $\widetilde{\Lambda_w}$ corresponds to the "correct value" as defined above. 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_w^j for the honest parties $P_j \in \bar{I}$ subject to $(\bigoplus_{i=1}^n \lambda_w^i) = \Lambda_w \oplus y_t$.

The output of H_4 is identically distributed to the output of H_3 as 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 $\mathsf{Decode}(\mathbf{x}_I)$ to the trusted party computing f and receives the output $\tilde{y}$. $\mathcal{S}$ fixes $y = \mathsf{Encode}(\tilde{y})$. Let $y = (y_1, \ldots, y_m)$. Moreover, $\mathcal{S}$ defines $(\bigoplus_{i=1}^n \lambda_w^i) = \Lambda_w \oplus y_t$ 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 [27]]. Furthermore, this hybrid produces a distribution according to $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$. □

4. MPC in Four Rounds

Our actively secure protocol is built on the four-round defensible multiplication protocol for the BMR encoding function $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$ from Sect. 3.4, dented by Π_{DMPC}. To enforce correct behavior we also rely on: (1) A three-round weak one-many non-malleable commitment scheme against synchronizing adversaries (cf. Definition 2.12), with messages $\mathsf{nmcom} = (\mathsf{nmcom}[1], \mathsf{nmcom}[2], \mathsf{nmcom}[3])$. (2) A two-round resettable reusable witness indistinguishable proof (cf. Definition 2.13, built from ZAPs), where we denote the messages by $\mathsf{ZAP} = (\mathsf{ZAP}[1], \mathsf{ZAP}[2])$.

At a high level, the parties use non-malleable commitments to commit to their inputs, their randomness for the BMR encoding, and (most of) their randomness in Π_{DMPC}. They run a ZAP proof for "proving correct behavior" in Π_{DMPC}, where correct behavior is defined the same way as the defense in the previous section (cf. the language $\mathcal{L}^*_{P_i}$ in the DEFENSE step in Protocol 2). This is meant to narrow down the potential attacks that the adversary can mount to essentially just what a defensible adversary can do (and those we know how to handle as per the previous section). The structure of our protocol is as follows:

Base protocol Π_{DMPC}**:** The core of our protocol involves executing the Π_{DMPC} protocol in the 4 rounds.

Non-malleable commitments: For every i and j, P_i runs two executions of non-malleable commitments $\mathsf{nmcom}^0_{i,j}, \mathsf{nmcom}^1_{i,j}$ with every other party P_j in the first three rounds.

Let wit_i be a witness for P_i's actions in Π_{DMPC}, as needed for the defense in this protocol. P_i chooses two random strings $w^0_{i,j}, w^1_{i,j}$, and uses specifically the GRRV protocol [35] to commit to them separately, in two commitment protocols. In the third round, together

with the last messages of $\mathsf{nmcom}^0_{i,j}$, $\mathsf{nmcom}^1_{i,j}$, P_i also sends $\tilde{w}^0_{i,j} := \mathsf{wit}_i \oplus w^0_{i,j}$ and $\tilde{w}^1_{i,j} := \mathsf{wit}_i \oplus w^1_{i,j}$, respectively. [10]

ZAPs from Round 1 to Round 2 ($\mathsf{ZAP}_{\mathrm{ENC}}$): Every party P_i proves to every other party P_j in the first two rounds that it generated correctly all the public keys and ciphertexts in the first round of Π_{DMPC}.

ZAPs from Round 2 to Round 3 ($\mathsf{ZAP}_{\mathrm{COM}}$): Every party P_i proves to every other party P_j from Round 2 to Round 3 that at least one of $\mathsf{nmcom}^0_{i,j}$, $\mathsf{nmcom}^1_{i,j}$ is a valid commitment to a valid witness. Namely for some $b \in \{0, 1\}$, $\mathsf{nmcom}^b_{i,j}$ is a valid commitment to a value $w^b_{i,j}$ such that $w^0_{i,j} \oplus \tilde{w}^0_{i,j}$ is a valid witness for the language $\mathcal{L}^*_{P_i}$ from Protocol 2 in the previous section.[11]

4.1. *Proof Overview and Highlights*

On a high-level, we will follow the proof structure of Π_{DMPC} protocol. First we recall the simulator for Π_{DMPC}. The basic idea here is that the messages for the Π_{DPOLY} subprotocol are simulated by choosing random inputs for the honest party and then the fourth round is simulated only using the defense provided by the adversary. Importantly, the random inputs chosen for Π_{DPOLY} are independent of the fourth round message generated for the honest parties. The simulator for our protocol in this section is an extension of the simulator for Π_{DMPC} that additionally generates the non-malleable commitments and ZAPs (using random inputs for the honest parties), and uses rewinding to extract the "defense."

Proving indistinguishability of our simulation involves the set of hybrid experiments considered for Π_{DMPC}, interspersed with additional intermediate hybrid experiments to deal with rewinding and non-malleability.

First we recall the hybrid experiments for the Π_{DMPC} protocol at a high-level. Let $\mathsf{INP}_{\mathsf{BMR}}$ denote the inputs used by the honest parties in the real world for the Π_{DPOLY} subprotocol and let RND be the inputs chosen by the simulator. The proof consists of three main steps:

1. The first step consists of hybrids where the challenger generates the fourth round messages for honest parties using $\mathsf{INP}_{\mathsf{BMR}}$ and the "defense" of the adversary, but does not use any of the intermediate values obtained in the execution of Π_{DPOLY}. Recall that in any instance of the Π_{DMULT} subprotocol of Π_{DPOLY}, party P_3 relies on its intermediate values to determine its output shares used in generating the fourth message. For P_1 and P_2, the outputs s_1 and s_2 can also be generated using the inputs chosen for the honest parties and the defense provided by the adversary. (These are hybrids H_2 and H_3 in the proof of Π_{DMULT} in Sect. 3.2.)
2. In the second step, the inputs for the honest parties in the Π_{DPOLY} subprotocol is switched from $\mathsf{INP}_{\mathsf{BMR}}$ to RND. We remark here that the fourth round message

[10] This is essentially the technique of Ciampi et al. [16], adjusting these commitment schemes to handle delayed inputs.

[11] It is important to include a proof that $\mathsf{nmcom}^b_{i,j}$ is a valid commitment in the ZAP, since the GRRV subprotocol does not ensure this.

is still computed using $\mathsf{INP}_{\mathsf{BMR}}$ in these set of hybrids. This involves running through the hybrids in the proof of Lemma 3.1 for each Π_{DMULT} instance in the Π_{DPOLY} subprotocol. (These are hybrids H_4 through H_7 in the proof of Π_{DMULT} in Sect. 3.2.)

3. In the last step, we switch the fourth round message from the real to the simulated garbling circuit. The real garbled circuit is constructed according to $\mathsf{BMR.Encode}^{\widehat{C}}$, while the simulated garbled circuit follows the procedure SimGarble from the proof of Lemma 3.6. (These are hybrids $\{\mathsf{H}_2^g\}_{g\in[G]}, \mathsf{H}_3, \mathsf{H}_4, \mathsf{H}_5$ from the proof of Π_{DMPC} in Sect. 3.4.)

The hybrids for the protocol in this section will follow the same three steps. In fact, the first and third steps remain the same. The main technical challenge is executing the hybrids for the second step. This involves decoupling the inputs used in the Π_{DPOLY} from the fourth round message. The main issue is that this is affected by the non-malleable commitments and ZAPs that depend on the inputs and randomness used in the Π_{DPOLY} subprotocol. Crucially, since we do not rely on zero-knowledge, the challenger must possess a valid witness whenever it needs to generate the ZAP proofs on behalf of the honest parties.

We now discuss this second step in more detail, referring to the hybrids from the proof of Π_{DMULT} in Sect. 3.2. We describe this for each of the three parties P_1, P_2 and P_3 as they involves slightly different set of intermediate hybrid experiments.

Switching P_1's input from x_1 to $\widehat{x}_1$: Recall that P_1 uses its input x_1 in the first message of the protocol and we switch this in two steps. First, we switch the u value to a random u' used to generate $\mathsf{C}_\gamma^1[1]$ and $\mathsf{C}_\gamma^2[1]$ in Hybrid H_5 and then switch x_1 to $\widehat{x}_1$ in H_6. This is needed since we cannot carry out a reduction to the semantic security of the underlying encryption scheme while producing the correct u.

Switching P_2's input from x_2 to $\widehat{x}_2$: If P_1 and P_3 generate their first messages correctly (meaning the key generation was correct), then x_2 is statistically hidden because of the randomization due to r_2.

Switching P_3's input from x_3 to $\widehat{x}_3$: This involves switching the ciphertexts generated by P_3 one by one from an encryption of x_3 to an encryption of $\widehat{x}_3$.

In the following we focus on the first three rounds where we highlight how to perform the reductions specified above in our final protocol. We begin with the three-round non-malleable commitment scheme that we employ.

GRRV Non-malleable commitment [35]. We will rely on a three-round non-malleable commitment scheme implicit in the work of GRRV. This protocol has the form (Commit, Challenge, Response). Given a man-in-the-middle synchronized adversary, engaging in one left commitment as the receiver and many right commitments as the committer, the commitment satisfies the weak one-many non-malleability property. Additionally we will rely on two other parties satisfied by the commitment scheme: (1) First, the third round, namely the response message is "simulatable", and (2) second, the message can be extracted from two valid transcripts with common first message. See Appendix 2.5.1 for the details of this commitment scheme and a formalism of these properties.

Real execution. In the real execution the values Λ_w in the third round of Π_{DMPC} are computed using the inputs $\mathsf{INP}_{\mathsf{BMR}}$ for the BMR encoding, and the real input INP_x.

Ideal execution. In the ideal execution, the simulator honestly generates the messages on behalf of the honest parties. In the BMR part, the simulation is carried out by sampling random inputs RND for the BMR encoding, honestly committing to a defense using nmcom and completing the ZAP proofs using the committed valid defense. The only difference in the simulation is that the Λ_w values are generated at random, independently of $\mathsf{INP}_{\mathsf{BMR}}$ and INP_x. Consequently, the simulator can simulate the first three rounds just by following the honest strategy with input RND, including all the ZAPs.

Our protocol employ two instances of the nmcom where in the real world, the honest parties commit to their defenses according to $\mathsf{INP}_{\mathsf{BMR}}$ whereas in the ideal world, the simulator commits to defenses according to RND. We consider a sequence of intermediate hybrid experiments where we switch from $\mathsf{INP}_{\mathsf{BMR}}$ to RND both in the nmcom as well as in the actions of the honest parties in Π_{DPOLY} subprotocol.

Premature rewinding. Towards this, in our intermediate hybrid experiments, we will introduce an additional rewinding phase, referred to as the *premature rewinding* phase. In this phase, the "challenger" extracts a "trapdoor," which is used in the main execution. The trapdoor is essentially the secret decryption keys of the adversary. More precisely, the challenger first simulates the first round of the protocol, then rewinds the second and third rounds until it extracts the trapdoor from the adversary. Once it has the trapdoor, it rewinds to the end of the first message, and then completes the main execution to determine the view of the adversary (depending on the hybrid). Note that, executing premature rewinding requires that the challenger knows some witness for the ZAP proofs in rounds 2-3. This does not pose an issue for the challenger, since it knows the "real" $\mathsf{INP}_{\mathsf{BMR}}$ and can use it during premature rewinding. It is only the simulator, who does not do premature rewinding, that lacks knowledge of $\mathsf{INP}_{\mathsf{BMR}}$.

As stated above, we employ two instances of nmcom, each committing to either $\mathsf{INP}_{\mathsf{BMR}}$ or RND, where in some of the hybrids we have two additional instances due to the premature rewinding phase (again, each using either $\mathsf{INP}_{\mathsf{BMR}}$ or RND). We thus can describe each hybrid by a tuple that specify the input used in each of these instances. For example $(-, -; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}})$ denotes the real execution without premature rewinding, whereas $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}}; \mathsf{RND}, \mathsf{RND})$ denotes a hybrid in which the challenger uses $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}})$ in the premature rewinding phase and $(\mathsf{RND}, \mathsf{RND})$ in the main execution, etc. We also sometimes use GRB to denote a "garbage value" that the challenger does not know and does not need to use.

Importantly, the challenger *cannot* use RND (or GRB) in the premature rewinding phase, since it can only produce a ZAP for it after running premature rewinding and extracting a witness. Similarly, the challenger can use RND in the main execution *only if* it was able to extract a witness in the premature rewinding phase. However, it is "legitimate" to extract a trapdoor using, e.g., $(\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}})$ in the premature rewinding phase, and then use the extracted trapdoor to generate a transcript for $(\mathsf{RND}, \mathsf{RND})$ in the main execution.

Our goal is to show that $(-, -; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}})$ generated in a real execution is indistinguishable from $(-, -; \mathsf{RND}, \mathsf{RND})$ generated in the simulation. We continue with several observations:

1. For any $W, X, Y, Z \in \{\mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND}, \mathsf{GRB}\}$, the only difference between the view in the two hybrids $(-, -; Y, Z)$ and $(W, X; Y, Z)$ (i.e., with or without pre-

mature rewinding) is that in the later, the challenger may fail to extract a trapdoor in the premature rewinding phase and hence aborts rather than continue to the main phase. We therefore need to set the failure probability low enough (by setting the number of reqinding attempts high enough), relative to the distinguishing advantage between these hybrids.
Namely, if we want to show that the distinguishing advantage between these hybrids is bounded below some ε (which we assumed toward contradiction is non-negligible), and we are dealing with an adversary that completes the proof with probability δ (also non-negligible), then the challenger needs to rewind n times such that $(1-\delta)^n \ll \varepsilon$. This will ensure that we have $SD((-,-;Y,Z),(W,X;Y,Z)) \leq \varepsilon$ as needed.[12]

2. For any $Y, Z \in \{\mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND}\}$ and any hybrid $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}}; Y, Z)$, we can always replace one of the $\mathsf{INP}_{\mathsf{BMR}}$ by GRB, and get an indistinguishable view by a reduction to the hiding property of the non-interactive commitment in the first round of GRRV. This game is formalized as the simulatable property in Appendix 2.5.1. Namely

$$(\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}}; Y, Z) \overset{c}{\approx} (\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}}; Y, Z) \overset{c}{\approx} (\mathsf{INP}_{\mathsf{BMR}}, \mathsf{GRB}; Y, Z).$$

3. For any $X, Y, Z \in \{\mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND}\}$, it holds that

$$(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{GRB}; X, Y) \overset{c}{\approx} (\mathsf{INP}_{\mathsf{BMR}}, \mathsf{GRB}; X, Z)$$

by a reduction to the weak non-malleability of GRRV. Namely, as long as the challenger does not need to use any information about the internals of the GRB commitment, we can setup a reduction where this commitment comes "from the outside", and the reduction needs to tell if it commits to Y or to Z. In the reduction, we receive the first message for Y or Z and treat it as GRB, then we carry out the premature rewinding locally, without sending the second round message, and only when we manage to extract the trapdoor we complete the reduction sending the second and third messages.
A similar argument shows that $(\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}}; Y, X) \overset{c}{\approx} (\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}}; Z, X)$.

Using these observations, we can show a chain of indistinguishable hybrids between a real execution $(-,-;\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}})$ and an ideal execution $(-,-;\mathsf{RND},\mathsf{RND})$

[12] In our proof of security we assume that the challenger knows ε. This allows us to prematurely rewind only a fixed (but sufficiently more than $1/\varepsilon$) number of times.

as follows:

$$
\begin{aligned}
(\ -\ ,-\ ;\ \mathsf{INP}_{\mathsf{BMR}},\ \mathsf{INP}_{\mathsf{BMR}}\ ) &\overset{c}{\approx} (1)\ (\ \mathsf{INP}_{\mathsf{BMR}},\ \mathsf{INP}_{\mathsf{BMR}};\ \mathsf{INP}_{\mathsf{BMR}},\ \mathsf{INP}_{\mathsf{BMR}}\ )\\
&\overset{c}{\approx} (2)\ (\ \mathsf{INP}_{\mathsf{BMR}},\ \ \mathsf{GRB};\ \ \mathsf{INP}_{\mathsf{BMR}},\ \mathsf{INP}_{\mathsf{BMR}}\ )\\
&\overset{c}{\approx} (3)\ (\ \mathsf{INP}_{\mathsf{BMR}},\ \ \mathsf{GRB};\ \ \mathsf{INP}_{\mathsf{BMR}},\ \ \mathsf{RND}\ \ )\\
&\overset{c}{\approx} (2)\ (\ \mathsf{INP}_{\mathsf{BMR}},\ \mathsf{INP}_{\mathsf{BMR}};\ \mathsf{INP}_{\mathsf{BMR}},\ \ \mathsf{RND}\ \ )\\
&\overset{c}{\approx} (2)\ (\ \ \mathsf{GRB},\ \ \mathsf{INP}_{\mathsf{BMR}};\ \mathsf{INP}_{\mathsf{BMR}},\ \ \mathsf{RND}\ \ )\\
&\overset{c}{\approx} (3)\ (\ \ \mathsf{GRB},\ \ \mathsf{INP}_{\mathsf{BMR}};\ \ \mathsf{RND},\ \ \mathsf{RND}\ \ )\\
&\overset{c}{\approx} (1)\ (\ \ \ -,\ \ \ \ -;\ \ \mathsf{RND},\ \ \mathsf{RND}\ \ )
\end{aligned}
$$

As a final remark, we note that the preceding discussion is a simplified view of the hybrids that highlights the main ideas behind premature rewinding and non-malleability. Additional intermediate hybrids are required to switch the ZAP witness. For instance, in the sequence above, the ZAP witness can be switched from depending on the first non-malleable commitment to the second between ($\mathsf{INP}_{\mathsf{BMR}}$, $\mathsf{INP}_{\mathsf{BMR}}$; $\mathsf{INP}_{\mathsf{BMR}}$, RND) and (GRB, $\mathsf{INP}_{\mathsf{BMR}}$; $\mathsf{INP}_{\mathsf{BMR}}$, RND) hybrids.

4.2. *Four-Round Actively Secure MPC*

In this section we formally describe our protocol. For a slightly modular exposition, we present the main protocol and the description of the simulator by abstracting the 4-round protocol π_{DMPC} from the previous section. Our proof of security however will rely on the internals of the π_{DMPC} protocol and we inline them in the hybrid experiments.

Protocol 3. (Actively secure protocol Π_{MPC}) INPUT: *Parties $P_1, \ldots, P_n$ are given input $\hat{x}_1, \ldots, \hat{x}_n$ of length κ', respectively, and a circuit $\widehat{\mathsf{C}}$.*

- LOCAL PRE-PROCESSING: *Each party P_i chooses a random MAC key α_i and sets $x_i = \mathsf{Encode}(\hat{x}_i, \alpha_i)$. Let κ be the length of the resulting x_i's, and we fix the notation $[x_i]_j$ as the j^{th} bit of x_i. Next P_i chooses all the randomness that is needed for the BMR encoding of the circuit $\widehat{\mathsf{C}}$. Namely, for each wire w, P_i chooses the masking bit $\lambda_w^i \in \{0,1\}$, random wire PRF keys $k_{w,0}^i, k_{w,1}^i \in \{0,1\}^\kappa$, random functions from a pairwise independent hash family $h_{w,0}^i, h_{w,1}^i : \{0,1\}^{4\kappa} \to \{0,1\}^\kappa$ and random hash inputs $T_{w,0}^i, T_{w,1}^i \in \{0,1\}^{4\kappa}$.*
  Then, for every non-output wire w and every gate g for which w is one of the inputs, P_i computes all the PRF values $\Theta_{j,r_1,r_2}^{i,w,g} = F_{k_{w,r_1}^i}(g, j, r_1, r_2)$ for $j = 1, \ldots, n$ and $r_1, r_2 \in \{0,1\}$. (The values λ_w^i, $T_{w,r}^i$, and $\Theta_{j,r_1,r_2}^{i,w,g}$, will play the role of P_i's inputs to the protocol that realizes the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$.)

The parties identify the set of 3-monomials that should be computed by the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathsf{C}}}$ and enumerate them by integers from $[M]$. Moreover, each party P_i identifies the set of monomials, denoted by Set_i, that depends on any of its inputs (λ_w^i, $T_{w,r}^i$, or $\Theta_{j,r_1,r_2}^{i,w,g}$). As described in Sect. 3.4, each P_i also determines the role, denoted by

$\mathsf{Role}(t,i) \in \{P_1, P_2, P_3\}$, *that it plays in the computation of* M_t *(which is set to* $\perp$ *if* P_i *does not participate in the computation of* M_t*).*

- Round 1:*For* $i \in [n]$ *each party* P_i *proceeds as follows:*
  - *Engages in an instance of the three-round non-malleable commitment protocol* nmcom *with every other party* P_j*, committing to arbitrarily chosen values* $w_{0,i}, w_{1,i}$*. Denote the messages sent within the first round of this protocol by* $\mathsf{nmcom}^0_{i,j}[1]$,$\mathsf{nmcom}^1_{i,j}[1]$*, respectively.*
  - *Broadcasts the message* $\Pi^{i,j}_{\mathrm{DMPC}}[1]$ *to every other party* P_j*.*
  - *Engages in a ZAP protocol with every party other* P_j *for the* NP *language* $\mathcal{L}'_{\mathsf{Role}(t,i)}$ *defined in Sect. 3.2, for every monomial* M_t *in case* $\mathsf{Role}(t,i) \in \{P_1, P_3\}$*. Note that the first message, denoted by* $\mathsf{ZAP}_{i,j}[1]$ *is sent by* P_j *(so* P_i *sends the first message to all the* P_j*'s for their respective ZAPs).*
- Round 2: *For* $i \in [n]$ *each party* P_i *proceeds as follows:*
  - *Sends the messages* $\mathsf{nmcom}^0_{i,j}[2]$ *and* $\mathsf{nmcom}^1_{i,j}[2]$ *for the second round of the respective non-malleable commitment.*
  - *Engages in a ZAP protocol with every party other* P_j *for the* NP *language* $\mathcal{L}_{\mathsf{Role}(t,i)}$ *defined in Sect. 3.2 for every monomial* M_t*. As above, the first message, denoted by* $\mathsf{ZAP}_{i,j}[1]$ *is sent by* P_j *(so* P_i *sends the first message to all the* P_j*'s for their respective ZAPs).*
  - *Sends the message* $\Pi^{i,j}_{\mathrm{DMPC}}[2]$ *to every other party* P_j*.*
  - *Sends the second message of the ZAP proof for the language* $\mathcal{L}'_{\mathsf{Role}(t,i)}$*.*
- Round 3: *For* $i \in [n]$ *each party* P_i *proceeds as follows:*
  - *Sends the messages* $\mathsf{nmcom}^0_{i,j}[3]$, $\mathsf{nmcom}^1_{i,j}[3]$ *for the third round of the respective non-malleable commitment and the BMR encoding protocols. For* $b \in \{0,1\}$ *define the NP language:*

$$\mathcal{L}_{\mathsf{nmcom}} = \Big\{ \mathsf{nmcom}^*_{i,j}[1], \mathsf{nmcom}^*_{i,j}[2], \mathsf{nmcom}^*_{i,j}[3] \,\big|\, \exists\, b \in \{0,1\} \text{ and } (w_i, \rho_i) \;\; s.t. \;\; \mathsf{nmcom}^b_{i,j} = \mathsf{nmcom}(w_i; \rho_i) \Big\}.$$

  - *Chooses* $\tilde{w}_{0,i}$ *and* $\tilde{w}_{1,i}$ *such that* $\forall m \in [\mathsf{Set}_i], w_{0,i} + \tilde{w}_{0,i} = w_{1,i} + \tilde{w}_{1,i} = \mathsf{wit}_i$ *where* wit_i *is the witness of transcript* $(\mathsf{trans}^0_{\mathsf{Role}(1,i)} || \ldots || \mathsf{trans}^0_{\mathsf{Role}(|\mathsf{Set}_i|,i)} || \mathsf{trans}^0_{\mathsf{nmcom}})$ *and* $\mathsf{Role}(t,i) \in \{P_1, P_2, P_3\}$*, where* $\mathsf{trans}^b_\star$ *is as defined in Sect. 3.2.*
  - *Generates the message* $\mathsf{ZAP}_{i,j}[2]$ *for the second round of the ZAP protocol relative to the* NP *language*

$$\mathcal{L}_{\mathsf{Role}(1,i)} \wedge \ldots \wedge \mathcal{L}_{\mathsf{Role}(|\mathsf{Set}_i|,i)} \wedge \mathcal{L}_{\mathsf{nmcom}} \wedge \left(w_{b,i} + \tilde{w}_{b,i} = \mathsf{wit}_i\right)$$

    where $\mathcal{L}_{\mathsf{Role}(\cdot,i)}$ *is defined in protocol 1.*
  - *Broadcasts the message* $\Pi^{i,j}_{\mathrm{DMPC}}[3]$ *to every other party* P_j*.*

For every $j \in [n]$, let $\{S_{\ell,j}\}_{\ell \in M}$ be the output of party P_j for the M degree-3 polynomials. It reassembles the output shares to obtain $S^{g,j}_{r_1,r_2}$ for every garbled row r_1, r_2 and gate g.

- ROUND 4: *Finally, broadcasts the message $\Pi^{i,j}_{\text{DMPC}}[4]$ to every other party P_j.*
- OUTPUT: *As defined in Π_{DMPC}.*

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

Theorem 4.1. (Main) *Assuming the existence of affine homomorphic encryption (cf. Definition 2.6) and enhanced trapdoor permutations, protocol Π_{MPC} securely realizes any n-input function f in the presence of static, active adversaries corrupting any number of parties.*

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

The description of the simulator.

Rounds 1-3. In the first three rounds, the simulator simulates the honest party messages by simply following the honest code where the input and random tape are sampled uniformly at random. More precisely, $\mathcal{S}$ samples random inputs $\{\mathbf{x}'_i\}_{i \in \bar{I}}$ for the honest parties and generates its messages in the first three rounds following the honest strategy. Recall that in the first three rounds only Λ_w broadcasted in the third round depends on the real inputs of the honest parties to function f. The rest of the messages only depend on randomness sampled by the honest parties (eg, inputs to the underlying Π_{DPOLY} subprotocol that is part of the Π_{DMPC} protocol). We denote the inputs that are used in the Π_{DPOLY} subprotocol as RND. In the third round, the simulator sends the Λ_w values for every input wire of an honest party, computed based on the random inputs $\{\mathbf{x}'_i\}_{i \in \bar{I}}$ and RND. If the adversary aborts before completing the third round or the verification of the ZAP proofs fails, the simulator halts outputting the adversary's view.

Otherwise the simulator rewinds the adversary to extract a valid defense within the non-malleable commitments. Namely, it executes the extractor provided by the non-malleable commitment that rewinds the execution to the end of the first round and plays the second round with new challenges. The honest party's messages are played according to the inputs chosen for them. The extractor repeatedly rewinds until it extracts the committed values (namely, the defences). If it runs longer than $2^{\kappa/2}$ steps, the simulator aborts outputting fail.

Round 4. Upon extracting the defense for every monomial M_l and corrupted party that participate in this computation, the simulator extracts from the defense the adversary's input and the errors introduced in the computation as in the simulation for Π_{DMPC}, and sends this input to $\mathcal{F}$. Upon receiving the response from the functionality, the simulator uses the procedure SimGarble defined for $\mathcal{S}_{\text{DMPC}}$ to generate the fourth round messages of the honest parties, which it feeds to the adversary. If

the adversary sends its fourth round message, $\mathcal{S}$ runs the ReconGarble procedure defined for $\mathcal{S}_{\text{DMPC}}$ to determine whether the honest parties receive the output.

Claim 4.1. $\boldsymbol{REAL}_{\Pi_{\text{MPC}},\mathcal{A}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n) \overset{c}{\approx} \boldsymbol{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$.

Proof. Assume for contradiction, there exists an adversary $\mathcal{A}$, distinguisher D and polynomial $p(\cdot)$ such that the probability with which D distinguishers the two distributions $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$ and $\mathbf{REAL}_{\Pi_{\text{MPC}},\mathcal{A}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$ for infinitely many κ is $\frac{1}{p(\kappa)}$. Fix κ and a set of inputs for the 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 indistinguishability via standard hybrid arguments. 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 $\frac{1}{p(\kappa)q(\kappa)}$.

Hybrid H_0. This experiment proceeds identically to the real execution. More precisely, in H_0 we consider a simulator $\mathcal{S}_0$ that has all the honest parties real inputs $\{x_j\}_{j\in\bar{I}}$ and starts an execution with $\mathcal{A}$ providing it fresh randomness and inputs $\{x_j\}_{j\in I}$ and emulating the actions of the honest parties using the real inputs. The output of the experiment is $\mathbf{REAL}_{\Pi_{\text{MPC}},\mathcal{A}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$ which consists of $\mathcal{A}$'s view and the output of honest parties. Note that the Λ_w values are computed correctly using the inputs $\mathsf{INP}_{\mathsf{BMR}}$ and $\{x_j\}_{j\in\bar{I}}$.

Hybrid H_1. This experiment proceeds identically to H_0 with the following exception: the simulator will try to extract the adversary's defenses 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 the corrupted parties by rewinding the non-malleable commitment made by the 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 C^* for nmcom 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 protocol demands a ZAP proof from the committer ensuring that one of the two commitments is well-formed. Recall that between every pair of parties P_i and P_j, two non-malleable commitments are made by P_i to P_j. For every corrupted party P_i, the simulator chooses an honest party P_j and tries to extract one of the two commitments made by P_i to 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 nmcom^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 $\{\mathsf{wit}_i^*\}_{i\in I}$ as follows:

$$(\mathsf{wit}_i^*) = w_b + \widetilde{w}_b$$

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).

- $\mathcal{S}_1$ 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, by the soundness argument of the ZAP it is also ensured 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 4.2. *The adversary's view in* H_1 *is statistically indistinguishable from its view in* H_0.

It suffices to argue that $\mathcal{S}_1$ runs in expected polynomial time. We claim that if the simulator is able to obtain three transcripts with the same first message but different challenges for the two non-malleable commitments where the ZAP proof in the third proof is convincing, the simulator can extract a valid defense. First we observe that extracting three transcripts where the ZAP proof verifies can be achieved in expected polynomial time. This is because the probability p with which the rewinding is initiated is also the probability with which the ZAP proof is convincing, which implies the expected rewindings to obtain one such transcript is $p \times 1/p = 1$. It only remains to show that obtaining three of them is sufficient to extract a defense. We argue this next.

By the special-spound property of the GRRV commitment scheme (See Sect. 2.5.1), we have that if the simulator obtains two valid transcripts with the same first message of any of the two non-malleable commitments then it can extract a message. However, the simulator cannot identify from a transcript whether the commitment is valid. The soundness of the ZAP proof implies that if the proof is convincing, the transcript is valid for at least one of the two commitments and that the message encoded in that commitment is a valid defense. Now, we observe that if the simulator obtains three transcripts with the same first message where the ZAP proof verifies, then it must be the case that two of those transcripts must be a valid commitment for at least one of the two non-malleable commitment. Then, the simulator can simply try to extract values from both commitments from each of the three pairs of transcripts and validate if any of the extracted value is a valid defense.

Hybrid H_2. This experiment proceeds identically to the previous experiment until the third round after a defense has been extracted. Then, in the fourth round, the messages of the honest parties are generated differently. In H_1 the fourth round messages were computed by 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. Specifically, the simulator

obtains λ_w^j and $T_{w,0}^j$, $T_{w,0}^j \oplus T_{w,1}^j$ for every corrupted party P_j and wire w that is an output of some NAND gate. 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. Using this defense and the inputs and randomness of the honest parties, $\mathcal{S}_2$ will generate the fourth round message on behalf of the honest parties. More precisely, the simulator first determines the $S_{\star,\star}^{\star,\star}$ values using the honest parties inputs and defense provided by the adversary. Then, following the same procedure as in hybrid H_3 in Sect. 3.1 generates the shares in the fourth round for the honest parties.

Following the claim made in hybrid H_3 in Sect. 3.1, we can conclude here that the shares of the honest parties in these two hybrids are identically distributed, conditioned on successfully extracting a valid defense from the adversary. Therefore, we have the following claim.

Claim 4.3. *The adversary's view in* H_2 *is statistically indistinguishable from its view in* H_1.

In the next sequence of hybrids, we will modify the inputs of the honest parties in the different instances of the subprotocol Π_{DMULT} employed in Π_{DMPC}. In these hybrids, we will switch from generating the honest parties' messages in the first three rounds using $\mathsf{INP}_{\mathsf{BMR}}$ to RND where recall that $\mathsf{INP}_{\mathsf{BMR}}$ are the original inputs chosen for the honest party (and consistent with how the fourth message is generated) while RND are the fake inputs chosen by the simulator for the Π_{DPOLY} subprotocol. However, we will continue to generate the fourth round messages using the original inputs $\mathsf{INP}_{\mathsf{BMR}}$ chosen for the honest parties. At the end of the next set of hybrids, we would have decoupled the fourth round message from the first the inputs used in the Π_{DMULT} instances in the first three rounds. Following this we will replace the garbled circuit from being constructed honestly to a simulated garbled circuit following the hybrids in the proof of Lemma 3.6. For ease of comparison with the hybrids in Sect. 3.2, we will denote P_1, P_2 and P_3's input in the multiplication protocol by x_1, x_2, x_3 for inputs chosen according to $\mathsf{INP}_{\mathsf{BMR}}$ and x_1', x_2', x_3' for inputs chosen according to RND.

- First, we consider a sequence of hybrids where in each Π_{DMULT} instance where P_1 is controlled by an honest party, we will modify its action in $\mathsf{C}_\gamma^1[2]$ and $\mathsf{C}_\gamma^2[2]$ so that instead of using u as the input (which was computed from $\mathsf{C}_\alpha^1[2]$, $\mathsf{C}_\alpha^2[2]$), we will use 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_{DMULT} instance where we switch P_1's input in $\mathsf{C}_\alpha^2[1]$ from x_1 to x_1'.
- Following, this we consider a sequence of hybrids we do an analogous change to P_3's input in $\mathsf{C}_\beta^2[1]$ and $\mathsf{C}_\gamma^2[1]$ 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 $\mathsf{INP}_{\mathsf{BMR}}$ to being generated according to RND.

- Then we go through the hybrids in a reverse order where we switch the other witness to correspond to RND. Now, the actions of all honest parties in the first three rounds are consistent with the honest strategy on input RND.

Interlude. The simulation in the next sequence of hybrids will involve a *premature rewinding* phase. In more detail, the simulator stalls the simulation in the main thread after the first round messages are exchanged and proceeds to rewinding in the premature rewinding phase. The purpose of the rewinding is to obtain some trapdoor information before simulating the second and third messages in the main thread. The trapdoor can be extracted from the messages in the non-malleable commitments just as in the previous hybrids. Premature rewinding is problematic, in general, as we do not know the number of times we need to rewind. In the previous hybrids (and in the actual simulation) we rewinded the adversary until we successfully extracted the defences, but employed the extractor only if in the main execution the adversary completes the first three messages of the protocol. Conditioning on this event, namely, not aborting in the third round, allows us to argue that the expected number of rewindings will be polynomial. In premature rewinding however we rewind without such conditioning.

Nevertheless, since we employ premature rewinding only in the intermediate hybrids (and specifically not in the final simulation), we will require to rewind only a fixed polynomial number of times. This is because we can allow a distinguisher dependent simulation in intermediate hybrids. In more detail, by way of contradiction, there is a distinguisher that distinguishes the real and simulated worlds with probability $\frac{1}{p(\kappa)}$. This means that the distinguishing probability cannot be smaller than $\frac{1}{q(\kappa)p(\kappa)}$ for all two intermediate hybrids, where $q(\kappa)$ is the total number of intermediate hybrids. If we now consider a simulation in the intermediate hybrids which cuts off the rewinding phase after some fixed polynomial number of rewinding attempts, then we can argue that the simulation fails only with some small probability. This means that we still have a distinguisher that can distinguish the hybrids with non-negligible probability even conditioned on the simulation failing.

In slight more detail, we will cut off the rewinding phase after $8q(\kappa)p(\kappa)$ steps and argue that the simulation error is bounded by $\frac{1}{4q(\kappa)p(\kappa)}$. Suppose that the adversary does not abort before the third round 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 the probability that the simulator fails to extract in $8q(\kappa)p(\kappa)$ attempts is bounded by $\frac{1}{4q(\kappa)p(\kappa)}$. 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.

Intermediate hybrids with premature rewinding. We consider a sequence of hybrids where we first replace the inputs entered by an *honest* P_1 in the individual Π_{DMULT} instances. Then we proceed to replace the inputs of honest P_2 and P_3. Throughout these hybrids, we employ premature rewinding and generate the fourth round message using the real inputs of the honest parties, namely $\{\mathbf{x}_i\}_{i\in\bar{I}}$.

More formally, for every honest party P_i such that $i \in \bar{I}$ and every monomial M_t such that $\mathsf{Role}(t, i) = 1$, consider the following intermediate experiments.

Hybrid $\mathsf{H}_3^{1,j}$ $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{GRB}; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}})$. This experiment proceeds identically to H_2 with the exception that the non-malleable commitment $\mathsf{nmcom}_{i,j}^1$ is simulated differently. Specifically, the value committed within $\mathsf{nmcom}_{i,j}^1$ by the honest party P_1 in Π_{DMULT}, played by P_i and sent to some (corrupted) party P_j, is switched from w_1 to a random value R. In particular the messages $\widetilde{w}_1$ sent in the third round will not satisfy the condition for which $R + \widetilde{w_1}$ is a valid defense. Next, the simulator performs 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. Upon sending the first message in the main thread, the simulator stalls the main thread and proceeds to rewinding in the premature rewinding phase. As mentioned above, in the premature rewinding the simulator invokes the extractor and guarantees correct extraction with probability at least $1 - \frac{1}{4q(\kappa)p(\kappa)}$. In the rewinding, the simulator generates the third message of $\mathsf{nmcom}_{i,j}^1$ 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. When completing the premature extraction, the simulator proceeds to the main thread and completes the execution until the third round. Then 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.

Claim 4.4. *The adversary's view in* $\mathsf{H}_3^{1,j}$ *is indistinguishable from its view in* H_2.

Proof. The indistinguishability of these experiments follows by a relying on the weak one-many non-malleability against synchronizing adversaries property of the underlying non-malleable commitment scheme. In more detail, we employ the simulation described above with the exception that the non-malleable commitment $\mathsf{nmcom}_{i,j}^1$ will be received from an external committer in the weak one-many non-malleability game. On the right, the simulator forwards all the non-malleable commitments made by the adversary. For the premature rewinding phase, the simulator forwards the first message from this non-malleable commitment internally to generate the first round. Then it performs the premature rewinding. Recall that the input and randomness for $\mathsf{nmcom}_{i,j}^1$ is not used internally in the premature rewinding. After the premature rewinding phase, the simulator continues with the main thread where the second and third messages for the non-malleable commitment $\mathsf{nmcom}_{i,j}^1$ are exchanged with the external challenger and all non-malleable commitments made by the adversary forwarded on the right. Now we employ the extractor provided by the non-malleable commitment to extract the values committed by the adversary from which a valid defense can be extracted. This is used to generate the fourth message. Suppose that the distinguisher distinguishes $\mathsf{H}_3^{1,j}$ from H_2 with probability $\frac{1}{p(\kappa)q(\kappa)}$, then it distinguishes the simulated experiments with prob-

ability at least $\frac{1}{p(\kappa)q(\kappa)} - 2 \cdot \frac{1}{4p(\kappa)q(\kappa)} = \frac{1}{2p(\kappa)q(\kappa)}$. This will contradict the one-many non-malleability property. □

Hybrid $\mathsf{H}_3^{2,j}$ $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{GRB}; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND})$. This experiment proceeds identically to $\mathsf{H}_3^{1,j}$ with the exception that $\tilde{w}_1$ is 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 messages $\mathsf{C}_\gamma^1[2]$ and $\mathsf{C}_\gamma^2[2]$ involving u and s_1. Recall from the proof of Claim 3.3 in Protocol 1 that if we compute $\mathsf{C}_\gamma^1[2]$ and $\mathsf{C}_\gamma^2[2]$ using a random bit u' instead of the actual bit u, the view of the adversary remains statistically indistinguishable using the equivocation property of the encryption and the randomization by s_1. A second defense can therefore be obtained for a random u' by using the equivocation property if the trapdoors are known (which are obtained via premature rewinding).

In more detail, the simulator proceeds honestly using u and s_1 to generate all messages honestly except $\tilde{w}_1$. Instead for $\tilde{w}_1$, the simulator samples random u', computes $s_1' = (u - u')x_3 + s_1$ and generates a defense for $\mathsf{C}_\gamma^1[2]$ and $\mathsf{C}_\gamma^2[2]$ according to u', s_1' from the equivocation trapdoors obtained from the defense provided for P_3. To recap, in this hybrid, 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 $w_1 + \tilde{w}_1$ is the second defense for a random u'. The only difference between the $\mathsf{H}_3^{1,j}$ and $\mathsf{H}_3^{2,j}$ is in how the third message is generated. Using the equivocation property of the underlying encryption scheme and statistical independence of u from the first three messages as proved in Claim 3.3, we have that the simulator always succeeds in obtaining the second witness except with negligible probability conditioned on the premature rewinding succeeding. Furthermore, as we have replaced the non-malleable commitment $\mathsf{nmcom}_{i,j}^1$ to commit to a random string R, w_1 (that is chosen uniformly at random) is independent of the view of the adversary which in turn means $\tilde{w}_1$ revealed in the third message is identically distributed in both the hybrids. Therefore, we have the following claim.

Claim 4.5. *The adversary's view in* $\mathsf{H}_3^{1,j}$ *is statistically indistinguishable from its view in* $\mathsf{H}_3^{2,j}$.

Hybrid $\mathsf{H}_3^{3,j}$. $(\mathsf{INP}_{\mathsf{BMR}}, \mathsf{INP}_{\mathsf{BMR}}; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND})$. In this experiment the simulator proceeds identically as in the previous hybrid, with the exception that it reverts the change made in $\mathsf{H}_3^{1,j}$, 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 receives a commitment to R in the previous hybrid and to w_1 in the current hybrid. We thus 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 4.6. *The adversary's view in* $\mathsf{H}_3^{3,j}$ *is indistinguishable from its view in* $\mathsf{H}_3^{2,j}$.

Hybrid. $\mathsf{H}_3^{4,j}$. This experiment proceeds identically to $\mathsf{H}_3^{3,j}$ 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 indistinguishability 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 4.7. *The adversary's view in* $\mathsf{H}_3^{4,j}$ *is indistinguishable from its view in* $\mathsf{H}_3^{3,j}$.

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 $\mathsf{C}_\alpha^1[2], \mathsf{C}_\alpha^2[2]$.

Hybrid $\mathsf{H}_3^{5,j} - \mathsf{H}_3^{7,j}$. Next, we consider a set of experiments analogous to $\mathsf{H}_3^{1,j}$-$\mathsf{H}_3^{3,j}$, where we will switch the first non-malleable commitment $\mathsf{nmcom}_{i,j}^0$ and $\tilde{w}_0$ such that $w_0 + \tilde{w}_0$ equals to the second witness. Namely, we consider $(\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}}; \mathsf{INP}_{\mathsf{BMR}}, \mathsf{RND})$, $(\mathsf{GRB}, \mathsf{INP}_{\mathsf{BMR}}; \mathsf{RND}, \mathsf{RND})$ and $(-, -; \mathsf{RND}, \mathsf{RND})$ distributions in these hybrids.

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 $\mathsf{C}_\gamma^1[2]$ and $\mathsf{C}_\gamma^2[2]$ according to the randomly chosen value u'.

Hybrid $\mathsf{H}_3^{8,j}$. This experiment is identical to $\mathsf{H}_3^{7,j}$ with the exception that the witness used in the ZAP proof is changed from using $\mathsf{nmcom}_{i,j}^1$ to $\mathsf{nmcom}_{i,j}^0$. Indistinguishability follows directly from the resettable reusable witness indistinguishability of the ZAP as in $\mathsf{H}_3^{4,j}$.

Hybrid $\mathsf{H}_4^{1,j}$. This experiment is identical to hybrid $\mathsf{H}_3^{8,j}$ with the exception that the non-malleable commitment $\mathsf{nmcom}_{i,j}^1$ is simulated differently. First the value committed by the honest party in $\mathsf{nmcom}_{i,j}^1$ 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 $\mathsf{nmcom}_{i,j}^1$. 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 $\mathsf{H}_4^{2,j}$. Identical to $\mathsf{H}_4^{1,j}$ except that we remove the inputs on behalf of the honest party P_i (acting as P_1) from $\mathsf{C}_\alpha^1[1]$ from x_1 to a x_1'. Indistinguishability follows from the semantic security of the encryption scheme as in Hybrid H_6 in Sect. 3.2. Note that the sequence of $\mathsf{H}_3^{\star,\star}$ 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 $\mathsf{H}_4^{3,j}$. This experiment proceeds identically to $\mathsf{H}_4^{2,j}$ 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 messages $\mathsf{C}_\alpha^1[1]$ and $\mathsf{C}_\alpha^2[1]$, namely x_1' used in $\mathsf{C}_\alpha^2[1]$. Indistinguishability follows essentially as in hybrid $\mathsf{H}_3^{2,j}$.

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

At the end of hybrid $\mathsf{H}_4^{4,j}$, we have changed the inputs of P_1 for every multiplication instance where P_1 is controlled by an honest party from what was chosen in the real execution x_1 to a random input x_1'.

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 $\mathsf{H}_3^{1,j}$ to $\mathsf{H}_3^{3,j}$ where we set up $\mathsf{nmcom}_{i,j}^1$ and $\tilde{w}_1$ such that $w_1 + \tilde{w}_1$ contains a defense for x_2'. This sequence is simpler as we do not have to decouple any locally computed value such as u from the simulation before we change the input. We will rely on the equivocation property of the encryption scheme, the same way as we did for P_1 to obtain the second defense.

Following this sequence, we consider the scenario where P_3 is controlled by an honest party. The sequence hybrids involves replacing its input from x_3 to x_3' will be analogous to the sequence $\mathsf{H}_4^{1,j}$ to $\mathsf{H}_4^{4,j}$.

Now, we have that $w_1 + \tilde{w}_1$ is a defense according to RND and $w_0 + \tilde{w}_0$ is a defense according to $\mathsf{INP}_{\mathsf{BMR}}$. Now we consider a hybrid where we switch the witness in the ZAP statement from a defense according to $\mathsf{INP}_{\mathsf{BMR}}$ to a defense according to RND. We replace the ZAP witness for both the proofs that conclude in the second step as well as the third step. Indistinguishability will rely on the reusable resettable WI property of the ZAP.

Next, we consider a sequence of hybrids in a reverse order so that $w_0 + \tilde{w}_0$ is a valid defense for RND followed by switching the ZAP witness using $w_0, \tilde{w}_0$.

Conclusion. This completes our hybrids that relies on premature rewinding phase. Now, we have switched the inputs and actions of the honest party in the first three rounds from using $\mathsf{INP}_{\mathsf{BMR}}$ to RND. The next sequence of hybrids involves changing the simulation of the Λ_w values and the fourth messages. Thus far, these message have been correctly according to the inputs $\mathsf{INP}_{\mathsf{BMR}}$ for the honest party, namely, the garbled circuit and the input encoding are constructed according to $\mathsf{BMR.Encode}^{\widehat{C}}$. The next set of hybrids will involve switching these messages from the honest party using SimGarble and then relying on ReconGarble to determine whether the outputs should be delivered to the honest parties in the ideal world. We simply repeat the hybrids $\mathsf{H}_2^g, \mathsf{H}_3, \mathsf{H}_4, \mathsf{H}_5$ from the proof of protocol Π_{DMPC} and the proof of indistinguishability follows identically. Once we execute these hybrids, our final hybrid models the real simulator and this concludes the proof. □

A. Secure Multi-party Computation

We briefly present the standard definition for secure multi-party computation and refer to [33, 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 re-

ceives $f_i(x_1, \ldots, x_n)$. 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 presence of an adversary that may actively (i.e. maliciously) or passively (i.e. semi-honest) corrupt parties. 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

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

where κ is the security parameter.

A.2. *The Active 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 active 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 $I \subset [n]$,

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

where κ is the security parameter.

References

[1] P. Ananth, A.R. Choudhuri, A. Goel, A. Jain, Round-optimal secure multiparty computation with honest majority, in *CRYPTO* (2018), pp. 395–424

[2] P. Ananth, A.R. Choudhuri, A. Jain, A new approach to round-optimal secure multiparty computation, in *CRYPTO* (2017), pp. 468–499

[3] B. Applebaum, Y. Ishai, E. Kushilevitz, Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162 (2006)

[4] B. Applebaum, Y. Ishai, E.Kushilevitz, Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888 (2006)

[5] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, D. Wichs, Multiparty computation with low communication, computation and interaction via threshold FHE, in *EUROCRYPT* (2012), pp. 483–501

[6] B. Barak, How to go beyond the black-box simulation barrier, in *FOCS* (2001), pp. 106–115

[7] S. Badrinarayanan, V. Goyal, A. Jain, D. Khurana, A. Sahai, Round optimal concurrent MPC via strong simulation, in *TCC* (2017), pp. 743–775

[8] S. Badrinarayanan, V. Goyal, A. Jain, Y.T. Kalai, D. Khurana, A. Sahai, Promise zero knowledge and its applications to round optimal MPC, in *CRYPTO* (2018), pp. 459–487

[9] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract), in *STOC* (1988), pp. 1–10

[10] Z. Brakerski, S. Halevi, A. Polychroniadou, Four round secure computation without setup, in *TCC* (2017), pp. 645–677
[11] F. Benhamouda, H. Lin, k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits, in *EUROCRYPT* (2018), pp. 500–532
[12] D. Beaver, S. Micali, P. Rogaway, The round complexity of secure protocols (extended abstract), in *STOC* (1990), pp. 503–513
[13] D. Chaum, C. Crépeau, I. Damgård, Multiparty unconditionally secure protocols (abstract), in *CRYPTO* (1987), p. 462
[14] A.R. Choudhuri, M. Ciampi, V. Goyal, A. Jain, R. Ostrovsky, On round optimal secure multiparty computation from minimal assumptions. *IACR Cryptol. ePrint Arch.*, 2019:216 (2019)
[15] R. Cramer, Y. Dodis, S. Fehr, C. Padró, D. Wichs, Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors, in *EUROCRYPT* (2008), pp. 471–488
[16] M. Ciampi, R. Ostrovsky, L. Siniscalchi, I. Visconti, Concurrent non-malleable commitments (and more) in 3 rounds, in *CRYPTO* (2016), pp. 270–299
[17] M. Ciampi, R. Ostrovsky, L. Siniscalchi, I. Visconti, Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds, in *TCC 2017* (2017)
[18] M. Ciampi, R. Ostrovsky, L. Siniscalchi, I. Visconti. Round-optimal secure two-party computation from trapdoor permutations, in *TCC* (2017), pp. 678–710
[19] I. Damgård, Y. Ishai, Constant-round multiparty computation using a black-box pseudorandom generator, in *CRYPTO* (2005), pp. 378–394
[20] I. Damgård, Y. Ishai, Scalable secure multiparty computation, in *CRYPTO* (2006), pp. 501–520
[21] I. Damgård , M. Jurik, A generalisation, a simplification and some applications of Paillier's probabilistic public-key system, in *Public Key Cryptography* (2001), pp. 119–136
[22] C. Dwork, M. Naor, Zaps and their applications. *SIAM J. Comput.*, 36(6):1513–1543 (2007)
[23] T. El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472 (1985)
[24] S. Garg, C. Gentry, S. Halevi, M. Raykova, Two-round secure MPC from indistinguishability obfuscation, in *TCC* (2014), pp. 74–94
[25] D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, E. Tromer, Circuits resilient to additive attacks with applications to secure computation, in *STOC* (2014), pp. 495–504
[26] D. Genkin, Y. Ishai, A. Polychroniadou, Efficient multi-party computation: From passive to active security via secure SIMD circuits, in *CRYPTO* (2015), pp. 721–741
[27] D. Genkin, Y. Ishai, M. Weiss, Binary amd circuits from secure multiparty computation, in *TCC* (2016), pp. 336–366
[28] S. Garg, S. Kiyoshima, O. Pandey, On the exact round complexity of self-composable two-party computation, in *EUROCRYPT* (2017), pp. 194–224
[29] S. Goldwasser, S, Micali, Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299 (1984)
[30] S. Garg, P. Mukherjee, O. Pandey, A. Polychroniadou, The exact round complexity of secure computation, in *EUROCRYPT* (2016), pp. 448–476
[31] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or A completeness theorem for protocols with honest majority, in *STOC*, (1987) pp. 218–229
[32] O. Goldreich, *Foundations of Cryptography: Basic Tools*. Cambridge University Press (2001)
[33] O. Goldreich, *Foundations of Cryptography: Basic Applications*. Cambridge University Press (2004)
[34] V. Goyal, Constant round non-malleable protocols using one way functions, in *STOC* (2011), pp. 695–704
[35] V. Goyal, S. Richelson, A. Rosen, M. Vald, An algebraic approach to non-malleability, in *FOCS* (2014), pp. 41–50
[36] S. Garg, A. Srinivasan, Two-round multiparty secure computation from minimal assumptions, in *EUROCRYPT* (2018), pp. 468–499
[37] I. Haitner, Y. Ishai, E. Kushilevitz, Y. Lindell, E. Petrank, Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266 (2011)
[38] C. Hazay, A. Polychroniadou, M. Venkitasubramaniam, Composable security in the tamper-proof hardware model under minimal complexity, in *TCC* (2016), pp. 367–399
[39] C. Hazay, P. Scholl, E. Soria-Vazquez, Low cost constant round MPC combining BMR and oblivious transfer, in *ASIACRYPT* (2017), pp. 598–628

[40] R. Impagliazzo, L.A. Levin, M. Luby, Pseudo-random generation from one-way functions (extended abstracts), in *STOC* (1989), pp. 12–24
[41] D. Khurana, Round optimal concurrent non-malleability from polynomial hardness, in *TCC* (2017), pp. 139–171
[42] J. Katz, R. Ostrovsky, A.D. Smith, Round efficiency of multi-party computation with a dishonest majority, in *EUROCRYPT* (2003), pp. 578–595
[43] H. Lin, R. Pass, Constant-round non-malleable commitments from any one-way function, in *STOC* (2011), pp. 705–714
[44] Y. Lindell, B. Pinkas, N.P. Smart, A. Yanai, Efficient constant round multi-party computation combining BMR and SPDZ, in *CRYPTO*, pp. 319–338, (2015)
[45] P. Mukherjee, D. Wichs, Two round multiparty computation via multi-key FHE, in *EUROCRYPT* (2016), pp. 735–763
[46] M. Naor, Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158 (1991)
[47] R. Ostrovsky, A. Paskin-Cherniavsky, B. Paskin-Cherniavsky, Maliciously circuit-private FHE, in *CRYPTO* (2014), pp. 536–553
[48] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in *EUROCRYPT* (1999), pp. 223–238
[49] R. Pass, Bounded-concurrent secure multi-party computation with a dishonest majority, in *STOC* (2004), pp. 232–241
[50] O. Regev, On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40 (2009)
[51] A.C.C. Yao, How to generate and exchange secrets (extended abstract), in *FOCS* (1986), pp. 162–167