Round-Optimal Secure Multi-Party Computation

Shai Halevi[1], Carmit Hazay[1], Antigoni Polychroniadou[3], and Muthuramakrishnan Venkitasubramaniam[4]

[1] IBM Research
shaih@alum.mit.edu,
[2] Bar-Ilan University
carmit.hazay@biu.ac.il,
[3] Cornell Tech and University of Rochester
antigoni@cornell.edu,
[4] University of Rochester
muthuv@cs.rochester.edu

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

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 [Yao86, CCD87, GMW87, BGW88].

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

In [GMPP16], Garg, et al., proved a lower bound of four rounds for MPC protocols that relies on black-box simulation. Following this work, in independent works, Ananth, Choudhuri and Jain [ACJ17] and Brakerski, Halevi and Polychroniadou, [BHP17] showed a matching upper bound by constructing 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 [COSV17b] closed the gap for two-party protocols by constructing a four-round protocol from standard polynomial-time assumptions. The same authors in another work [COSV17a] 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:

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

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. [COSV17b] 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 [GGHR14, MW16] or three-round assuming tamper-proof hardware tokens [HPV16].[5] In the case of passive adversaries, (or even the slightly stronger setting of semi-malicious[6] adversaries) three round protocols based on the Learning With Errors assumption have been constructed by Brakerski et al. [BHP17]. Ananth et al. gave a five-round protocol based on DDH [ACJ17]. Under subexponential hardness assumptions, four-round constructions were demonstrated in [BHP17, ACJ17]. Under some relaxations of superpolynomial simulation, the work of Badrinarayanan et al. [BGJ$^+$17] 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. [COSV17b]. Finally, if we assume an honest majority, the work of Damgard and Ishai [DI05] provided a three-round MPC protocol. If we allow trusted setup (i.e. not the plain model) then a series of works [CLOS02, **?**, MW16, BL18, GS17] have shown how to achieve two-round multiparty computation protocols in the common reference string model under minimal assumptions. In the tamper proof setup model, the work of [HPV16] show how to achieve three round secure multiparty computation assuming only one-way functions.

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

[6] 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.

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, 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 multiparty computation protocols. 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 are required to instantiate the non-malleable commitment scheme from [GRRV14] and can be instantiated using the QR. In summary, all our primitives can be instantiated by the single QR assumptions and therefore we have the following corollary

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.*

1.2 Our Techniques

Starting point: the [ACJ17] protocol. We begin from the beautiful work of Ananth, Choudhuri and Jain [ACJ17], where they used randomized encoding [AIK06] 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 oblivious-transfer subprotocol, as illustrated in Figure 1.

Using this three-round multiplication subprotocol, Ananth et al. constructed a four-round protocol for the semi-honest model, then enforced correctness in the third and fourth rounds using zero-knowledge proofs to get security against a malicious adversary. In particular, the proof of correct behavior in the third round required a special

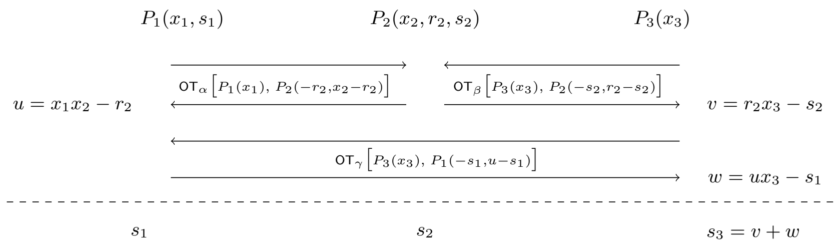
$P_1(x_1, s_1)$
$P_2(x_2, r_2, s_2)$
$P_3(x_3)$
$\mathsf{OT}_\alpha\left[P_1(x_1),\ P_2(-r_2, x_2-r_2)\right]$
$\mathsf{OT}_\beta\left[P_3(x_3),\ P_2(-s_2, r_2-s_2)\right]$
$u = x_1x_2 - r_2$
$v = r_2x_3 - s_2$
$\mathsf{OT}_\gamma\left[P_3(x_3),\ P_1(-s_1, u-s_1)\right]$
$w = ux_3 - s_1$
s_1
s_2
$s_3 = v + w$

Fig. 1. The three-bit multiplication protocol from [ACJ17], using two-round oblivious transfer. The OT sub-protocols 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. The sum of s_1, s_2, s_3 results into the output $x_1x_2x_3$.

three-round non-malleable zero-knowledge proof, 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 proofs (aka ZAPs [DN07]).

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.[7]

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 the 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

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

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.[8] Thus we must rearrange the simulater 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 Figure 1, there is only one place where a value received in one of the OTs is used before the end of the third round. To wit, 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.

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 a random bit, and argue that these hybrids are close by reduction to the OT receiver privacy. Inside the reduction, we will have no access to the values 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. [9]

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.).

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 [GIP+14, GIP15, GIW16]. Namely, they described a

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

[9] 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.

compiler that transforms an arbitrary circuit C to another circuit C$'$ that is resilient to additive attacks. Unfortunately, using these transformations does 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 encoding 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. In a nutshell, we need distributed randomized encoding that has offline (input independent) and online (input dependent) procedures that satisfies the following three conditions:

- The offline encoding has degree-3 (in the randomness);
- The online procedure is decomposable (encodes each bit separately);
- The offline procedure is resilient to additive attacks on the internal wires of the computation.

As such the encoding procedure in [AIK06] does not meet these conditions.

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 Rogaway [BMR90] (referred to as "BMR encoding") and show how it can be massaged to satisfy the required properties.[10] 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 3.2. 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 [AIK06] is that it can be instantiated based on any one-way function. In contrast the randomized encoding of [AIK06] requires the assumption of PRGs in NC^1.

A Sketch of the Final Protocol Combining all these ideas, our (almost) final protocol proceeds as follows: Let C be a circuit that we want to evaluate securely, we first apply to it the transformation of Genkin et al. to get resilience against additive attacks, then apply BMR encoding to the result. This gives us a randomized encoding for our original circuit C. We use the fact that the BMR encoding has the form $\mathrm{C}_{\mathsf{BMR}}(x;(\lambda,\rho)) = (x \oplus \lambda, g(\lambda,\rho))$ where each output bit of g has degree three (or less) in the (λ,ρ). Given the inputs $x = (x_1,\ldots,x_n)$, the parties choose their 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 Figure 1), to compute the outputs of $g(\lambda,\rho)$.

[10] We remark that our BMR encoding differs from general randomized encoding as we allow some "local computation" on the inputs before it is fed into the offline encoding procedure.

In addition to the third round message of Π', each party P_i also broadcasts its masked input $x_i \oplus \lambda^i$.

Let wit_i be a witness of "correct behavior" of party P_i in Π' (where the witness of an OT-receiver includes the randomness for only one of the two instances in an OT pair). In parallel with the execution of Π', each party P_i also engages in three-round non-malleable commitment protocols for wit_i, and two-round ZAP proofs that wit_i is indeed a valid witness for "correct behavior" (in parallel to rounds 2,3). Once all the proofs are verified, the parties broadcast their final messages s_i in the protocol Π', allowing them to complete the computation of the encoding output $g(\lambda, \rho)$. They now all have the BMR encoding $\mathrm{C}_{\mathsf{BMR}}(x; (\lambda, \rho))$, so they can locally apply the corresponding BMR decoding procedure to compute $\mathrm{C}(x)$.

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 [Khu17] shows how to construct such commitments in three rounds. However, our proof approach demands additional properties of the underlying non-malleable commitment, but 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 [GRRV14, Khu17] 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 yeilding 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. [COSV16] for making non-malleable commitments into input-delayed ones. Finally, we observe that we can instantiate the "weak simulation extractable non-malleable commitments" that we need from the three-round non-malleable commitment scheme implicit in the work of Goyal et al. [GRRV14].

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).

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.

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 [ACJ17]. 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 can not violate privacy or correctness. It was observed in prior work [LPSY15] 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 [HSS17], 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 a "defensible" adversary. 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" adversary. 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.[11]

This model is somewhat similar to the semi-malicious adversary model of Asharov et al. [AJL$^+$12] 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 [ACJ17], where, if a protocol is secure against defensible adversaries 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" adversaries merely as a convenience to make the presentation more modular, we believe that it is a useful tool for obtaining round-efficient protocols.

1.3 Related and Concurrent Work

The earliest MPC protocol is due to Goldreich, Micali and Wigderson [GMW87]. The round complexity of this approach is proportional to the circuit's multiplication depth (namely, the largest number of multiplication gates in the circuit on any path from input to output) and can be non-constant for most functions. In Table 1, we list relevant prior works that design secure multiparty computation for arbitrary number parties in the stand-alone plain model emphasizing on the works that have improved the round complexity or cryptographic assumptions.

In concurrent work, simultaneously Benhamouda and Lin [BL18] and Garg and Srinivasan [GS17] construct a five-round MPC protocol based on minimal assumptions. While these protocols rely on the minimal assumption of 4-round OT protocol, they require an additional round to construct their MPC.

[11] The name "defensible adversaries" is adapted from the work of Haitner et al. [HIK$^+$11].

Protocol	Functionality	Round	Assumptions	Sub-exponential
[BMR90, KOS03]	General	$O(1)$	CRHF, ETDP	Yes
[Pas04]	General	O(1)	CRHF, ETDP	No
[PW10]	General	O(1)	ETDP	Yes
[LP11, Goy11]	General	O(1)	ETDP	No
[LPV12]	General	O(1)	OT	No
[GMPP16]	General	6	LWE	Yes
		5	iO	Yes
[ACJ17]	General	5	DDH	No
		4	DDH	Yes
[BHP17]	General	4	LWE	Yes
[COSV17b]	Coin Tossing	4	ETDP	No

Table 1. Prior works that design secure computation protocols for arbitrary number of parties in the plain model where we focus on constant round constructions.

In another concurrent work, Badrinarayanan et al. [BGJ+18] establish the main feasibility result presented in this work, albeit with different techniques and slightly different assumptions. Their work compiles the semi-malicious protocol of [BL18, GS17] while we build on modified variants of BMR and the 3-bit multiplication due to [ACJ17]. Both works rely on injective OWFs, and whereas we also need ZAPs and affine homomorphic encryption scheme, they also need dense cryptosystems and two-round OT.

2 Preliminaries

2.1 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 2.1 (Affine homomorphic PKE) *We say that a public key encryption scheme* $(\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* affine homomorphic *if*

- Affine transformation: *There exists an algorithm* AT *such that for every* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$, $m \in \mathcal{M}_\kappa$, $r_c \leftarrow \mathcal{D}_{rand}(1^\kappa)$ *and every* $a, b \in \mathcal{M}_\kappa$, $\mathsf{Dec}_{\mathrm{SK}}(\mathsf{AT}(\mathrm{PK}, c, a, b)) = am + b$ *holds with probability 1, and* $c = \mathsf{Enc}_{\mathrm{PK}}(m; r_c)$, *where* $\mathcal{D}_{rand}(1^\kappa)$ *is the distribution of randomness used by* Enc.
- Equivocation: *There exists an algorithm* $\mathsf{Explain}$ *such that for every* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$, *every* $m, a_0, b_0, a_1, b_1 \in \mathcal{M}_\kappa$ *such that* $a_0 m + b_0 = a_1 m + b_1$ *and every* $r_c \leftarrow \mathcal{D}_{rand}(1^\kappa)$, *it holds that the following distributions are statistically close over* $\kappa \in \mathbb{N}$*:*

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

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

In the full version [HHPV17], we demonstrate how to meet Definition 2.1 under a variety of hardness assumptions.

Definition 2.2 (Resettable reusable WI argument) *We say that a two-message delayed-input interactive argument (P, V) for a language L is* resettable reusable witness indistinguishable*, if for every PPT verifier V^*, every $z \in \{0,1\}^*$, $Pr[b = b'] \leq 1/2 + \mu(\kappa)$ in the following experiment, where we denote the first round message function by $m_1 = \mathsf{wi}_1(r_1)$ and the second round message function by $\mathsf{wi}_2(x, w, m_1, r_2)$. The challenger samples $b \leftarrow \{0,1\}$. V^* (with auxiliary input z) specifies $(m_1^1, x^1, w_1^1, w_2^1)$ where w_1^1, w_2^1 are (not necessarily distinct) witnesses for x^1. V^* then obtains second round message $\mathsf{wi}_2(x^1, w_b^1, m_1^1, r)$ generated with uniform randomness r. Next, the adversary specifies arbitrary $(m_1^2, x^2, w_1^2, w_2^2)$, and obtains second round message $\mathsf{wi}_2(x^2, w_b^2, m_1^2, r)$. This continues $m(\kappa) = \mathrm{poly}(\kappa)$ times for a-priori unbounded m, and finally V^* outputs b.*

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

2.2 Additive Attacks and AMD Circuits

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

Definition 2.3 (AMD code [CDF$^+$08]) *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 $\mathbf{x} \in \mathbb{F}^n$,*

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

– 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*

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

Definition 2.4 (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 2.5 (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* $(\mathbf{x}, \mathbf{A})$*,* $f^{\mathbf{A}}(\mathbf{x}, \mathbf{A})$ *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* $\mathbf{x}$.

Definition 2.6 (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 2.7 ([GIW16], Theorem 2) *For any boolean circuit* $\mathrm{C} : \{0,1\}^n \to \{0,1\}^m$*, and any security parameter* κ*, there exists a* $2^{-\kappa}$*-additively-secure implementation* $\widehat{\mathrm{C}}$ *of* C*, where* $|\widehat{\mathrm{C}}| = \mathsf{poly}(|\mathrm{C}|, n, \kappa)$*. Moreover, given any additive attack* $\mathbf{A}$ *and input* $\mathbf{x}$*, it is possible to identify* a^{In} *such that* $\widehat{\mathrm{C}}^{\mathbf{A}}(\mathbf{x}) = f(\mathbf{x} + a^{\mathrm{In}})$.

Remark 2.1. Genkin et al. [GIW16] 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. [GIW16] 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 $\mathsf{Decode}(\mathbf{x}_{\mathsf{Encode}} + \mathbf{A}_{\mathrm{In}}) - \mathbf{x}$ where $\mathbf{x}_{\mathsf{Encode}}$ 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 ERROR.

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 $I \subset [3]$ 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 $(\mathsf{deliver}, e_{\text{Out}})$ from $\mathcal{S}$, the functionality sends $y + e_{\text{Out}}$ to all parties.

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

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 Section 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 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. 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 Figure 2.

Our protocol relies on an equivocable affine-homomorphic-encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{AT}, \mathsf{Explain})$ (over $\mathbb{F}_2$) as per Definition 2.1, and an additive secret sharing scheme $(\mathsf{Share}, \mathsf{Recover})$ for sharing 0. The details of our protocol are as follows. We usually assume that randomness is implicit in the encryption scheme, unless specified explicitly. See Figure 3 for a high level description of protocol Π_{DMULT}.

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

Input & Randomness: *Parties P_1, P_2, P_3 are given inputs $(x_1, e_{\text{In}}), x_2, x_3$, 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).*

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* $\big((\mathrm{PK}_\beta^1, \mathsf{C}_\beta^1[1]), (\mathrm{PK}_\beta^2, \mathsf{C}_\beta^2[1])\big)$ *(to be used by P_2), and* $(\mathrm{PK}_\gamma^1, \mathrm{PK}_\gamma^2)$ *(to be used in round 3 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 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 use 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 encrypt* $\mathsf{C}_\gamma^1[1] := \mathsf{Enc}_{\mathrm{PK}_\gamma^1}(x_3)$ *and* $\mathsf{C}_\gamma^2[1] := \mathsf{Enc}_{\mathrm{PK}_\gamma^2}(x_3)$ *and broadcast* $(\mathsf{C}_\gamma^1[1], \mathsf{C}_\gamma^2[1])$ *(to be used by P_1).*

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*

$$\mathsf{trans} \stackrel{\text{def}}{=} \begin{pmatrix} \mathrm{PK}_a^1, \mathsf{C}_\alpha^1[1], \mathsf{C}_\alpha^1[2], \mathrm{PK}_a^2, \mathsf{C}_\alpha^2[1], \mathsf{C}_\alpha^2[2],\ \mathrm{PK}_\beta^1, \mathsf{C}_\beta^1[1], \mathsf{C}_\beta^1[2], \mathrm{PK}_\beta^2, \mathsf{C}_\beta^2[1], \mathsf{C}_\beta^2[2], \\ \mathrm{PK}_\gamma^1, \mathsf{C}_\gamma^1[1], \mathsf{C}_\gamma^1[2], \mathrm{PK}_\gamma^2, \mathsf{C}_\gamma^2[1], \mathsf{C}_\gamma^2[2] \end{pmatrix}$$

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

$$\mathsf{trans}_{P_2}^0 = \mathsf{trans}_{P_2}^1 \stackrel{\text{def}}{=} \Big(\mathsf{C}_\alpha^1[2], \mathsf{C}_\alpha^2[2],\ \mathsf{C}_\beta^1[2], \mathsf{C}_\beta^2[2]\ \Big)$$

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

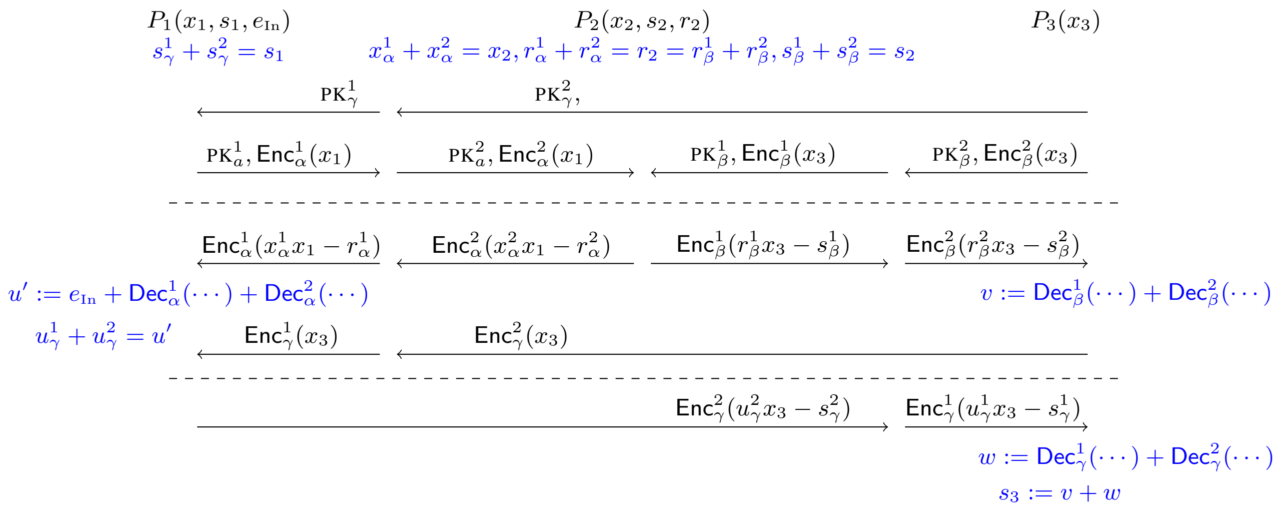
$P_1(x_1, s_1, e_{\mathrm{In}})$
$s^1_\gamma + s^2_\gamma = s_1$
$P_2(x_2, s_2, r_2)$
$x^1_\alpha + x^2_\alpha = x_2, r^1_\alpha + r^2_\alpha = r_2 = r^1_\beta + r^2_\beta, s^1_\beta + s^2_\beta = s_2$
$P_3(x_3)$
PK^1_γ
$\mathrm{PK}^2_\gamma,$
$\mathrm{PK}^1_\alpha, \mathsf{Enc}^1_\alpha(x_1)$
$\mathrm{PK}^2_\alpha, \mathsf{Enc}^2_\alpha(x_1)$
$\mathrm{PK}^1_\beta, \mathsf{Enc}^1_\beta(x_3)$
$\mathrm{PK}^2_\beta, \mathsf{Enc}^2_\beta(x_3)$
$\mathsf{Enc}^1_\alpha(x^1_\alpha x_1 - r^1_\alpha)$
$\mathsf{Enc}^2_\alpha(x^2_\alpha x_1 - r^2_\alpha)$
$\mathsf{Enc}^1_\beta(r^1_\beta x_3 - s^1_\beta)$
$\mathsf{Enc}^2_\beta(r^2_\beta x_3 - s^2_\beta)$
$u' := e_{\mathrm{In}} + \mathsf{Dec}^1_\alpha(\cdots) + \mathsf{Dec}^2_\alpha(\cdots)$
$v := \mathsf{Dec}^1_\beta(\cdots) + \mathsf{Dec}^2_\beta(\cdots)$
$u^1_\gamma + u^2_\gamma = u'$
$\mathsf{Enc}^1_\gamma(x_3)$
$\mathsf{Enc}^2_\gamma(x_3)$
$\mathsf{Enc}^2_\gamma(u^2_\gamma x_3 - s^2_\gamma)$
$\mathsf{Enc}^1_\gamma(u^1_\gamma x_3 - s^1_\gamma)$
$w := \mathsf{Dec}^1_\gamma(\cdots) + \mathsf{Dec}^2_\gamma(\cdots)$
$s_3 := v + w$

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.

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

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

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

$$\mathcal{L}_{P_3} = \left\{ \mathsf{trans} \;\middle|\; \begin{array}{l} \exists\, (x_3, \rho_\beta, \mathrm{SK}_\beta, \sigma_\beta, \rho_\gamma, \mathrm{SK}_\gamma, \sigma_\gamma) \\ s.t. \begin{pmatrix} (\mathrm{PK}^1_\beta, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}^1_\beta[1] = \mathsf{Enc}_{\mathrm{PK}^1_\beta}(x_3; \sigma_\beta)) \\ \vee\, (\mathrm{PK}^2_\beta, \mathrm{SK}_\beta = \mathsf{Gen}(\rho_\beta) \wedge \mathsf{C}^2_\beta[1] = \mathsf{Enc}_{\mathrm{PK}^2_\beta}(x_3; \sigma_\beta)) \end{pmatrix} \\ \quad \wedge \begin{pmatrix} (\mathrm{PK}^1_\gamma, \mathrm{SK}_\gamma = \mathsf{Gen}(\rho_\gamma) \wedge \mathsf{C}^1_\gamma[1] = \mathsf{Enc}_{\mathrm{PK}^1_\gamma}(x_3; \sigma_\gamma)) \\ \vee\, (\mathrm{PK}^2_\gamma, \mathrm{SK}_\gamma = \mathsf{Gen}(\rho_\gamma) \wedge \mathsf{C}^2_\gamma[1] = \mathsf{Enc}_{\mathrm{PK}^2_\gamma}(x_3; \sigma_\gamma)) \end{pmatrix} \end{array} \right\} \quad (3)$$

ROUND 4:

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

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

Lemma 3.1 *Protocol* Π_{DMULT} *securely realizes the functionality* $\mathcal{F}^{\mathbf{A}}_{\mathrm{MULT}}$ *(cf. Figure 2) in the presence of a "defensible adversary" that always broadcasts valid defense at the end of the third round.*

Proof. We first show that the protocol is correct with a benign adversary. Observe that $u' = e_{\text{In}} + x_1(x_\alpha^1 + x_\alpha^2) - (r_\alpha^1 + r_\alpha^2) = e_{\text{In}} + x_1x_2 - r_2$, and similarly $v = x_3r_2 - s_2$ and $w = x_3u' - s_1$. Therefore,

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

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 $x_1, e_{\text{In}}, x_2, x_3$, 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 random inputs for each honest party (denote these values by $\widehat{x}_i$), and then follows the honest protocol execution using these random 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 $\mathsf{C}_\star^{\mathsf{i}}[j]$, 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 $\mathsf{C}_\beta^{\mathsf{i}}[1]$'s and one of the $\mathsf{C}_\gamma^{\mathsf{i}}[1]$'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 $\mathsf{C}_\alpha^{\mathsf{i}}[1]$'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 $v := r_2x_3^* - s_2$, $w := ux_3^* - s_1$ 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 $e_{\rm Out} := y - \sum_j \tilde{S}_j$ where $\tilde{S}_j$ are the values that were broadcast in the fourth round. The simulator finally sends $(\mathsf{deliver}, e_{\rm Out})$ 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 two distributions $\mathbf{REAL}_{\Pi_{\rm DMULT},\mathcal{A}(z),I}(\kappa,(x_1,e_{\rm In}),x_2,x_3)$ and $\mathbf{IDEAL}_{\mathcal{F}^{\mathbf{A}}_{\rm MULT},\mathcal{S}(z),I}(\kappa,(x_1,e_{\rm In}),x_2,x_3)$ are indistinguishable. We argue this via a standard hybrid argument. We provide a brief sketch below.

High-level sketch of the proof. On a high-level, in the first two intermediate hybrids, we modify the fourth message of the honest parties to be generated using the defense and the inputs chosen for the honest parties, rather than the internal randomness and values obtained in the first three rounds of the protocol. Then in the next hybrid below we modify the messages S_i that are broadcast in the last round. In the hybrid following this, we modify P_3 to use fake inputs instead of its real inputs where indistinguishability relies on the semantic security of the underlying encryption scheme. In the next hybrid, the value u is set to random u' rather than the result of the computation using $\mathsf{C}^2_\alpha[1]$ and $\mathsf{C}^2_\alpha[2]$. This is important because only then we carry out the reduction for modifying P_1's input.Indistinguishability follows from the equivocation property of the encryption scheme. Then we modify the input x_1 and indistinguishability relies on the semantic security. Then, we modify the input of P_2 from real to fake which again relies on the equivocation property. Finally we modify the S_i's again to use the output from the functionality $\mathcal{F}^{\mathbf{A}}_{\rm MULT}$ which is a statistical argument and this is the ideal world. A formal proof appears in the full version [HHPV17].

Between Defensible and Real Security. In Section 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, we relied in the proof 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 $\mathsf{C}^{1,2}_\alpha[1]$ to P_2, then learn both $x^{1,2}_\alpha$ (and hence x_2) from P_2's reply.

One way of addressing this concern would be to rely on maliciously secure encryption (as defined in [OPP14]), but this is a strong requirement, much harder to realize than our Definition 2.1. 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:

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

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

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.

3.2 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 $\mathbf{z}_i = (z^1_i, \ldots, z^n_j) \leftarrow \mathsf{Share}(0, n)$, and sends z^j_i 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

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

where M denotes the number of degree-3 monomials. 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.

3.3 Step 3: Arbitrary Functionalities

We recall from the works of [BMR90, DI06, LPSY15] that securely realizing arbitrary functionalities f can be reduced to securely realizing the "BMR-encoding" of the Boolean circuit C that computes f. Our starting point is the observation that the BMR encoding of a Boolean circuit C 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 $\mathcal{F}^{\mathbf{A}}_{\text{MULT}}$), 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 $\{\mathsf{MAC}_\alpha\}$: This will be required to protect the output translation tables from being manipulated by a rushing adversary. Namely, each party contributes a MAC key and along with the output of the function its authentication 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 2.3): 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 2.2): This will be required to protect the computation of the internal wire values from an additive attack by the adversary. Recall from Section 3.1 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 of the BMR encoding. The parties broadcast all keys in a masked format, namely, $h, h(T) \oplus k$ for a random string T, key k and hash function h. 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 fucntion

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

Let C be a Boolean circuit that computes f_2.

3. Next we apply the transformations of Genkin et al. [GIP+14, GIW16] 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 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 $\{\lambda_w^j, k_{w,0}^j, k_{w,1}^j, T_{w,0}^j, T_{w,1}^j, h_{w,0}^j, h_{w,1}^j\}_{w \in [W]}$ 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 $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$ on input $((x_1, R_1), ..., (x_n, R_n))$ outputs

$$\left\{ \begin{array}{ll} (R_{00}^{g,j}, R_{01}^{g,j}, R_{10}^{g,j}, R_{11}^{g,j})_{g \in [G], j \in [n], r_1, r_2 \in \{0,1\}} & \texttt{// Garbled Tables} \\ (h_{w,b}^j, \Gamma_{w,b}^j)_{w \in [W], j \in [n], b \in \{0,1\}}, & \texttt{// masked key values} \\ (\Lambda_w, k_{w,\Lambda_w}^1, \ldots, k_{w,\Lambda_w}^n)_{w \in \mathsf{Inp}}, & \texttt{// keys and masks for input wires} \\ (\lambda_w)_{w \in \mathsf{Out}} & \texttt{// Output translation table} \end{array} \right\}$$

where

$$\begin{aligned}
R^{g,j}_{r_1,r_2} &= \Big(\bigoplus_{i=1}^{n} \mathsf{F}_{k^i_{a,r_1}}(g,j,r_1,r_2)\Big) \oplus \Big(\bigoplus_{i=1}^{n} \mathsf{F}_{k^i_{b,r_2}}(g,j,r_1,r_2)\Big) \oplus S^{g,j}_{r_1,r_2} \\
S^{g,j}_{r_1,r_2} &= T^j_{c,0} \oplus \chi_{r_1,r_2} \cdot (T^j_{c,1} \oplus T^j_{c,0}) \\
\chi_{r_1,r_2} &= \mathsf{NAND}\big(\lambda_a \oplus r_1, \lambda_b \oplus r_2\big) \oplus \lambda_c \;=\; [(\lambda_a \oplus r_1)\cdot(\lambda_b \oplus r_2) \oplus 1] \oplus \lambda_c \\
\varGamma^j_{w,b} &= h^j_{w,b}(T^j_{w,b}) \oplus k^j_{w,b} \\
\lambda_w &= \begin{cases} \lambda^{j_w}_w & \text{if } w \in \mathsf{Inp} \quad \texttt{// input wire} \\ \lambda^1_w \oplus \cdots \oplus \lambda^n_w & \text{if } w \in [W]/\mathsf{Inp} \quad \texttt{// internal wire} \end{cases} \\
\varLambda_w &= \lambda_w \oplus x_w \text{ for all } w \in \mathsf{Inp} \qquad \texttt{// masked input bit}
\end{aligned}$$

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 $k^j_{w,\varLambda_w}$ 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

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

Let $\varLambda_c$ denote the bit for which $T^j_c = T^j_{c,\varLambda_c}$ and define $k^j_c = \varGamma^j_{c,\varLambda_c} \oplus h^j_{c,\varLambda_c}(T^j_c)$. Finally given $\varLambda_w$ for every output wire w, compute the output carried in wire w as $\varLambda_w \oplus \Big(\bigoplus_{j=1}^n \lambda^j_w\Big)$.

Securely computing $\mathsf{BMR.Encode}$ **using** $\varPi_{\text{DPOLY}}$**.** We decompose the computation of $\mathsf{BMR.Encode}$ 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 mask $\varLambda_w$ values for input wires. More precisely, the parties compute

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

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 $\varPi_{\text{DPOLY}}$. Since the $\varLambda_w$ values for the input wires depend only on the inputs and internal randomness of party P_{j_w}, the $\varLambda_w$ value can be broadcast by that party P_{j_w}. The offline phase comprises of executing all instances of $\varPi_{\text{DPOLY}}$ in parallel in the first three rounds. Additionally, the $\varLambda_w$ 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 $\widehat{\text{C}}$ and then rely on the additive resilience property of $\widehat{\text{C}}$. 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 it is "correct".[12] 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 [LPSY15, HSS17], 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.

[12] 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.

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 $\Pi_{\rm DMULT}$ (Section 3.1), 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 $\Pi_{\rm DMULT}$ 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 $e_{\rm In}\delta$ where δ is some bit in Δ^j_c, where it follows from our simulation that $e_{\rm In}$ will be independent of δ for honest P_j.

We now proceed to a formal description of our protocol.

Protocol 2 (Protocol $\Pi_{\rm DMPC}$ secure against defensible adversaries)

INPUT: *Parties $P_1, \ldots, P_n$ are given input $\hat{x}_1, \ldots, \hat{x}_n$ of length κ', respectively, and a circuit $\widehat{\mathrm{C}}$ 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 $\widehat{\mathrm{C}}$. 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 compute 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 $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$.)

The parties identity the set of 3-monomials that should be computed by the BMR encoding $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$ 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 the t-th monomial (which is set to $\perp$ if P_i does not participate in the computation of the t-th monomial).

– ROUNDS 1,2,3: *For each $i \in [M]$, parties $P_1, \ldots, P_n$ execute Π_{DPOLY} for the monomial 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 monomials. 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 that assembles the BMR encoding. The defense for every monomial is as defined in protocol Π_{DMULT} from Section 3. Namely, for each party P_i there is an NP language*

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

– 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 for every output wire w, $\Lambda_w \oplus \left(\bigoplus_{j=1}^n \lambda^j_w \right)$ and decode the outcome using Dec.

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

Lemma 3.2 (MPC secure against defensible adversaries) *Protocol Π_{DMPC} securely realizes any n-input function f in the presence of a "defensible adversary" that always broadcasts valid defense at the end of the third round.*

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 $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$ 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($\mathsf{defense}$)**:**
  1. The simulator extracts from the defense λ_w^j and $T_{w,0}^j, T_{w,0}^j \oplus T_{w,1}^j$ for every corrupted party P_j and internal wire w. Finally, it obtains the vector of errors $e_{r_1,r_2}^{g,j}$ 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.[13]
  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 of 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_w^j 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 $T_{c,0}^j$ and $T_{c,1}^j$ 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 $S_{\Lambda_a,\Lambda_b}^{g,j}$ to

[13] 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.

be $(e^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\Lambda_c}) \oplus (\bar{e}^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\bar{\Lambda}_c})$ if the majority of the bits in $e^{g,j}_{\Lambda_a,\Lambda_b}$ is 1 and $(\bar{e}^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\Lambda_c}) \oplus (e^{g,j}_{\Lambda_a,\Lambda_b} \cdot T^j_{c,\bar{\Lambda}_c})$ otherwise. Here $\bar{e}^{g,j}_{\Lambda_a,\Lambda_b}$ refers to the complement of the vector $e^{g,j}_{\Lambda_a,\Lambda_b}$ and "$\cdot$" is bitwise multiplication.

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.

– The simulator sends the messages as indicated by the procedure above on behalf of the honest parties. If the adversary provides its fourth message, namely, $\widetilde{R}^{g,j}_{r_1,r_2}$ for $j \in [n], g \in [G], r_1, r_2 \in \{0,1\}$, 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 the procedure outputs the OK message, otherwise it outputs $\perp$.

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

In the full version [HHPV17], we provide a formal proof of the following claim:

Claim 3.3 $\mathbf{REAL}_{\Pi_{\mathrm{DMPC}},\mathcal{A}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n) \overset{c}{\approx} \mathbf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(\kappa, \hat{x}_1, \ldots, \hat{x}_n)$.

4 Four-Round Actively Secure MPC Protocol

In this section we formally describe our protocol.

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 $\widehat{\mathrm{C}}$.*

– 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 $\widehat{\mathrm{C}}$. 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 pairwise independent 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 $\mathsf{BMR.Encode}^{\widehat{\mathrm{C}}}$.)

The parties identify the set of 3-monomials that should be computed by the BMR encoding $\mathsf{BMR.Encode}^{\tilde{\mathrm{C}}}$ 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^i_w, $T^i_{w,r}$, or $\Theta^{i,w,g}_{j,r_1,r_2}$). As described in Section 3.3, 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 the t-th monomial(which is set to $\perp$ if P_i does not participate in the computation of the t-th monomial).

- 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 Section 3.1, for every monomial in case $\mathsf{Role}(t, i) \in \{P_1, P_3\}$. Note that the first message, denoted by $\mathsf{ZAP}^{\mathrm{ENC}}_{i,j}[1]$ 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 other party P_j for the NP language $\mathcal{L}_{\mathsf{Role}(t,i)}$ defined in Section 3.1 for every monomial M_t. As above, the first message, denoted by $\mathsf{ZAP}^{\mathrm{COM}}_{i,j}[1]$ 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 $\mathsf{ZAP}^{\mathrm{ENC}}_{i,j}[2]$ 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. For $b \in \{0, 1\}$ define the NP language:*

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

  - *Chooses $\tilde{w}_{0,i}$ and $\tilde{w}_{1,i}$ such that $\forall t \in [\mathsf{Set}_i]$, $w_{0,i} + \tilde{w}_{0,i} = w_{1,i} + \tilde{w}_{1,i} = \mathsf{wit}_i$ 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 Section 3.1.*
  - *Generates the message $\mathsf{ZAP}^{\mathrm{COM}}_{i,j}[2]$ for the second round of the ZAP protocol relative to the NP language*

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

    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* $\Pi^{i,j}_{\mathrm{DMPC}}[4]$ *to every other party* P_j.
– OUTPUT: *As defined in* Π_{DMPC}.

This concludes the description of our protocol. The proof for the following theorem can be found in [HHPV17].

Theorem 4.1 (Main) *Assuming the existence of affine homomorphic encryption (cf. Definition 2.1) 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.*

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. Following Ananth et. al. [ACJ17], we would like to acknowledge Yuval Ishai's contribution in the three-bit three-round multiplication protocol employed in this work. We would also like to thank Daniel Genkin, Yuval Ishai and Mor Weiss for several discussions on binary AMD resilient circuits.

The first author was supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236. The second author was supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The third author was supported by the National Science Foundation under Grant No. 1617676, 1526377 and 1618884, IBM under Agreement 4915013672 and the Packard Foundation under Grant 2015-63124. The last author was supported by the National Science Foundation under Grant No. 1526377 and 1618884, a Google Faculty Research grant and DIMACS Special Focus on Cryptography program. The work was partially done while the fourth author was at Cornell Tech.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

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

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, pages 483–501, 2012.

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

[BGJ$^+$18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal mpc. 2018.

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

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *TCC*, pages 645–677, 2017.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *EUROCRYPT*, pages 500–532, 2018.

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

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

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

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

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

[COSV17a] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In *TCC 2017*, 2017.

[COSV17b] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In *TCC*, pages 678–710, 2017.

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

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

[DN07] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM J. Comput.*, 36(6):1513–1543, 2007.

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

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

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

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

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

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

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 695–704, 2011.

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

[GS17] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. *IACR Cryptology ePrint Archive*, 2017:1156, 2017.

[HHPV17] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. *IACR Cryptology ePrint Archive*, 2017:1056, 2017.

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

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

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

[Khu17] Dakshita Khurana. Round optimal concurrent non-malleability from polynomial hardness. In *TCC*, pages 139–171, 2017.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam D. Smith. Round efficiency of multi-party computation with a dishonest majority. In *EUROCRYPT*, pages 578–595, 2003.

[LP11] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *STOC*, pages 705–714, 2011.

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

[LPV12] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for UC from only OT. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 699–717, 2012.

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

[OPP14] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2014.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC*, pages 232–241, 2004.

[PW10] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from sub-exponential one-way functions. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 638–655, 2010.

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