Constant-round MPC protocols with Fall-back Security

Anasuya Acharya[1⋆], Aditya Patankar[4⋆⋆], Arpita Patra[2], Divya Ravi[3], and Raghavendra Vernekar[2]

[1] Aalborg University, anasuyaa@es.aau.dk
[2] Indian Institute of Science, {arpita,raghavendrav}@iisc.ac.in
[3] University of Amsterdam, d.ravi@uva.nl
[4] adityashri.patankar@gmail.com

Abstract. The notion of Best-of-Both-Worlds introduced in the work of Ishai et al. (CRYPTO 2006) investigated whether an MPC protocol can simultaneously provide two incomparable security guarantees depending on the number of corrupted parties. As a special case of this, Chaum et al. initiated the study of protocols that tolerate unbounded corruption within a certain adversary structure and PPT corruption of any number of parties beyond that. More recently, Acharya et al. (CRYPTO 2023) formalized this notion as MPC with fall-back security. Although the feasibility of such protocols has now been thoroughly studied in prior works, most of the existing protocols require round complexity linear in the number of parties and the computation size.

In this work, we study the round complexity of MPC with fall-back security in the threshold corruption setting, presenting constant-round protocols for optimal thresholds. We present a semi-honest fall-back secure protocol for $t < \frac{n}{2}$ with 3 rounds, in the plain model, whereas the best known protocol in the same setting takes at least 11 rounds. In the CRS model, we present a maliciously fall-back secure protocol for the same threshold with 4 rounds, satisfying unanimous abort (UA). Finally, we extend this to a 5-round protocol that satisfies *fairness* in the presence of unbounded adversaries for $t < \frac{n}{2}$ corruptions and UA tolerating PPT adversaries for arbitrary corruption beyond that. In the malicious setting, we construct the first constant-round fall-back secure protocols.

Keywords: MPC with Fall-back Security, Best-of-Both-Worlds Security, Exact Round Complexity, Protocol compilers

1 Introduction

Secure multiparty computation (MPC) allows participants with private inputs to collaboratively compute a function in a way that reveals only the function output. Moreover, this security is required to hold even if some participants

⋆ This work was done while the author was affiliated with Aarhus University, Denmark.
⋆⋆ This work was done while the author was affiliated with Indian Institute of Science.

are under the control of a coordinated adversary that can alter their behavior arbitrarily. MPC has been rigorously studied for the past four decades since its inception [GMW86, Yao86] and remains a vibrant field of research, due to its ability to capture most security tasks. It is widely recognized as a cornerstone cryptographic primitive.

The security of MPC is generally characterized by an adversary model which specifies the computational limits (bounded or unbounded), the subset of participants it can manipulate (honest or dishonest majority or more generally an arbitrary family of subsets), and the types of attacks it can initiate against the protocol (e.g., denial of service) or network conditions (synchronous or asynchronous with or without broadcast). The feasibility of MPC for different combinations of these attacks has been thoroughly studied, and we have a clear understanding of the achievable limits in each scenario and their feasibility.

In this paper, our focus is on MPC protocols that achieve security against an unbounded adversary corrupting an honest majority while simultaneously providing security against a computationally bounded (PPT) adversary against a dishonest majority. The work of Chaum et al. [Cha89, CDvdG87] initiated the study of the feasibility of such protocols, followed by the works of Fitzi et al. [FHHW03, FHW04], which study the feasibility of broadcast and consider scenarios that further fine-grain the power of the adversary. More recently, Lucas et al. [LRM10] identify when maliciously secure MPC is feasible with guaranteed output delivery (GOD) and fairness[1] against unbounded adversaries and security with unanimous abort (UA) against PPT adversaries. All of these works have focused on the threshold setting, where a single number is used to capture the maximum number of participants that an adversary can control.

More generally, the set of participants controlled by the adversary can be specified via a family of subsets referred to as an access structure or general adversary model [HM97]. In the general adversary model, Hirt and Maurer [HM97] characterized $\mathcal{Q}^2$ adversary structures[2] as the set of structures where unconditionally secure MPC can be feasibly realized for any functionality. In a recent work, Acharya et al. [AHPV23] study the question of achieving simultaneously security against unbounded adversaries for one access structure $\mathcal{Z} \in \mathcal{Q}^2$ and PPT adversaries corrupting an arbitrary number of parties beyond that. In both the malicious and semi-honest security settings, they show that for any access structure for which security against an unbounded adversary alone is feasible (i.e. for any $\mathcal{Z} \in \mathcal{Q}^2$), any MPC protocol tolerating this structure can be compiled to a protocol that additionally achieves security against maximal corruptions (i.e. up to $n-1$ participants) by a PPT adversary. However, all such protocols in this and previous works incur round complexities linear in computation size and either the number of parties or the size of the adversary structure. They leave open the question of the exact round complexity of protocols that satisfy

[1] A weaker property than guaranteed output where all participants, including the adversary, either all learn the result or none of them do.

[2] $\mathcal{Q}^2$ contains all families of subsets such that no two sets in the adversary structure cover the complete set of n parties.

fall-back security for such optimal corruption structures, even for the threshold case.

1.1 Our Contributions

In this work, we design *constant-round* protocols satisfying MPC with fall-back security in the threshold setting. Henceforth, *a protocol is said to satisfy fall-back security if it achieves information-theoretic (IT) security when there are less than $\frac{n}{2}$ corruptions (this threshold is optimal) and computational security for any number of corruptions beyond that.* We consider both semi-honest and malicious settings. Our protocols can securely realize any functionality whose computation can be captured in NC^1. We leave it open to design such protocols for computation beyond that, as designing constant-round protocols that are secure in the presence of unbounded adversaries for realizing functionalities beyond NC^1 is a big open question. All our protocols assume secure point-to-point and broadcast channels between all parties.

Semi-honest security. In the semi-honest setting, where all corrupt parties faithfully follow the protocol description but may collude to infer extra information from their views, we present a 3-round protocol with fall-back security in the plain model assuming only oblivious transfer (OT).

Theorem (Informal) 1.1.1. *Assuming the existence of a 2-round OT protocol with one-sided IT security against a semi-honest adversary, any n-party functionality in NC^1 can be securely implemented via a 3-round protocol with semi-honest fall-back security.*

Our construction is a compiler that compiles any 2-round semi-honest, IT-secure protocol tolerating $t < n/2$ corruptions into a 3-round semi-honest fall-back secure protocol. Such a 2-round protocol is given in [ABT18]. We note that our compiler itself does not prevent going beyond NC^1 functionalities; this restriction arises only because constant-round IT-secure protocols are currently known only for this class of functionalities, including the 2-round protocol we mention above.

Malicious security. In the malicious setting, an adversary can cause the parties it corrupts to arbitrarily deviate from the protocol. In this setting, different security guarantees can be achieved depending on the permissible corruption. One such guarantee is security with UA, where, after the corrupt parties receive the protocol output, the adversary can cause the protocol to abort before the honest parties receive theirs. All the honest parties are in agreement regarding whether the protocol has aborted, and such a security can be guaranteed in the presence of arbitrary corruption. In this setting, we present the following result.

Theorem (Informal) 1.1.2. *Assuming the existence of a 2-round OT protocol with one-sided IT security against a malicious adversary in the CRS model, any n-party functionality in NC^1 can be securely implemented via a 4-round protocol with malicious fall-back UA security in the CRS model.*

This protocol, similar to the semi-honest case, is a compiler that turns a 2-round malicious IT and UA secure protocol tolerating $t < n/2$ corruptions to a 4-round malicious fall-back UA secure protocol. Such a 2-round MPC protocol is presented in [ACGJ19]. Our compiler uses additional techniques to ensure that any deviation from the base protocol is detected and causes the overall protocol to unanimously abort.

In the malicious setting, a slightly stronger security guarantee than UA is fairness, where either both the honest and corrupt parties receive the protocol output, or the protocol aborts and neither does. Fairness can only be achieved when less than $\frac{n}{2}$ parties are corrupted [Cle86], and we present an extension to our maliciously secure protocol above that satisfies fairness in the honest-majority case (as opposed to UA). This extension incurs one extra round.

Corollary (Informal) 1.1.3. *Assuming the existence of a 2-round OT protocol with one-sided IT security against a malicious adversary in the CRS model, any n-party functionality in* NC^1 *can be securely implemented via a 5-round protocol simultaneously satisfying:*

- *fairness in the presence of an unbounded adversary corrupting* $t < \frac{n}{2}$ *parties*
- *UA in the presence of PPT adversaries corrupting* $t \geq \frac{n}{2}$

Similar to the semi-honest case, our compilers in the malicious setting are restricted to NC^1 functionalities due to the use of a constant-round IT-secure protocol.

1.2 Related Work

The exact round complexity of MPC protocols varies significantly depending on the adversarial model, number of corrupted parties, and the required robustness guarantees. It is known that, for general functionalities, one-round secure computation is impossible in any security setting [HLP11].

In the honest-majority setting with corruption threshold $t < n/2$, the optimal round complexity for MPC is 2 rounds. In the semi-honest setting, [ABT18] showed that any functionality can be computed information-theoretically in 2 rounds in the plain model. Subsequently, [ACGJ19] established a matching 2-round upper bound for malicious security with unanimous abort (UA) in the plain model. Together, these works establish the round-optimal baseline for information-theoretic MPC under standard security notions. Stronger robustness guarantees require higher round complexity. [ACGJ19] also implies 3-round fair MPC protocols tolerating unbounded malicious adversaries. In the honest-majority setting, [AKP20a, AKP23] proved that 4 rounds are both necessary and sufficient for statistically secure MPC with GOD. For perfect security with corruption threshold $t < n/3$ (optimal), [AKP20b] showed that 4 rounds are necessary and sufficient. In the computational setting with PPT malicious adversaries, protocols achieving stronger guarantees, such as fairness or GOD, require 3 rounds in the plain and CRS models, and 2 rounds in the private setup model assuming both CRS and PKI, as shown in [GLS15, ACGJ18]. The lower bounds

of [HLP11, PR18, GIKR02] establish the optimality of these round complexities. [AKP22] improves upon these results by constructing a 3-round protocol achieving GOD using Minicrypt-type assumptions [5]. All the constant-round IT-secure protocols mentioned above realize NC^1 efficiently.

In contrast, achieving MPC with a dishonest majority ($t < n$) necessarily relies on cryptographic assumptions. In the two-party setting, [Yao86] gave a 2-round protocol secure against semi-honest adversaries in the plain model assuming only oblivious transfer (OT). Extending to the multiparty setting, [GS18] constructed a 2-round semi-honest secure MPC protocol in the plain model based on OT, although their construction uses OT in a non-black-box manner. Subsequently, [PS21] obtained a 3-round protocol for semi-honest MPC making black-box use of a 2-round semi-honest secure OT. In the malicious setting, however, stronger robustness notions such as fairness or guaranteed output delivery (GOD) are impossible to achieve in the dishonest-majority regime [Cle86]. Consequently, the security guarantee is typically relaxed to security with abort. Under these relaxed notions, the achievable round complexity depends crucially on the setup assumptions. In the CRS model, [GS18] obtained a 2-round protocol with UA against malicious adversaries. Later, [PS21] constructed a 3-round protocol in the CRS model, making black-box use of a 2-round maliciously secure OT with equivocal receiver security. In the plain model, the round complexity increases further. [KM20] showed that maliciously secure OT requires 4 rounds when output is delivered to a designated party, and 5 rounds when both parties receive output. Building on these lower bounds, [GMPP16, CCG+20] established that 4 rounds are necessary and sufficient for general MPC with UA tolerating arbitrary PPT corruptions in the plain model.

Constant-round protocols for fall-back secure MPC remain largely unexplored in the literature. Most prior works focus primarily on establishing the feasibility of fall-back security, without explicitly optimizing round complexity. The work of [Cha89] introduced the first fall-back secure protocols in the semi-honest and malicious settings, while [LRM10] later extended the study in the malicious setting by considering various robustness guarantees, such as UA, fairness, and GOD. Their constructions incur non-constant round complexity. More recently, Acharya et al. [AHPV23] constructed fall-back secure protocols in both the semi-honest and malicious settings. Their techniques yield an 11-round semi-honest fall-back secure protocol for NC^1 functionalities.

1.3 Technical Overview

Existing approaches for constructing fall-back secure protocols follow two broad paradigms. The first, originating in the work of [Cha89] and also adopted in [LRM10], uses an IT-secure protocol to realize a party's actions within a computationally secure protocol. Consequently, when using their approach to design a constant-round protocol, the relevant computation in the computationally

[5] Minicrypt-type assumptions refer to cryptographic assumptions that imply one-way functions but are not known to imply public-key cryptography.

secure protocol must be realizable in NC^1. Since the computations arising in our setting are not known to lie in NC^1, this approach does not help in designing efficient constant-round protocols. A different approach, explored in [AHPV23], reverses the order of composition. Instead of realizing a computational protocol through instances of an information-theoretically secure protocol, it begins with an information-theoretically secure protocol and realizes it using designated-party computational protocols. This avoids the NC^1 bottleneck inherent in the approach used in [Cha89, LRM10]. Our construction follows this latter paradigm and yields constant-round fall-back secure protocols. In this section, we describe the main ideas and technical challenges underlying our constructions.

Semi-honest fall-back secure protocol. Our first contribution is a 3-round n-party protocol in the plain model that is semi-honest secure in the presence of an unbounded adversary corrupting up to $t < \frac{n}{2}$ parties, or a PPT adversary corrupting $t < n$ parties. We construct this in three steps:

- **Step 1: Virtual Protocol.** Our starting point is a 2-round MPC protocol [ABT18] that securely realizes any functionality against *semi-honest* unbounded adversaries corrupting fewer than $\frac{n}{2}$ parties. We will denote this by "virtual protocol", since the role of the parties in this protocol will be executed via a special kind MPC protocol described next.
- **Step 2: Designated-Party Protocol.** We construct a 2-round *designated-party* protocol for computing any functionality. That is, for a designated $i \in [n]$, this protocol tolerates semi-honest unbounded adversaries for any set of corrupt parties, as long as party P_i is not corrupted. The protocol additionally tolerates arbitrary corruption by a semi-honest PPT adversary – it is PPT secure even when P_i is corrupted. Assuming a 2-round OT, we construct such protocols in the plain model by adapting techniques presented in [GS18], which presents only computationally-secure protocols.
- **Step 3: Compiling to Semi-honest Fall-back.** We use multiple instances of the designated-party protocol to compile the virtual protocol (given in Step 1) and obtain fall-back security. We do this using garbled circuits and party virtualization techniques, inspired by ideas in [AHPV23].

Our main technical contributions lie in the design of the designated-party protocol and the compilation techniques discussed in Steps 2 and 3. Figure 1 illustrates our approach. We outline the main ideas of these constructions next and refer the reader to Section 3 for their details.

Step 2: the designated-party protocol. We present a compiler that can take any MPC protocol that is semi-honest secure, tolerating arbitrary PPT corruption and convert it into a 2-round designated-party protocol for the same functionality, where the designated party can be any of the n parties and is known to all. Let P_1 be the designated party for simplicity. Then this compiled protocol is PPT secure in the presence of arbitrary corruption, and also IT-secure as long as P_1 is not corrupted.

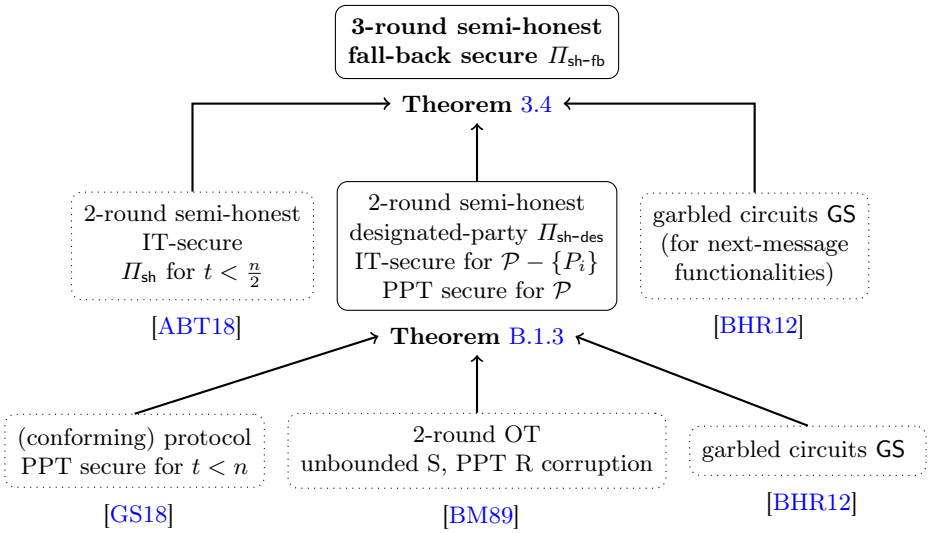

Fig. 1. Compiling a 2-round semi-honest information-theoretically secure protocol into a 3-round semi-honest protocol with fall-back security. Dotted boxes and lines indicate primitives and implications from prior work.

For this, we closely follow the compilation technique given in [GS18]: each party garbles all of its next-message functions in the multi-round protocol and broadcasts this to all other parties; then all parties can step through the protocol by locally evaluating all these garbled circuits[3]. This compiler uses a 2-round OT protocol to distribute input labels for the garbled circuits corresponding to the first-round computation, which in turn will produce the input labels for the computation of the next round, and so on. The security of the overall protocol relies on the security of garbling and OT, and therefore, the protocol is only secure in the presence of PPT adversaries.

In contrast, we additionally require information-theoretic security in the presence of any collusion as long as the designated party P_1 is honest. However, we consider protocols in which the designated party is the only one receiving the output[4]. Looking ahead, this suffices for use in our final fall-back secure protocol. Therefore, in our protocol, only the designated party receives the garbled circuits and performs the evaluation. Further, we use 2-round OT protocols that provide IT security to the receiver and PPT security to the sender, with P_1 acting only as the receiver for all OT invocations. Our compiled protocol roughly works as follows:

[3] The compiler in [GS18] starts with "conforming protocols" that admit a certain sequential structure. [GS18] also shows that any protocol can be converted into a conforming protocol while preserving its security properties.

[4] In the semi-honest setting, this can be easily altered into a protocol where all parties receive the output, using an extra round where P_1 broadcasts this output.

– **Round 1:**
  - Every party sends OT receiver messages to all other parties for all OT invocations, using (functions of) its input as the choice bits.
  - The parties communicate among themselves to share the information required to construct compatible garbled circuits.
– **Round 2:**
  - P_1 receives messages in all OT protocols with all other parties as senders.
  - All parties garble each computation corresponding to their actions in the protocol, with their inputs hard-coded in; these garbled circuits are sent to P_1.
– **Output Computation:**
  - P_1 obtains the input labels for the first garbled circuit computation.
  - P_1 then evaluates the garbled circuits sequentially and decodes the OT output to obtain the output labels. These labels serve as the input labels for the next. Evaluating the final garbled circuit yields the output $f(x)$.

In the above protocol, note that if a PPT adversary corrupts any number of parties, just like in [GS18], it follows from the privacy of garbling and the OT protocol that the protocol remains secure. Finally, even if any number of parties (except the designated party) are corrupted by an unbounded adversary, none of these parties ever receive garbled circuits of other parties. In particular, the only messages they receive are the OT-receiver messages from the honest parties. It therefore follows from the IT security provided to the receiver in the OT scheme that the overall protocol execution remains secure.

Step 3: compiling to general fall-back security. To construct a 3-round protocol with fall-back security, our starting point is the virtual protocol i.e. the 2-round semi-honest IT-secure honest-majority protocol [ABT18]. Let us refer to the parties in this protocol by $\{V_i\}_{i\in[n]}$. We first observe that any such protocol can be parsed as a set of next-message functions. There are 3 such functions for each party, one corresponding to each of the 2 rounds, along with an additional function for output computation. The high-level idea for the compiler is to instantiate functionalities computing these next-message functions using the designated-party protocol. Specifically, all next-message functions associated with a virtual party V_i are executed using an instance in which P_i acts as the designated party. Thus, P_i emulates V_i, and the parties $\{P_i\}_{i\in[n]}$ emulate an execution of the virtual protocol.

This idea was used in [AHPV23] wherein they construct a fall-back secure protocol by first secret-sharing all the parties' inputs, and then, round-by-round, execute n designated-party protocols to emulate each round of the virtual protocol in sequence. They use a designated-party protocol where *all* parties receive the output and maintain the invariant that the complete state of all the parties of the virtual protocol sits secret shared among all of the n parties. For a virtual protocol taking r_v rounds and designated-party protocol taking r_d rounds, their protocol would take $(r_v + 1) \times r_d + 2$ rounds. Thus, a direct application using a 2-round virtual protocol together with our designated-party protocol would require

$(r_v + 1) \times (r_d + 1) + 2$ rounds, where the additional round in the designated-party protocol is needed to deliver the output to all parties. Instantiating with $r_v = 2$ and $r_d = 2$ therefore yields an overall $((2 + 1) \times (2 + 1) + 2) = 11$-round protocol. We reduce the round complexity from 11 rounds to 3 rounds.

However, in order to design a 3-round protocol, further techniques are required. Towards this, we observe that within our 2-round virtual protocol, each party V_i's first-round messages can be computed locally by P_i, without inputs from other parties and therefore need not be executed as an n-party functionality. All the messages of the second-round of the virtual protocol are then computed using designated-party protocols designed in "Step 2" – this would take 2 rounds. Finally, if all the functions for output decoding were to be executed as designated-party protocols, this would take 2 extra rounds in sequence, after the first two. To prevent this, we use n instances of designated-party protocols to compute garbled circuits that decode to the virtual protocol output – party P_i is the designated party that receives the garbled circuit for decoding V_i's output in the virtual protocol. These garbled circuits are generated in parallel with the designated-party protocols computing the second-round messages. These garbled circuits receive their input labels from the output of the second-round computation of the virtual protocol. The third round is needed to send these labels. Our compiled protocol therefore, works as follows:

- **Rounds 1 and 2:**
  1. Each party P_i locally executes the first next-message function of V_i.
  2. Execute n designated-party protocols that each compute a garbled circuit: a protocol with P_i as the designated party is used to compute the garbled circuit that outputs V_i's output in the virtual protocol.
  3. Execute n designated-party protocols that use all the first-round messages as input, compute the second-round messages, and use these to choose input labels to the output-decoding garbled circuits.
- **Round 3:** Each party P_i gives to each party P_j, the labels that it obtained from the designated-party protocol instance (step 3 above), which P_j needs to evaluate its garbled circuit.
- **Output Computation:** Each party P_i uses the output decoding garbled circuit (step 2 above), and all input labels it received to evaluate its garbling and receive V_i's output.

In the above protocol, note that if a PPT adversary corrupted any number of parties, it follows from the garbling privacy and the cryptographic security of all the designated-party protocols that the overall protocol remains secure. Finally, if less than $\frac{n}{2}$ parties were corrupted by an unbounded adversary, all the designated-party protocols (where the corrupt party is designated) that these parties participate in and the garbled circuits that they see become compromised. In this case, for any corrupt party P_i, note that all this reveals is the view of V_i in the virtual protocol. However, as long as there are less than $\frac{n}{2}$ corrupt parties, this view is simulatable in the virtual protocol, and therefore the compiled protocol remains secure.

A 2-round fall-back secure protocol? With a 3-round blue-print of a fall-back secure protocol as above, a natural question is if this 3 is the optimal bound. Towards this, we provide an evidence that proving the impossibility of the existence of a 2-round protocol may be a challenging problem. In more detail, we can show that there exists a 2-round MPC protocol for realizing any functionality with fall-back security, assuming the existence of statistically robust Additive Randomized Encodings (statistical RARE – Definition A.16) for any function. Note that the encoding function is locally computed by each party, so the corrupt parties can compute encodings on multiple inputs, decode the outputs with respect to these encoded inputs, and learn some extra information. To prevent this, each party computes and sends commitments on its protocol input in the first round. The function for the statistical RARE additionally takes these commitments (or their shares) and the respective randomness as inputs, along with the actual protocol inputs (which are committed upon), and internally checks the commitment before computing the functionality. We use statistically hiding as well as statistically binding commitments in order to ensure security against both unbounded and computational adversaries. In order to hide the honest parties' inputs in the unbounded case, the statistically binding commitments are t-out-of-n Shamir shared. Our 2-round protocol works as follows:

- **Round 1:** Each party computes a statistically binding commitment of its input, and for $t < \frac{n}{2}$, creates a t-out-of-n secret sharing of this commitment. It also generates a statistically hiding commitment of its input and n-out-of-n additive secret shares of the 0-value. The statistically hiding commitment is broadcast, and the shares of statistical binding commitment and the shares of 0 are distributed.
- **Round 2:** Each party encodes the messages received in Round 1, together with its own input and the randomness used in computing its commitments, to generate its encoded input. This local encoding is masked with the sum of all the 0-shares received from the previous round, and this value is broadcast.
- **Output Computation:** Each party, on receiving all the masked encodings of the RARE, sums them to get an encoding of the output, and decodes this to get the actual output.

We now describe the function corresponding to the RARE in greater detail. This function does the following:

1. reconstructs each statistically binding commitment from the t-out-of-n shares and verifies that the reconstructed commitment matches the statistical binding commitment provided by that party.
2. checks that the commitment computed using the input and the randomness provided by the party matches the statistically hiding commitment provided by that party.
3. computes the desired functionality f on the inputs to obtain output y if the above checks pass, else it aborts by sending $\perp$.

Details of this protocol and its proof can be found in Appendix B.2. Although instantiations for a statistical RARE are not yet known, this construction demonstrates the difficulty of proving a round complexity lower bound, implying the

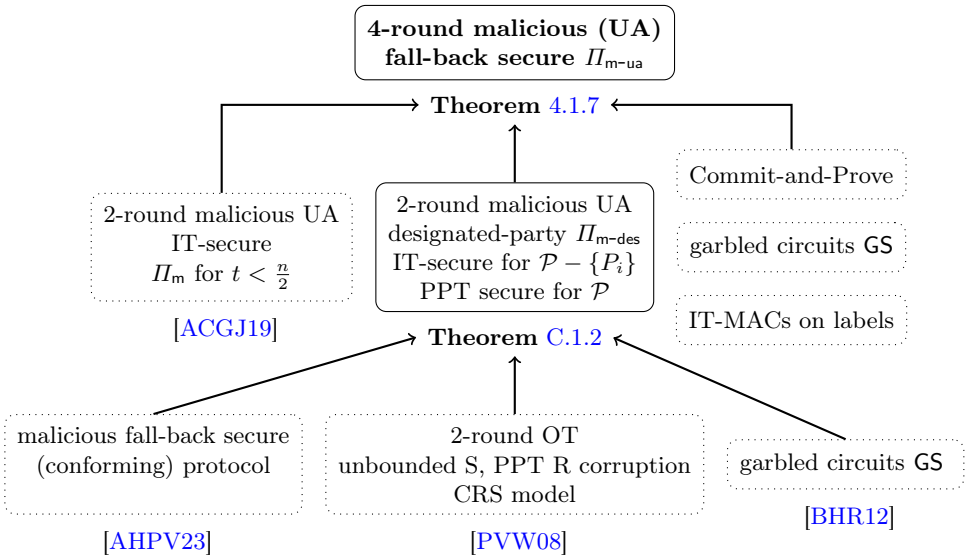

Fig. 2. Compiling a 2-round malicious information-theoretically secure protocol into a 4-round protocol with malicious fall-back security, achieving security with UA. Dotted boxes and lines indicate primitives and implications from prior work.

impossibility of a 2-round fall-back secure protocol for general functionalities. We leave it as an interesting open question to design such a 2-round protocol using building blocks with known instantiations (that are possibly weaker than a statistical RARE) or show that any such 2-round protocol implies a statistical RARE.

Maliciously fall-back secure protocol. As our second result, we present a 4-round n-party protocol in the CRS model that achieves malicious security and satisfies UA against either an unbounded adversary corrupting $< \frac{n}{2}$ parties or a PPT adversary corrupting $< n$ parties. The protocol builds on the same high-level ideas underlying our semi-honest construction We construct this in three stages:

- **Step 1: Virtual Protocol.** Our starting point is a 2-round MPC protocol [ACGJ19] that can securely compute any functionality in the presence of *malicious* unbounded adversaries satisfying security with UA for $< \frac{n}{2}$ corruption.
- **Step 2: Designated-Party Protocol.** The designated-party protocol used in the compilation must also provide malicious security. We obtain such a 2-round protocol by adapting [GS18] compilation for the *malicious* adversary case. As in the semi-honest case, only the designated-party receives the OT outputs and evaluates the garbled circuits. In order to enable this, we require a 2-round OT protocol that provides IT security to the receiver and PPT

security to the sender, in the presence of malicious adversaries. Such OT protocols are only known to exist in the CRS or random oracle (RO) model. Therefore, our overall protocol is designed in the *CRS model*.
- **Step 3: Compiling to Malicious Fall-back.** We use multiple instances of the malicious designated-party protocol to compile the 2-round MPC protocol from Step 1 and obtain malicious fall-back security. Similar to the semi-honest setting, the compilation uses garbled circuits and party virtualization techniques inspired by [AHPV23]. However, extending the compilation to handle malicious adversaries introduces additional challenges, which we discuss next.

Randomness Generation: Recall that in the semi-honest secure compiled protocol, each party P_i locally computes the first-round messages of virtual party V_i using its inputs and internally (honestly) sampled randomness. However, for the compiled protocol to be maliciously secure, even in the case that a PPT adversary corrupts a majority of parties, more than $\frac{n}{2}$ parties cannot be allowed to sample arbitrary randomness for use in the virtual protocol. Therefore, the first-round virtual protocol messages in the compiled protocol are generated using n instances of parallely executed designated-party protocols: the designated party P_i supplies the input of V_i and all parties supply an additive share of the randomness for V_i in the virtual protocol.

The 4-round protocol outline. In the malicious security setting, garbled circuits are used to evaluate both the second-round and output-decoding functions of the virtual protocol, while the first-round function is computed directly through the designated-party protocol instance. Beyond randomness generation, compiling to malicious fall-back requires mechanisms for broadcast consistency, label authenticity, and consistency across protocol instances. Together, this results in a 4-round protocol, with the following rough outline:

- *Rounds 1 and 2:* Execute n designated-party protocols, where the protocol with P_i as the designated party does the following:
  1. Compute the garbled circuit that performs output-decoding to obtain V_i's output in the virtual protocol.
  2. Compute the garbled circuit that computes the second-round messages of V_i in the virtual protocol, and then uses these messages to select the input labels for the output-decoding garbled circuits of all parties V_j.
  3. Compute the first-round messages of V_i, and use these to choose input labels to the circuits computing the second-round messages of all V_j.
- *Round 3:* Each party P_i sends to each party P_j the labels obtained from the designated-party protocol (step 3 above). Using these labels as inputs, P_j evaluates the garbled circuit corresponding to the virtual party V_j. This garbled circuit computes the second-round messages of V_j in the virtual protocol and outputs the labels corresponding to these second-round messages.
- *Round 4:* Each party P_i uses the garbled circuit (step 2 above) and the input labels it received to evaluate its garbled circuit and obtains the input labels

to all the output-decoding garbled circuits. P_i gives to each party P_j the labels that it obtained. P_j uses these labels to evaluate its output decoding garbled circuit to obtain the output.
– *Output Computation:* Each party P_i uses the output decoding garbled circuit (step 1 above) and all input labels it received in round 4 to evaluate its garbled circuit and receive V_i's output.

The security of the above protocol would roughly follow from the malicious security of the individual protocols and the security of garbling in a similar way as that of the semi-honest compiled protocol. However, there is one last hole to plug, which we talk about in the ensuing text.

Simulating Broadcast: The malicious security of the protocol relies on secure broadcast, which prevents an adversary from sending inconsistent messages to different honest parties. In the compiled protocol, however, broadcast messages are represented by garbled-circuit labels. Thus, all honest parties must use label pairs corresponding to the same semantic broadcast value. We ensure this by enforcing that the same label pair for every broadcast message is used across all honest parties' garbled circuits.

Label Authenticity: Another challenge is ensuring the authenticity of the labels exchanged in rounds 3 and 4, since a corrupt party may send labels different from those obtained from its garbled circuit. To prevent this, we authenticate all labels using IT-MACs and one-time multi-key IT-MACs. This guarantees that even an unbounded adversary cannot forge valid labels. Additional techniques are then used to preserve label confidentiality. We refer the reader to Section 4.1 for the protocol details.

Consistency: The final challenge is ensuring that corrupt parties use a consistent virtual execution state across all designated-party protocol instances. To enforce this, each party first commits to its input and randomness, and then proves in zero knowledge that it used consistent values across protocol instances. Since parties receive garbled circuits in round 2 and the corresponding labels in round 3, the consistency proof must be completed before circuit evaluation; otherwise, honest parties may evaluate inconsistent or maliciously generated circuits. This also necessitates a 3-round zero-knowledge proof protocol. The final output of the compiled protocol is only accepted if all these proofs verify.

Figure 2 illustrates this compilation of a 2-round malicious information-theoretically secure protocol into a 4-round maliciously fall-back secure protocol satisfying UA, highlighting the building blocks used. We refer the reader to Section 4 for further details.

Achieving fairness for the honest majority setting. Our maliciously secure protocol also benefits from the fact that it can be altered to one that satisfies *fairness* in the presence of unbounded adversaries while still satisfying UA in the PPT corruption case. This is done by having the virtual protocol output an $\frac{n}{2}$-threshold robust

secret-sharing of the required function output $f(x)$. This is a standard technique for achieving fairness from UA used avidly in prior work [IKP+16, AHV25].

Our compiled protocol would therefore take one extra round (five rounds total), with the last round involving reconstructing $f(x)$ from these shares. Additionally, we employ one-time multi-key IT-MACs to authenticate each share of $f(x)$ in the functionality realized. This enables each honest party to differentiate the correct shares from corrupt ones, for use during reconstruction. We refer the reader to Section 4.2 for details of the protocol. Such a transform satisfies fairness when $< \frac{n}{2}$ parties are corrupted, if the protocol had (unanimously) aborted before the last round, the adversary cannot learn enough shares to reconstruct $f(x)$. However, if the last round is completed, both the adversary and honest parties are guaranteed to recover $f(x)$.

On the difficulty of achieving guaranteed output delivery. In the model considering malicious adversaries, the strongest security notion that an MPC protocol can hope to achieve is that of guaranteed output delivery (GOD). This is a notion that can be only achieved in the presence of an honest majority, and in the Best-of-both-Worlds setting, there exist feasibility lower bounds [Kat07, IKK+11] stating that if a protocol guarantees output delivery for up to $t < \frac{n}{2}$ corruption, it can only guarantee security (even computational UA) for at most $s < n - t$ parties. No security can be guaranteed for corruption beyond that. Although computationally secure constant-round Best-of-Both-Worlds protocols satisfying GOD and UA exist for such corruption thresholds, designing constant-round protocols for the case where additionally unconditional security is guaranteed in the honest-majority setting is an open question and beyond the scope of this work. This is largely owing to the fact that existing techniques that constant-round protocols use to achieve GOD rely on computational assumptions to identify corrupt parties and exclude them from the computation. It is open as to how these can be adapted to a setting that additionally requires unconditional security against a corrupt minority, while letting the protocol remain constant-round.

2 Preliminaries

We denote by κ a computational security parameter and s a statistical security parameter that captures a statistical error of up to 2^{-s}. We assume $s \leq \kappa$. The set $\{1, 2, \ldots, n\}$ is denoted as $[n]$. We defer the additional preliminaries, including standard MPC definitions, to Appendix A.

3 Semi-Honest Fall-back Secure MPC

In this section, we construct a 3-round protocol that achieves fall-back security against semi-honest adversaries. Our construction compiles: (i) a 2-round "virtual protocol" Π_{sh} that is information-theoretically secure against an unbounded adversary corrupting less than $\frac{n}{2}$ parties, and (ii) a 2-round "designated-party protocol" $\Pi_{\mathsf{sh\text{-}des}}$ that provides:

– information-theoretic security against an unbounded adversary corrupting any subset of parties excluding the designated party, and
– computational security against a PPT adversary corrupting an arbitrary subset of parties, possibly including the designated party.

Assuming the designated party is P_i, we denote the corresponding adversary structure by $\mathcal{Z}_{\mathsf{D}} = \{Z \subseteq \{P_j\}_{j\in[n]\setminus\{i\}}\}$. We obtain $\Pi_{\mathsf{sh\text{-}des}}$ by modifying the construction of [GS18]. The details appear in Appendix B.

Following [ABT18], any multiparty functionality can be securely evaluated in 2 rounds assuming an honest majority. We instantiate Π_{sh} using this protocol and represent it via next-message functions as

$$\Pi_{\mathsf{sh}} = \left\{\mathsf{NM}_i^j\right\}_{i\in[n],\, j\in[3]}$$

Here, NM_i^3 denotes the local output computation performed by the i-th party, while NM_i^1 can be computed locally using input x_i and randomness r_i. We denote the parties of the virtual protocol as $\{V_i\}_{i\in[n]}$. At a high level, each party P_i emulates a virtual party V_i executing Π_{sh}, while maintaining V_i's view in an encoded form. The view of the party is encoded using labels that are additively secret shared among the parties. The party P_i locally computes NM_i^1, and then acts as the designated party in instances of $\Pi_{\mathsf{sh\text{-}des}}$ that compute NM_i^2 and generate garbled circuit for NM_i^3. The outputs of these computations are represented using additively shared labels, preventing the adversary from learning the joint view of more than $\frac{n}{2}$ virtual parties in the computational setting. By the end of round 2, each P_i obtains the labels corresponding to the output of NM_i^2 together with a garbled circuit for NM_i^3. In round 3, the parties exchange the required labels corresponding to the virtual round-2 messages. Each P_i then evaluates the garbled circuit for NM_i^3 using these labels and locally recovers its output.

To provide concrete intuition for the protocol's execution, we trace the message flow for a single party P_1 in a toy example with 3 parties P_1, P_2, P_3. We focus only on P_1's next-message functions since the other parties' next-message functions are similar. We denote the message sent by P_i to P_j in the r-th round of Π_{sh} as T_{ij}^r. Thus, NM_1^2 can be described as

$$\mathsf{NM}_1^2(x_1, r_1, T_{11}^1, T_{21}^1, T_{31}^1) \to (T_{11}^2, T_{12}^2, T_{13}^2)$$

We simplify further and consider each T_{1i}^2 as a single bit. Each party samples random shares of labels, corresponding to bits 0 and 1, for each of these bit messages. We now modify the next-message function to output the labels corresponding to T_{1i}^2 rather than the value itself. We only show how the label for T_{12}^2 is computed. Denote this modified next-message function as $\hat{\mathsf{NM}}_1^2$, which is described as follows:

$$\hat{\mathsf{NM}}_1^2((x_1, r_1, T_{11}^1, \mathsf{L}_1^0, \mathsf{L}_1^1), (T_{21}^1, \mathsf{L}_2^0, \mathsf{L}_2^1), (T_{31}^1, \mathsf{L}_3^0, \mathsf{L}_3^1)) \\ \to \mathsf{L}^{T_{12}^2}(= \mathsf{L}_1^{T_{12}^2} \oplus \mathsf{L}_2^{T_{12}^2} \oplus \mathsf{L}_3^{T_{12}^2})$$

where L_i^b denotes P_i's additive share of the label corresponding to bit b. Similarly, P_1 obtains $\mathsf{L}^{T^2_{11}}$ and $\mathsf{L}^{T^2_{13}}$ as well. This computation is done in 2 rounds using the $\Pi_{\mathsf{sh\text{-}des}}$ instance. In the next round, P_1 forwards the labels $\mathsf{L}^{T^2_{12}}$ and $\mathsf{L}^{T^2_{13}}$ to P_2 and P_3 respectively.

Next, we describe how P_2 obtains its garbled circuit for NM_2^3. For simplicity, we focus only on the input T_{12}^2 to NM_2^3. The idea is to generate a garbled circuit for NM_2^3, whose input label corresponding to bit b for T_{12}^2 is $\mathsf{L}_1^b + \mathsf{L}_2^b + \mathsf{L}_3^b$, where L_i^b is the additive share input by P_i to the $\hat{\mathsf{NM}}_1^2$ function as shown above. The garbling protocol does the following:

$$((\mathsf{L}_1^0, \mathsf{L}_1^1), (\mathsf{L}_2^0, \mathsf{L}_2^1), (\mathsf{L}_3^0, \mathsf{L}_3^1)) \rightarrow \widetilde{\mathsf{NM}}_2^3$$

where $(\mathsf{L}^0 = \mathsf{L}_1^0 + \mathsf{L}_2^0 + \mathsf{L}_3^0, \mathsf{L}^1 = \mathsf{L}_1^1 + \mathsf{L}_2^1 + \mathsf{L}_3^1)$ are the labels for the input bit T_{12}^2 in the garbled circuit $\widetilde{\mathsf{NM}}_2^3$. Thus, when P_1 forwards the label $\mathsf{L}^{T^2_{12}}$ to P_2, P_2 can evaluate this garbled circuit to obtain its output. The message T_{12}^2 remains encoded through this label, ensuring security in the computational case. The circuit $\widetilde{\mathsf{NM}}_2^3$ is obtained by P_2 at the end of round 2 along with the labels $\mathsf{L}^{T^2_{21}}, \mathsf{L}^{T^2_{22}}, \mathsf{L}^{T^2_{23}}$, through another designated-party protocol instance. Thus, the overall protocol consists of each party running 2 designated-party protocol instances to get the labels and garbled circuit, respectively, and an additional round where parties exchange the labels obtained from this instance.

In the above example, although we speak of separate designated-party protocol instances for computing NM_i^2 and NM_i^3, these can be unified to a single protocol instance. Note that the two instances share some common input (additive shares of the label), and ensuring that the same input is used in the two instances is necessary for the correctness of the protocol. Although this is trivially ensured in the semi-honest case, looking ahead, we will need additional checks to ensure this in the malicious security case. Unifying these instances eliminates the problem of ensuring consistency of common inputs and is thus key in simplifying the maliciously secure protocol.

Notation. We assume that every message exchanged between any two parties in the protocol Π_{sh} is ℓ bits long. Thus, the second and third next-message functions accept $n\ell$ bits as input. For each message T_{ij}^2 (where $i, j \in [n]$), the parties collectively hold ℓ pairs of label shares. Specifically, for every $h \in [\ell]$, the label corresponding to the h-th bit of T_{ij}^2 is denoted by $\mathsf{L}_{i,j,h}^b$ for $b \in \{0,1\}$. Each party P_k holds a share of this label, denoted by $\mathsf{L}_{i,j,k,h}^b$ for $b \in \{0,1\}$.

The complete label corresponding to the h-th bit of T_{ij}^2 used in the garbled circuit $\widetilde{\mathsf{NM}}_j^3$ is reconstructed as:

$$\mathsf{L}_{i,j,h}^b = \bigoplus_{k=1}^{n} \mathsf{L}_{i,j,k,h}^b$$

To simplify the notation, we denote the complete set of labels corresponding to all bits of the message T_{ij}^2 as $\mathsf{L}_{i,j}^b = \{\mathsf{L}_{i,j,h}^b\}_{h \in [\ell]}$.

Protocol 3.1: 3-Round Fall-Back Secure Protocol $\Pi_{\mathsf{sh\text{-}fb}}$ for $t < \frac{n}{2}$

Setting: Let $P = \{P_i\}_{i\in[n]}$ be set of parties. $\Pi_{\mathsf{sh}} = \{\mathsf{NM}_i^1, \mathsf{NM}_i^2, \mathsf{NM}_i^3\}_{i\in[n]}$. Functionality G_i (Functionality 3.2) and Functionality H_i (Functionality 3.3) correspond to NM_i^2 and NM_i^3, respectively.
Building Blocks:

- Garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ (Definition A.15).
- Semi-honest designated-party protocol (Protocol B.1.1).

Inputs: Each party P_i has its input (x_i, r_i).
The Protocol:

- **Round 1:** Each party P_i does the following:
  1. Compute $\{T_{ij}^1\}_{j\in[n]} = \mathsf{NM}_i^1(x_i, r_i)$.
  2. Begin instances of $\Pi_{\mathsf{sh\text{-}des}}^{G_i}$ computing G_i (Functionality 3.2) and $\Pi_{\mathsf{sh\text{-}des}}^{H_i}$ computing H_i (Functionality 3.3) with P_i as the designated party.
  3. Participate in other instances of $\Pi_{\mathsf{sh\text{-}des}}^{G_j}$ and $\Pi_{\mathsf{sh\text{-}des}}^{H_j}$ for each $P_j\{j \neq i\}$ as the designated party.
- **Round 2:** Each party P_i
  1. Computes output of $\Pi_{\mathsf{sh\text{-}des}}^{G_i}$ as $(\mathsf{L}_{i,j})_{j\in[n]}$.
  2. Computes the output of $\Pi_{\mathsf{sh\text{-}des}}^{H_i}$ as $(\widetilde{\mathsf{NM}}_i^3, d_i)$.
- **Round 3:** Each party P_i does the following:
  1. Send $\mathsf{L}_{i,j}$ to P_j for all $j \in [n]$.
  2. Receive $\mathsf{L}_{j,i}$ from P_j for all $j \in [n]$.
  3. Obtain output $y_i \leftarrow \mathsf{De}(\mathsf{Ev}(\widetilde{\mathsf{NM}}_i^3, \{\mathsf{L}_{j,i}\}_{j\in[n]}), d_i)$.

Functionality 3.2: Function G_i based on NM_i^2

Inputs:
Each party P_k inputs T_{ki}^1 and $n\ell$ pairs of randomly sampled labels, denoted as $\{(\mathsf{L}_{i,j,k,h}^0, \mathsf{L}_{i,j,k,h}^1)\}_{j\in[n],h\in[\ell]}$.
P_i additionally inputs (x_i, r_i) in the protocol.
Computation:

1. Compute $\mathsf{L}_{i,j,h}^b = \bigoplus_{k=1}^n \mathsf{L}_{i,j,k,h}^b$ for $b \in \{0,1\}$ and $j \in [n]$.
2. Compute $\{T_{ij}^2\}_{j\in[n]} = \mathsf{NM}_i^2(x_i, r_i, \{T_{ji}^1\}_{j\in[n]})$.
3. For each $j \in [n]$, parse $T_{ij}^2 = (z_{j,1}, ..., z_{j,\ell})$ and compute $\mathsf{L}_{i,j} = \{\mathsf{L}_{i,j,h}^{z_{j,h}}\}_{h\in[\ell]}$.

Output:
G_i gives $(\mathsf{L}_{i,j})_{j\in[n]}$ to P_i.

Functionality 3.3: Function H_i based on NM_i^3

Inputs:
Each party P_k inputs $n\ell$ pairs of randomly sampled labels, denoted as $\{(\mathsf{L}_{j,i,k,h}^0, \mathsf{L}_{j,i,k,h}^1)\}_{j\in[n],h\in[\ell]}$.

The protocol also takes the description of the next-message function NM_i^3 as input and uses the garbling scheme discussed in [LP09][6].

Computation:

1. Compute $\mathsf{L}_{j,i,h}^b = \bigoplus_{k=1}^n \mathsf{L}_{j,i,k,h}^b$ for $b \in \{0,1\}$ and $j \in [n]$.
2. Generate a garbled circuit $(\widetilde{\mathsf{NM}}_i^3, e_i, d_i)$ with input labels corresponding to h-th bit of T_{ji}^2 as $\mathsf{L}_{j,i,h}^b$ for $j \in [n], h \in [\ell], b \in \{0,1\}$.

Output:

H_i gives $(\widetilde{\mathsf{NM}}_i^3, d_i)$ to P_i.

Theorem 3.4 (3-round Semi-honest Fall-back Compiler)**.** *Let Π_{sh} be a 2-round n-party protocol that securely computes a functionality f in the presence of unbounded semi-honest adversaries with corruption threshold $t < \frac{n}{2}$. Let $\Pi_{\mathsf{sh\text{-}des}}$ be a 2-round n-party protocol that achieves fall-back security with respect to $\mathcal{Z}_{\mathsf{D}}$ adversary structure. Assuming $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is a garbling scheme, there exists a 3-round protocol $\Pi_{\mathsf{sh\text{-}fb}}$ that securely computes f with semi-honest fall-back security, tolerating $t < \frac{n}{2}$ corruption in the plain model.*

We provide a proof sketch here. The formal proof of Theorem 3.4 appears in Appendix D.2.

Proof Outline. We informally describe how the protocol $\Pi_{\mathsf{sh\text{-}fb}}$ achieves fall-back security. We first consider the case when $t < \frac{n}{2}$ parties are corrupted by an unbounded adversary. In this case, we argue that the security of our protocol reduces to the security of the underlying information-theoretically secure protocol Π_{sh}. In the protocol $\Pi_{\mathsf{sh\text{-}fb}}$, each party computes its next-message functions in the protocol Π_{sh} through a designated-party protocol instance. In the case where the designated party is honest, from the security guarantees of $\Pi_{\mathsf{sh\text{-}des}}$, the adversary doesn't learn anything about the honest parties' inputs, and hence, their view in the protocol Π_{sh} remains hidden. If the designated party is corrupt, $\Pi_{\mathsf{sh\text{-}des}}$ provides no security guarantees, and hence, the adversary can learn the honest parties' inputs. However, the honest parties' inputs in such a corrupt designated instance are the messages (or the labels corresponding to these messages) that they send to the corrupt parties in Π_{sh}. Hence, the privacy of the honest parties' inputs in such instances reduces to that in Π_{sh}.

In the computational case, the designated-party protocol $\Pi_{\mathsf{sh\text{-}des}}$ that is used to execute the next-message function of Π_{sh} provides computational security against a PPT adversary corrupting any arbitrary subset of parties. Thus, the designated-party protocol instances in this case do not leak anything to the adversary. Apart from these instances, the adversary receives labels sent by the honest parties in the third round. Since these labels are additively shared, they do not leak anything about the underlying messages in the virtual protocol. So, the outputs of any next-message function do not leak any information to a PPT adversary, and hence, the honest parties' inputs remain private.

[6] It is possible to design the garbling so that pre-defined input labels are used as inputs to the garbling. [LP09] gives such a construction.

4 Malicious Fall-back Secure MPC

In this section, we present a 4-round protocol that achieves malicious fall-back security with unanimous abort (UA). We do this by constructing a compiler that transforms any malicious statistically secure MPC protocol Π_{m} for $t < \frac{n}{2}$ into a protocol with malicious fall-back security $\Pi_{\mathsf{m\text{-}ua}}$. Our construction is similar to the semi-honest fall-back secure protocol. However, we require a few additional techniques to ensure that any deviation from the protocol results in a synchronized termination across all honest parties.

Specifically, we incorporate IT-MACs on the garbled labels, and the one-time multi-key IT-MACs on labels corresponding to broadcast labels, along with commitments and zero-knowledge proofs, ensuring unanimity among the honest parties. We then extend this construction to guarantee fairness in settings where a majority of parties are honest ($t < \frac{n}{2}$). This extension introduces an additional round of interaction, yielding a 5-round protocol $\Pi_{\mathsf{m\text{-}fair}}$ that provides the following simultaneous guarantees:

1. Fairness against an unbounded adversary when $t < \frac{n}{2}$.
2. Security with UA against any arbitrary PPT corruption with $t < n$.

Our 4-round protocol combines two primary components: a virtual protocol that is statistically secure against a malicious unbounded adversary, and a maliciously secure designated-party protocol that provides strong security guarantees to a designated party. As in the semi-honest case, we design a 2-round maliciously secure designated-party protocol by modifying the malicious compiler provided in [GS18], which provides the following security guarantees:

- Information-theoretic security against an unbounded malicious adversary corrupting any subset of parties excluding the designated party.
- Computational security against a malicious PPT adversary corrupting an arbitrary subset of parties (possibly including the designated party)

Assuming the designated party as P_i, we denote such an adversary structure as $\mathcal{Z}_{\mathsf{D}} = \{Z \subseteq \{P_j\}_{j\in[n]\setminus\{i\}}\}$. The construction and its details can be found in Appendix C. We provide details of our secure fall-back protocol below.

4.1 4-Round Unanimous Abort Protocol

We optimize the round complexity of $\Pi_{\mathsf{m\text{-}ua}}$ by instantiating both the virtual protocol Π_{m} and the designated-party protocol $\Pi_{\mathsf{m\text{-}des}}$ using constant-round constructions. [ACGJ19] presents a 2-round IT-secure MPC protocol in the plain model tolerating any $t < \frac{n}{2}$ malicious corruptions and achieving security with UA. We denote this protocol by:

$$\Pi_{\mathsf{m}} = \{\mathsf{NM}_i^j\}_{i\in[n],j\in[3]}$$

where NM_i^j denotes the next-message function of party P_i in round j, and NM_i^3 captures the post-interaction computation for output generation. To compile Π_{m}

into a fall-back secure protocol with UA, each next-message function is computed via an execution of the designated-party protocol $\Pi_{\mathsf{m\text{-}des}}$. As in the semi-honest setting, each real party P_i emulates a virtual party V_i executing Π_{m}.

We begin by discussing the challenges involved in designing a malicious fall-back protocol, along with the techniques used to overcome them. As discussed in Appendix C.1, we have a designated-party protocol that is maliciously secure, so that even an unbounded adversary cannot corrupt an honest party's next-message computation. Similar to the semi-honest protocol, each party's next-message functions in the virtual protocol Π_{m} are computed in a distributed way using instances of this designated-party protocol.

Randomness Generation: In the semi-honest protocol, the first next-message function is computed locally by each party, and parties interact only to compute the subsequent next-message functions. However, this cannot be done in the malicious case since the corrupt parties' random tapes may not be uniformly distributed. This means that in the computational case, up to $n-1$ parties' random tapes may follow some arbitrary distribution, and in such a case, the virtual protocol may not provide privacy. Hence, in order to achieve malicious security, each party's random tape needs to be additively secret shared among all the parties. As a result, the first next-message function is computed interactively through an instance of $\Pi_{\mathsf{m\text{-}des}}$, while the second and third next-message functions are computed by evaluating garbled circuits generated using $\Pi_{\mathsf{m\text{-}des}}$. The labels on which garbled circuits need to be evaluated are shared by all the parties in the third and fourth rounds, respectively. The generation of these garbled circuits, along with the evaluation of the first next-message function, is done through a single designated-party protocol instance, and parties compute and share the labels to evaluate these circuits in the subsequent rounds. We note that as in the semi-honest case, all the labels remain additively secret shared among the parties.

Authenticating Pairwise Labels: We next consider the challenge of ensuring the authenticity of these garbled circuits' input labels even against an unbounded adversary. To do this, we use IT-MACs to authenticate the input labels to the garbled circuits, and the party acting as the evaluator recovers a pair of tags for each input wire of its circuit. Since IT-MACs do not provide confidentiality, these tags are encrypted with random pads in order to hide the labels. The party sending the label also recovers the correct random pad and sends both these values to the evaluator, who decrypts the tags and checks the authenticity of the label. If the label shared corresponds to either 0 or 1, one of these checks is successful, and the evaluator proceeds to evaluate the circuit. Note that the tags received by the evaluator are permuted randomly so that the results of the MAC check reveal no information about the bit that the label encodes. Each party inputs two keys corresponding to bit values 0 and 1, which are used to compute the tags on the labels corresponding to bit values 0 and 1 respectively. These keys are internally permuted in the designated-party protocol instance before computing the tags. This is necessary to hide the bit being encoded by

the message, which is necessary in the computational case, wherein the view of each party in the virtual protocol must be hidden from a PPT adversary. For simplicity, we don't mention the key permutation explicitly in the protocol description.

Authenticating Broadcast Labels: This authentication suffices in the case of pairwise messages; however, for broadcast messages, we need some additional mechanism to ensure that the labels sent by the broadcasting party correspond to the same bit b in all the parties' garbled circuits. We resolve this by ensuring that each party's garbled circuit has the same pair of labels corresponding to a broadcasted bit. Note that the random pad used to encrypt the tag on such a broadcast label needs to be different in each party's circuit in order to hide the broadcast labels from a PPT adversary. In such a case, a pairwise IT-MAC scheme is insufficient for authenticating broadcast labels, since such a mechanism only guarantees pairwise consistency. In particular, a corrupt party broadcasting a label together with a vector of random pads could provide an incorrect random pad corresponding to one of the honest parties, while simultaneously providing correct random pads corresponding to the remaining honest parties. This leads to a selective abort,[7] thereby violating unanimity. Hence, we need some mechanism to ensure that all honest parties either abort together or proceed with the computation.

Towards this, for broadcast labels, a one-time multi-key IT-MAC scheme is used instead of a pairwise IT-MAC. For every broadcast bit, party P_i broadcasts the corresponding broadcast label along with a vector of n random pads, where the j-th component corresponds to the random pad of party P_j. Further, for every broadcast bit, each party P_j broadcasts its pair of encrypted tags corresponding to the two possible labels for each broadcast bit. These MAC tags are generated using a one-time multi-key IT-MAC scheme in which every party contributes a share of the MAC key. To verify the broadcast label, every party P_k uses the random pad corresponding to P_j to decrypt the pair of P_j's encrypted tags and checks the pair of tags with its share of the MAC key. If one of these decrypted tags verifies correctly, it proceeds with the computation and aborts otherwise.

Consistency Check: The final challenge is ensuring that the corrupt parties use the same shares in various instances of the designated-party protocol. To ensure this, we make each party broadcast a commitment on its input in the first round and then prove in zero-knowledge that it used the same input in all the instances. We now discuss the details of this commit-and-prove approach.

To ensure the consistency of the labels in all parties' garbled circuits, the corrupt parties must use consistent shares in the designated-party protocol instances. We do this by letting each P_i broadcast statistically binding commitments on its entire input (including the label shares) in all the designated-party protocol

[7] Selective abort refers to the ability of the adversary to abort the protocol depending on information learned during the execution, potentially causing the protocol to terminate only for specific inputs or parties.

instances except $\Pi^i_{\mathsf{m\text{-}des}}$, and provide zero-knowledge proofs to prove that it used consistent shares. Note that due to the computational hiding of the commitments, an unbounded adversary may learn all the label shares, and therefore, all the labels. This is not a problem because the labels encode the view of the virtual protocol, and the security of the protocol reduces to that of the virtual protocol.

However, note that the first round messages of $\Pi_{\mathsf{m\text{-}des}}$ are OT_1 messages of an OT scheme that provides IT security to the receiver. This means that such an OT_1 message provides no binding to the actual input used by the adversary, and it may get away with using inconsistent inputs. To resolve this, we let the adversary commit to the entire randomness (including the randomness used to generate OT_1 messages) that it uses in generating the first round messages of these designated-party protocol instances. This implies that the inputs of all the parties except the designated party are leaked to an unbounded adversary, but as argued above, this is not an issue.

We now describe the zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$ (4.1.1), which relies on the syntax of the designated-party protocol formalized in Protocol C.1.1. y denotes the input of a party P_i in a designated-party protocol instance with some designated party P_j ($j \neq i$), while r denotes the random tape it uses to compute its first message in that instance. In the context of our designated-party protocol construction, r is the randomness used in the $\mathsf{pre}(\cdot)$ function (denoted as r_{pre}), and in computing the OT_1 messages (denoted as r_{OT}). The set A denotes the steps in the conforming protocol where Φ, where P_i is supposed to perform computation. ω_y and ω_r denotes the randomness used to compute the commitment of y and r respectively. The NP relation R parameterizing $\mathcal{F}_{\mathsf{ZK}}$ simply checks if the first round messages $(z, \overline{\mathsf{ots}})$ and the commitments $\mathsf{Com}_y, \mathsf{Com}_r$ are generated correctly, given y, r and the commitment randomness ω_y, ω_r.

Functionality 4.1.1: $\mathcal{F}_{\mathsf{ZK}}$ for relation R with respect to $\Pi_{\mathsf{m\text{-}des}}$

The functionality is parametrized with an NP relation R defined as follows:
Inputs:
The prover and verifier have the common statement

$$\mathsf{x} = (\mathsf{Com_y}, \mathsf{Com_r}, (\mathsf{z}, \overline{\mathsf{ots}}), \mathsf{A})$$

where $A = \{(f_j, g_j, h_j)\}_{j\in[m]}$) The prover additionally has the witness

$$\mathsf{w} = ((y, \omega_y), (r, \omega_r))$$

Then $(\mathsf{x}, \mathsf{w}) \in R$ if the following holds:
Computation:

1. Parse $r = r_{\mathsf{pre}} \| r_{\mathsf{OT}}$ where $r_{\mathsf{OT}} = \{r^{j,\alpha,\beta}_{\mathsf{OT}}\}_{\alpha,\beta\in\{0,1\},j\in[\ell]}$
2. Compute $(z', v') = \mathsf{pre}(1^\kappa, i, y; r_{\mathsf{pre}})$
3. Let $|v'| = \ell$ and denote j-th bit of v' as v'_j. Compute

$$\overline{\mathsf{ots}}' = \mathsf{OT}_1(1^\kappa, v'_{h_j} \oplus \mathsf{NAND}(\alpha \oplus v'_{f_j}, \beta \oplus v'_{g_j}); r^{j,\alpha,\beta}_{\mathsf{OT}})\}$$

4. $(\mathsf{x}, \mathsf{w}) \in R$ if

$$(\overline{\mathsf{ots}}' = \overline{\mathsf{ots}}) \wedge (z' = z) \wedge (\mathsf{Com}_y = \mathsf{Com}_{\mathsf{SB}}(y; \omega_y)) \wedge (\mathsf{Com}_r = \mathsf{Com}_{\mathsf{SB}}(r; \omega_r))$$

Output:
$\mathcal{F}_{\mathsf{ZK}}$ outputs 1 to verifier if $(\mathsf{x}, \mathsf{w}) \in R$ and abort otherwise.

In our 4-round protocol, we instantiate $\mathcal{F}_{\mathsf{ZK}}$ with the 3-round UC-secure zero-knowledge protocol discussed in [CF01]. For the security of our protocol, the honest parties must know whether the corrupt parties have used consistent shares by round 3. If a corrupt party uses an inconsistent label or randomness shares, it means that the honest parties receive inconsistent garbled circuits, where the input labels for a broadcast bit differ in these circuits. An honest P_i receives the labels to evaluate its circuit $\widetilde{\mathsf{NM}}_i^2$ in round 3. Since this circuit isn't consistent with other honest parties' garbled circuits, the outputs of such inconsistent garbled circuits may leak something more to the adversary (like a selective abort attack). This implies that before an honest P_i evaluates $\widetilde{\mathsf{NM}}_i^2$, it needs to be convinced that the circuit it received is correct and consistent. This is ensured by the 3-round zero-knowledge protocol, which ends in round 3 before P_i broadcasts the labels output by $\widetilde{\mathsf{NM}}_i^2$.

Notation. The functionality for unanimous abort $\mathcal{F}_{\mathsf{ua}}$ is described in 4.1.2. We define the notation for our 4-round compiler that achieves malicious-fall-back security with UA and present the construction in Protocol 4.1.3. Within this construction, the next-message functions for each P_i are formally defined by the functionalities F_i, G_i, and H_i. In the protocol given by [ACGJ19], the parties communicate over pairwise channels only in the first round and use broadcast channels in both the rounds. We denote the message sent by P_i to P_j in the first round as T_{ij} and the message broadcasted by P_i in round r as b_i^r. We assume that every message sent in the protocol Π_{m} is ℓ bits long. The first next-message function generates pairwise and broadcast messages, while the second next-message function generates broadcast messages. The third next-message function takes broadcast messages from all parties and some private state information and obtains the output y.

Messages are represented using pairs of labels consisting of additive secret shares. For each bit of a message, we define labels L^b where $b \in \{0, 1\}$.

– **Pairwise Message Labels:** We consider each pairwise message T_{ij} to be ℓ bits long. For every pairwise message T_{ij} where $i, j \in [n]$, each party P_k holds ℓ pairs of label shares. For the h-th bit of T_{ij}, the share held by party P_k is denoted as $\mathsf{L}^b_{i,j,k,h}$, where $b \in \{0, 1\}$ and $h \in [\ell]$. The reconstructed label for the h-th bit is computed as the XOR summation:

$$\mathsf{L}^b_{i,j,h} = \bigoplus_{k=1}^{n} \mathsf{L}^b_{i,j,k,h}$$

The complete set of labels for the message T_{ij} is simplified as:

$$\mathsf{L}^b_{i,j} = \{\mathsf{L}^b_{i,j,h}\}_{h\in[\ell]}$$

– **Broadcast Message Labels:** A similar structure is applied to each broadcast message b^r_i, where $i \in [n]$ and $r \in \{1, 2\}$ is the round index. Each party P_k holds ℓ pairs of label shares for these messages. The share held by party P_k for the h-th bit of b^r_i is denoted as $\mathsf{L}^b_{i,\mathsf{BC},k,h,r}$, where $b \in \{0, 1\}$ and $h \in [\ell]$. The reconstructed label for the h-th bit is:

$$\mathsf{L}^b_{i,\mathsf{BC},h,r} = \bigoplus_{k=1}^{n} \mathsf{L}^b_{i,\mathsf{BC},k,h,r}$$

The set of labels for the broadcast message b^r_i is denoted as:

$$\mathsf{L}^b_{i,\mathsf{BC},r} = \{\mathsf{L}^b_{i,\mathsf{BC},h,r}\}_{h\in[\ell]}$$

– **Private State Labels:** The second round of the virtual protocol Π_{m} consists only of broadcast messages, but the output computation circuit of each party takes some private state information as input. We denote the labels corresponding to this private state as $\mathsf{L}^b_{i,\mathsf{st},h}$, and their respective additive shares as $\mathsf{L}^b_{i,\mathsf{st},k,h}$ for $k \in [n]$ and $h \in [\ell]$.

– **Randomness for Pairwise Messages:** For each label corresponding to a pairwise message, there is a corresponding randomness, which is used as a random pad to encrypt the tag on that label. As mentioned earlier, this is done in order to prevent a PPT adversary from learning labels other than the ones it receives in the protocol. Since we need this randomness to be hidden even when $n - 1$ parties are corrupted by a PPT adversary, it is additively shared among all the parties. For a label $\mathsf{L}_{i,j,h}$, we denote the corresponding randomness as $R_{i,j,h}$. The k-th share, i.e. P_k's additive share of such a randomness is denoted as $R_{i,j,k,h}$.

– **Randomness for Broadcast Messages:** Every broadcast message b^r_i sent by P_i, where $i \in [n]$ and $r \in \{1, 2\}$ denotes the round index, is assumed to be ℓ bits long. Although the broadcast labels are the same for all the parties, each party has a different random pad to encrypt the tag on these broadcast labels. For every $h \in [\ell]$, each party P_k holds shares of randomness for all P_j corresponding to both possible broadcast labels. For the h-th ($h \in [\ell]$) bit of b^r_i, the randomness share held by P_k corresponding to the label for bit $b \in \{0, 1\}$ and verifier party P_j is denoted by $R^b_{i,\mathsf{BC},j,k,h,r}$.
The random pad corresponding to P_j for a label broadcast by P_i is computed as the XOR summation of all the additive shares:

$$R^b_{i,\mathsf{BC},j,h,r} = \bigoplus_{k=1}^{n} R^b_{i,\mathsf{BC},j,k,h,r}$$

$$R^b_{i,\mathsf{BC},j,r} = \{R^b_{i,\mathsf{BC},j,h,r}\}_{h\in[\ell]}$$

To simplify notation, we denote $\boldsymbol{R}_{i,\mathsf{BC},h,r} = \{R_{i,\mathsf{BC},j,h,r}\}_{j\in[n]}$ in the protocol description and proof.

Commit-and-prove: We denote the input of a party P_i in the designated instance $\Pi^{j}_{\mathsf{m\text{-}des}}$ as $x_{i,j}$ (looking ahead, $x_{i,j}$ is the entire input of P_j in $\Pi^{F_j}_{\mathsf{m\text{-}des}}, \Pi^{G_j}_{\mathsf{m\text{-}des}}, \Pi^{H_j}_{\mathsf{m\text{-}des}}$), the random tape used by P_i for generating the first round messages in this instance as $r^{\mathsf{m-des}}_{i,j}$, and the first round messages of P_i as $m_{i,j} = (z_{i,j}, \overline{\mathsf{ots}}_{i,j})$. We also denote the commitments corresponding to the input $x_{i,j}$ as $\mathsf{c}_{x,i,j}$ and random tape $r^{\mathsf{m-des}}_{i,j}$ as $\mathsf{c}_{r,i,j}$. The zero-knowledge proof given by P_i in P_j's designated instance is denoted as $\mathsf{ZK}_{i,j}$, with the r-th round message in this proof being denoted as $\mathsf{zk}_{i,j,r}$ for $r \in [3]$. The witness in this proof is denoted as $\mathsf{w}_{i,j} = ((x_{i,j}, \omega_{x,i,j}), (r^{\mathsf{m-des}}_{i,j}, \omega_{r,i,j}))$ following the notation used in $\mathcal{F}_{\mathsf{ZK}}$.

Functionality 4.1.2: $\mathcal{F}_{\mathsf{ua}}$ corresponding to function f

Inputs:
Each party P_k has some input x_k.

Output:
$\mathcal{F}_{\mathsf{ua}}$ computes $y = f(x_1, ..., x_n)$ and sends y to the adversary $\mathcal{A}$. $\mathcal{F}_{\mathsf{ua}}$ sends y to the honest parties if it receives $\mathsf{continue}$ from $\mathcal{A}$, else sends abort.

Protocol 4.1.3: 4-Round Fall-Back Secure protocol $\Pi_{\mathsf{m\text{-}ua}}$ realizing $\mathcal{F}_{\mathsf{ua}}$

Setting: Let $P = \{P_i\}_{i\in[n]}$ be set of parties. $\Pi_{\mathsf{m}} = \{\mathsf{NM}^1_i, \mathsf{NM}^2_i, \mathsf{NM}^3_i\}_{i\in[n]}$

Building Blocks:

- Garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ (Definition A.15)
- Malicious designated protocol (Protocol C.1.1)
- IT-MAC scheme $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Verify})$ (Definition A.10)
- One-time Multi-key IT-MAC scheme $(\mathsf{MK.KeyGen}, \mathsf{MK.Sign}, \mathsf{MK.Verify})$ (Definition A.11)
- UC-secure equivocable, IT-binding commitment scheme $\mathsf{Com}_{\mathsf{SB}}$ (Definition A.12)
- UC-secure 3-round zero-knowledge protocol $\mathsf{ZK} = (\mathsf{ZK}_1, \mathsf{ZK}_2, \mathsf{ZK}_3, \mathsf{ZK}_4)$ (Definition A.14)
- Common Reference String (CRS)

Input: Each party P_k with input x_k samples

$$\Sigma_k := \{\mathsf{L}^b_{i,j,k,h}, \mathsf{L}^b_{i,\mathsf{BC},k,h,r}, R^b_{i,j,k,h}, R^b_{i,\mathsf{BC},j,k,h,r}\}_{i,j\in[n],h\in[\ell],b\in\{0,1\},r\in[2]}$$

the additive secret shares of the random tape $\{r_{i,k}\}_{i\in[n]}$, MAC keys for the IT-MAC as well as the one-time multi-key IT-MAC.

$$\{K^b_{i,k,h}, \mathsf{K}^b_{i,\mathsf{BC},j,k,h,r}\}_{i,j\in[n],h\in[\ell],b\in\{0,1\},r\in[2]}$$

where each $K \leftarrow \mathsf{MAC.KeyGen}(1^s)$ and $\mathsf{K} \leftarrow \mathsf{MK.KeyGen}(1^s)$.

The Protocol:

– **Round 1:** Each party P_k does the following:
  1. Compute and broadcast commitments on Σ_k and $\{r_{i,k}\}_{i\in[n]\setminus\{k\}}$.
  2. Begin $\Pi^{f_k}_{\mathsf{m\text{-}des}}$, with P_k as the designated party, where $f_k = (F_k, G_k, H_k)$ as given in Functionalities 4.1.4,4.1.5 and 4.1.6.
  3. Participate in other instances of $\Pi^{f_j}_{\mathsf{m\text{-}des}}$ with P_j as the designated party.
  4. For all $j \in [n] \setminus \{k\}$,
     - Sends P_j
     $$\mathsf{zk}_{k,j,1} \leftarrow \mathsf{ZK}_1(\mathsf{c}_{x,k,j}, \mathsf{c}_{r,k,j}, m_{k,j}, \mathsf{w}_{k,j})$$
     - Receive $\mathsf{zk}_{j,k,1}$ and commitments on $\Sigma_j, \{r_{i,j}\}_{i\neq j}$ from P_j.

– **Round 2:** Each party P_k does the following:
  1. Obtains output of F_k from $\Pi^{f_k}_{\mathsf{m\text{-}des}}$:
  $$\{\{\mathsf{L}_{k,j}, R_{k,j}\}_{j\in[n]}, \mathsf{L}_{k,\mathsf{BC},1}, \{\boldsymbol{R}_{k,\mathsf{BC},h,1}\}_{h\in[\ell]}\}$$
  2. Obtains the garbled circuits, decoding information and encrypted tags as output from $\Pi^{G_k}_{\mathsf{m\text{-}des}}$ and $\Pi^{H_k}_{\mathsf{m\text{-}des}}$:
  $$\left(\widetilde{\mathsf{NM}}^2_k, (\widetilde{\mathsf{NM}}^3_k, d^3_k), \{w_{j,k,h}, w_{j,\mathsf{BC},k,h,1}, w_{j,\mathsf{BC},k,h,2}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}\right)$$
  3. For all $j \in [n] \setminus \{k\}$,
     - Compute and sends to P_j
     $$\mathsf{zk}_{j,k,2} \leftarrow \mathsf{ZK}_2(\mathsf{zk}_{j,k,1}, \mathsf{c}_{x,j,k}, \mathsf{c}_{r,j,k}, m_{j,k})$$
     - Receive $\mathsf{zk}_{k,j,2}$ from P_j.

– **Round 3:** Each party P_k does the following:
  1. Sends label shares and randomness. For all $j \in [n]$:
  $$\begin{aligned} \text{Send to } P_j: &\quad \{\mathsf{L}_{k,j}, R_{k,j}\} \\ \text{Broadcast}: &\quad \{\mathsf{L}_{k,\mathsf{BC},1}, \boldsymbol{R}_{k,\mathsf{BC},h,1}, \{w_{j,\mathsf{BC},k,h,1}\}_{h\in[\ell]}\end{aligned}$$
  2. Receive from P_j for $j \in [n]$
  $$\{\mathsf{L}_{j,k}, \mathsf{L}_{j,\mathsf{BC},1}, R_{j,k}, \boldsymbol{R}_{j,\mathsf{BC},h,1}, \{w_{i,\mathsf{BC},j,h,1}\}_{i\in[n]}\}_{h\in[\ell]}$$
  3. Perform label verification for all $j \in [n], h \in [\ell]$ i.e. check if
  $$\mathsf{MAC.Verify}(K_{j,k,h}, \mathsf{L}_{j,k,h}, w^b_{j,k,h} \oplus R^b_{j,k,h}) = 1$$
  for some $b \in \{0, 1\}$. If not, broadcast abort.
  4. Perform broadcast label verification, i.e. check if
  $$\mathsf{MK.Verify}(K_{j,\mathsf{BC},i,k,h,1}, \mathsf{L}_{j,\mathsf{BC},h,1}, w^b_{j,\mathsf{BC},i,h,1} \oplus R^b_{j,\mathsf{BC},i,h,1}) = 1$$
  some $b \in \{0, 1\}$ and for all $i, j \in [n], h \in [\ell]$. If not, broadcast abort.

5. Evaluate the circuit $\widetilde{\mathsf{NM}}_k^2$ to obtain the output as

$$\{\mathsf{L}_{k,\mathsf{BC},h,2} \| \boldsymbol{R}_{k,\mathsf{BC},h,2}\}_{h\in[\ell]} \leftarrow \mathsf{Ev}(\widetilde{\mathsf{NM}}_k^2, \{\mathsf{L}_{j,k}\}_{j\in[n]}, \mathsf{L}_{j,\mathsf{BC},1})$$

6. For all $j \in [n] \setminus \{k\}$,
   - Compute and send to P_j

$$\mathsf{zk}_{k,j,3} \leftarrow \mathsf{ZK}_2(\{\mathsf{zk}_{k,j,r}\}_{r\in[2]}, \mathsf{c}_{x,k,j}, \mathsf{c}_{r,k,j}, m_{k,j}, \mathsf{w}_{k,j})$$

   - Receive $\mathsf{zk}_{j,k,3}$ from P_j and compute

$$b_{j,k} \leftarrow \mathsf{ZK}_4(\{\mathsf{zk}_{j,k,r}\}_{r\in[3]}, \mathsf{c}_{x,j,k}, \mathsf{c}_{r,j,k}, m_{j,k})$$

   If $b_{j,k} = 0$ for any $j \neq k$, broadcast abort and terminate.

– **Round 4:** Each party P_k does the following:
1. Broadcast $\{\mathsf{L}_{k,\mathsf{BC},h,2} \| \boldsymbol{R}_{k,\mathsf{BC},h,2}, \{w_{j,\mathsf{BC},k,h,2}\}_{j\in[n]}\}_{h\in[\ell]}$.
2. Receive $\{\mathsf{L}_{j,\mathsf{BC},h,2} \| \boldsymbol{R}_{j,\mathsf{BC},h,2}, \{w_{i,\mathsf{BC},j,h,2}\}_{i\in[n]}\}_{h\in[\ell]}$ from P_j for all $j \in [n]$.
3. Perform MAC verification on broadcast labels i.e. check if

$$\mathsf{MK.Verify}(\mathsf{K}^b_{i,\mathsf{BC},j,k,h,2}, \mathsf{L}_{i,\mathsf{BC},h,2}, w^b_{i,\mathsf{BC},j,h,2} \oplus R^b_{i,\mathsf{BC},j,h,2}) = 1$$

   for some $b \in \{0,1\}$ and all $i, j \in [n], h \in [\ell]$. If it fails, P_k broadcasts abort.
4. Evaluate the circuit $\widetilde{\mathsf{NM}}_k^3$ to obtain the output y.

If in any round P_i receives abort through the broadcast channel, it aborts the protocol.

Functionality 4.1.4: F_i **corresponding to** NM_i^1 **in** Π_{m}

Inputs:
Each party P_k has:

– A randomly sampled value $r_{i,k}$.
– $2n\ell$ pairs of random shares $\{R^b_{i,j,k,h}, R^b_{i,\mathsf{BC},j,k,h,1}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$.
– $(n+1)\ell$ pairs of label shares $\{\{\mathsf{L}^b_{i,j,k,h}\}_{j\in[n]}, \mathsf{L}^b_{i,\mathsf{BC},k,h,1}\}_{h\in[\ell],b\in\{0,1\}}$.

P_i additionally has its protocol input x_i.
Computation:

1. Compute for all $j \in [n], h \in [\ell], b \in \{0,1\}$:

$$\mathsf{L}^b_{i,j,h} = \bigoplus_{k=1}^n \mathsf{L}^b_{i,j,k,h} \qquad R^b_{i,j,h} = \bigoplus_{k=1}^n R^b_{i,j,k,h}$$

$$\mathsf{L}^b_{i,\mathsf{BC},h,1} = \bigoplus_{k=1}^n \mathsf{L}^b_{i,\mathsf{BC},k,h,1} \qquad R^b_{i,\mathsf{BC},j,h,1} = \bigoplus_{k=1}^n R^b_{i,\mathsf{BC},j,k,h,1}$$

2. Compute $r_i = \bigoplus_{j=1}^n r_{i,j}$ and compute NM_i^1.

$$(\{T_{ij}\}_{j\in[n]}, \mathsf{b}_i^1) = \mathsf{NM}_i^1(x_i, r_i)$$

3. For each $j \in [n]$, parse $T_{ij} = z_{j,1}||z_{j,2}||\ldots||z_{j,\ell}$ where $z_{j,h} \in \{0,1\}$ for all $h \in [\ell]$, and compute the labels as $\mathsf{L}_{i,j} = \{\mathsf{L}_{i,j,h}^{z_{j,h}}\}_{h\in[\ell]}$ and the corresponding randomness as $R_{i,j} = \{R_{i,j,h}^{z_{j,h}}\}_{h\in[\ell]}$.
4. Parse $\mathsf{b}_i^1 = z_{i,1}||z_{i,2}||\ldots||z_{i,\ell}$ where $z_{i,h} \in \{0,1\}$ for all $h \in [\ell]$, and compute the labels as $\mathsf{L}_{i,\mathsf{BC},1} = \{\mathsf{L}_{i,\mathsf{BC},h,1}^{z_{i,h}}\}_{h\in[\ell]}$ and the corresponding randomness for all $j \in [n], h \in [\ell]$ as $R_{i,\mathsf{BC},j,h,1} = R_{i,\mathsf{BC},j,h,1}^{z_{i,h}}$.

Output:
F_i gives $(\{\mathsf{L}_{i,j}, R_{i,j},\}_{j\in[n]}, \mathsf{L}_{i,\mathsf{BC},1}, \{\boldsymbol{R}_{i,\mathsf{BC},h,1}\}_{h\in[\ell]})$ to P_i.

Functionality 4.1.5: G_i corresponding to NM_i^2 in Π_{m}

Inputs:
Each party P_k has:

- $2(n+1)\ell$ pairs of labels shares, denoted as:

$$\{\{\mathsf{L}_{j,i,k,h}^b, \mathsf{L}_{j,\mathsf{BC},k,h,1}^b\}_{j\in[n]}, \mathsf{L}_{i,\mathsf{BC},k,h,2}^b, \mathsf{L}_{i,\mathsf{st},k,h}^b\}_{h\in[\ell],b\in\{0,1\}}$$

- $3n\ell$ pairs of random shares, denoted as:

$$\{R_{j,i,k,h}^b, R_{j,\mathsf{BC},i,k,h,1}^b, R_{i,\mathsf{BC},j,k,h,2}^b\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$$

- $n\ell$ MAC keys $\{\mathsf{K}_{j,\mathsf{BC},i,k,h,1}^b\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$, where $\mathsf{K} \leftarrow \mathsf{MK.KeyGen}(1^s)$.

P_i additionally has $n\ell$ MAC keys as $\{K_{j,i,h}^b\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$, where $K \leftarrow \mathsf{MAC.KeyGen}(1^s)$.
Computation:

1. Compute for all $j \in [n], h \in [\ell], b \in \{0,1\}$:

$$\mathsf{L}_{j,i,h}^b = \bigoplus_{k=1}^n \mathsf{L}_{j,i,k,h}^b \qquad R_{j,i,h}^b = \bigoplus_{k=1}^n R_{j,i,k,h}^b$$

$$\mathsf{L}_{j,\mathsf{BC},h,1}^b = \bigoplus_{k=1}^n \mathsf{L}_{j,\mathsf{BC},k,h,1}^b \qquad R_{j,\mathsf{BC},i,h,1}^b = \bigoplus_{k=1}^n R_{j,\mathsf{BC},i,k,h,1}^b$$

$$\mathsf{L}_{i,\mathsf{st},h}^b = \bigoplus_{k=1}^n \mathsf{L}_{i,\mathsf{st},k,h}^b \qquad R_{i,\mathsf{BC},j,h,2}^b = \bigoplus_{k=1}^n R_{i,\mathsf{BC},j,k,h,2}^b$$

$$\mathsf{L}_{i,\mathsf{BC},h,2}^b = \bigoplus_{k=1}^n \mathsf{L}_{i,\mathsf{BC},k,h,2}^b$$

2. Generate a garbled circuit for the function NM_i^2 with input labels as $\{\mathsf{L}_{j,i,h}^b, \mathsf{L}_{j,\mathsf{BC},h,1}^b\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$ and output labels[8] as

$$\mathsf{L}_{i,\mathsf{st},h}^b, \{\mathsf{L}_{i,\mathsf{BC},h,2}^b \| \boldsymbol{R}_{i,\mathsf{BC},h,2}^b\}_{h\in[\ell],b\in\{0,1\}}$$

Denote this circuit as $(\widetilde{\mathsf{NM}}_i^2, e_i^2, d_i^2)$.

[8] We can use a garbling scheme like [LP09] to encrypt $L\|R$ in the output label

3. Compute the encrypted MAC tags for $j \in [n], h \in [\ell], b \in \{0,1\}$ as:

$$w^b_{j,i,h} = R^b_{j,i,h} \oplus \mathsf{MAC.Tag}(K^b_{j,i,h}, \mathsf{L}^b_{j,i,h})$$
$$w^b_{j,\mathsf{BC},i,h,1} = R^b_{j,\mathsf{BC},i,h,1} \oplus \mathsf{MK.Sign}(\{K^b_{j,\mathsf{BC},i,k,h,1}\}_{k\in[n]}, \mathsf{L}^b_{j,\mathsf{BC},h,1})$$

4. For each $j \in [n], h \in [\ell]$, randomly permute $(w^0_{j,i,h}, w^1_{j,i,h})$ and $(w^0_{j,\mathsf{BC},i,h,1}, w^1_{j,\mathsf{BC},i,h,1})$, and denote these permuted tuples as $w_{j,i,h}$ and $w_{j,\mathsf{BC},i,h,1}$.

Output:
G_i gives $(\widetilde{\mathsf{NM}}^2_i, \{w_{j,i,h}, w_{j,\mathsf{BC},i,h,1}\}_{j\in[n],h\in[\ell]})$ to P_i.

Functionality 4.1.6: H_i corresponding to NM^3_i in Π_{m}

Inputs:
Each party P_k has:

- $(n+1)\ell$ pairs of label shares denoted as:

$$\{\{\mathsf{L}^b_{j,\mathsf{BC},k,h,2}\}_{j\in[n]}, \mathsf{L}^b_{i,\mathsf{st},k,h}\}_{h\in[\ell],b\in\{0,1\}}$$

- $n\ell$ pairs of random shares $\{R^b_{j,\mathsf{BC},i,k,h,2}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$.
- $n\ell$ pairs of MAC keys, denoted as $\{\mathsf{K}^b_{j,\mathsf{BC},i,k,h,2}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}$, where $\mathsf{K} \leftarrow \mathsf{MK.KeyGen}(1^s)$.

Computation:

1. Compute for all $j \in [n], h \in [\ell], b \in \{0,1\}$

$$\mathsf{L}^b_{j,\mathsf{BC},h,2} = \bigoplus_{k=1}^{n} \mathsf{L}^b_{j,\mathsf{BC},k,h,2} \qquad R^b_{j,\mathsf{BC},i,h,2} = \bigoplus_{k=1}^{n} R^b_{j,\mathsf{BC},i,k,h,2}$$
$$\mathsf{L}^b_{i,\mathsf{st},h} = \bigoplus_{k=1}^{n} \mathsf{L}^b_{i,\mathsf{st},k,h}$$

2. Generate a garbled circuit $(\widetilde{\mathsf{NM}}^3_i, e^3_i, d^3_i)$ with input labels $\{\{\mathsf{L}^b_{j,\mathsf{BC},h,2}\}_{j\in[n]}, \mathsf{L}^b_{i,\mathsf{st},h}\}_{h\in[\ell],b\in\{0,1\}}$.
3. Compute the encrypted MAC tags for $j \in [n], h \in [\ell], b \in \{0,1\}$ as

$$w^b_{j,\mathsf{BC},i,h,2} = R^b_{j,\mathsf{BC},i,h,2} \oplus \mathsf{MK.Sign}(\{K^b_{j,\mathsf{BC},i,k,h,2}\}_{k\in[n]}, \mathsf{L}^b_{j,\mathsf{BC},h,2})$$

4. For all $j \in [n], h \in [\ell]$, randomly permute $(w^0_{j,\mathsf{BC},i,h,2}, w^1_{j,\mathsf{BC},i,h,2})$, and denote this permuted tuple as $w_{j,\mathsf{BC},i,h,2}$.

Output:
H_i gives $(\widetilde{\mathsf{NM}}^3_i, d^3_i), \{w_{j,\mathsf{BC},i,h,2}\}_{j\in[n],h\in[\ell]}$ to P_i.

Theorem 4.1.7 (4-Round Malicious Fall-back Compiler)**.** *Let Π_{m} be an n-party protocol that securely computes a functionality f against unbounded malicious adversaries with corruption threshold $t < \frac{n}{2}$. Let $\Pi_{\mathsf{m\text{-}des}}$ be an n-party protocol achieving fall-back security with respect to the adversary structure $\mathcal{Z}_{\mathsf{D}}$. Assuming the existence of:*

- *a garbling scheme*
- *IT-MACs, one-time multi-key IT-MACs*
- *a UC-secure equivocable statistically binding commitment scheme, and*
- *a UC-secure 3-round unconditionally sound zero-knowledge protocol realizing $\mathcal{F}_{\mathsf{ZK}}$,*

there exists a 4-round protocol $\Pi_{\mathsf{m\text{-}ua}}$ in the CRS model that fall-back securely computes $\mathcal{F}_{\mathsf{ua}}$ against malicious adversaries tolerating $t < \frac{n}{2}$ corruptions.

We provide informal reasoning for the protocol's security here. The formal proof can be found in the Appendix E.2.

Proof Outline. We describe how the protocol $\Pi_{\mathsf{m\text{-}ua}}$ achieves fall-back security. We first consider the case where at most $t < \frac{n}{2}$ parties are corrupted by an unbounded adversary. In this setting, we argue that the security of our protocol reduces to that of the underlying statistically secure protocol Π_{m}. In the protocol $\Pi_{\mathsf{m\text{-}ua}}$, each party computes its next-message function in Π_{m} via an instance of the designated-party protocol. When the designated party is honest, the security guarantees of $\Pi_{\mathsf{m\text{-}des}}$ ensure that the adversary learns nothing about the inputs of the honest parties. Thus, their view in the protocol Π_{m} remains hidden.

On the other hand, if the designated party is corrupted, $\Pi_{\mathsf{m\text{-}des}}$ provides no security guarantees, and the adversary may learn the inputs of the honest parties. However, in such instances, the inputs of the honest parties correspond to the messages (or the associated labels) that they send to the corrupted parties in Π_{m}. Therefore, the privacy of the honest parties' inputs in these cases reduces to their privacy in the protocol Π_{m}.

In the computational setting, the designated-party protocol $\Pi_{\mathsf{m\text{-}des}}$ used to evaluate the next-message function of Π_{m} provides computational security against any PPT adversary corrupting an arbitrary subset of parties. Consequently, these designated-party protocol instances do not leak any information to the adversary. Beyond these instances, the adversary also observes the labels sent by the honest parties in the third and fourth rounds. However, since these labels are additively shared, they reveal no information about the underlying messages in the virtual protocol. We also have encrypted IT-MAC tags on each of the pairwise message labels sent in the protocol, which ensures the authenticity of the label sent. Similarly, a one-time multi-key IT-MAC ensures the authenticity of broadcast labels. Therefore, the outputs of the next-message functions are correct, and do not leak any information to a PPT adversary. Thus, the protocol $\Pi_{\mathsf{m\text{-}ua}}$ is fall-back secure.

4.2 5-Round Fair Protocol

We construct a 5-round fair protocol using t-out-of-n threshold secret-sharing scheme together with a one-time multi-key IT-MAC. Our goal is to guarantee fairness when $t < \frac{n}{2}$, while ensuring UA when $t < n$. Note that a t-out-of-n secret-sharing scheme (Definition A.9) ensures privacy against any coalition of at most t parties, and we use such a scheme due to Shamir [Sha79].

We begin by considering functionalities that output a common value y to all parties. Any functionality that produces distinct outputs $(y_1, \ldots, y_n)$ can be transformed into one with a common output as follows. Each party P_i provides an additional random input r_i, and the functionality outputs to all parties:

$$y = (y_1 \oplus r_1) \,\|\, (y_2 \oplus r_2) \,\|\, \cdots \,\|\, (y_n \oplus r_n)$$

In our construction, the functionality is modified to output a t-out-of-n secret-sharing of y, i.e. each party P_i receives a share $[y]_i^t$. Additionally, each party receives tags on all shares, generated using a one-time multi-key IT-MAC scheme. The protocol proceeds identically to $\Pi_{\mathsf{m\text{-}ua}}$, with an additional final round in which all parties broadcast their shares, and reconstruct their output.

By the properties of $\Pi_{\mathsf{m\text{-}ua}}$, at the end of round 4, every honest party either obtains its correct share or outputs $\perp$. Consider first the case when $t < \frac{n}{2}$. If all honest parties obtain valid shares and broadcast them, then reconstruction succeeds since at least $n - t > \frac{n}{2}$ correct shares are available. On the other hand, if honest parties output $\perp$, then the adversary controls at most t shares, which is insufficient to reconstruct y. This guarantees fairness in this regime.

For the case $t < n$, additional care is required to ensure correctness. A malicious adversary controlling more than t parties could obtain valid shares at the end of round 4 and then broadcast inconsistent shares corresponding to some $y' \neq y$ in round 5, thereby violating correctness for honest parties. This is prevented by authenticating the shares using tags, so that an honest party can detect a corrupt party's incorrect share.

Specifically, we augment the functionality so that each share $[y]_i^t$ is authenticated using a one-time multi-key IT-MAC, where each party inputs a key corresponding to each share. Thus, each P_i inputs a total of n keys, each of which is used to authenticate one of the n shares.

Formally, this is described in Functionality 4.2.2 where each party P_i provides augmented input $(x_i, \{\mathsf{K}_{i,j}\}_{j\in[n]})$, and receives output $([y]_i^t, \tau_i)$, where

$$\tau_i \leftarrow \mathsf{MK.Sign}(\{\mathsf{K}_{j,i}\}_{j\in[n]}, [y]_i^t)$$

These tags ensure that any malicious behavior in the broadcast phase is detected, thereby enforcing consistency and preserving correctness even when $t < n$. We describe the 5-round fair and UA protocol in Protocol 4.2.3, and the fairness functionality in 4.2.1.

Functionality 4.2.1: $\mathcal{F}_{\mathsf{fair}}$ corresponding to function f

Inputs:
Each party P_i has input x_i.
Output:
$\mathcal{F}_{\mathsf{fair}}$ computes $y = f(x_1, ..., x_n)$. If it receives $\perp$ from $\mathcal{A}$, it sends $\perp$ to all parties. Else, $\mathcal{F}_{\mathsf{fair}}$ sends y to all parties.

Functionality 4.2.2: $\mathcal{F}_{\mathsf{ss\text{-}ua}}$

Inputs:
Each party P_i has input x_i and one-time multi-key IT-MAC keys $\{\mathsf{K}_{i,j}\}_{j\in[n]}$, where each $\mathsf{K} \leftarrow \mathsf{MK.KeyGen}(1^s)$.
Computation:

1. Compute $y = f(x_1, ..., x_n)$.
2. Compute the Shamir sharing of y as $([y]_1^t, ..., [y]_n^t) \leftarrow \mathsf{SS.Share}(y)$.
3. For all $i \in [n]$, compute $\tau_i = \mathsf{MK.Sign}(\{\mathsf{K}_{j,i}\}_{j\in[n]}, [y]_i^t)$.

Output:
$\mathcal{F}_{\mathsf{ss\text{-}ua}}$ sends $\{([y]_i^t, \tau_i)\}_{P_i\in Z}$ to the adversary $\mathcal{A}$. If $\mathcal{F}_{\mathsf{ss\text{-}ua}}$ receives abort from $\mathcal{A}$, it sends abort to the honest parties. Else, it sends $([y]_i^t, \tau_i)$ to each honest P_i.

Protocol 4.2.3: 5-Round Fall-Back Secure protocol $\Pi_{\mathsf{m\text{-}fair}}$

Setting: Let $P = \{P_i\}_{i\in[n]}$ be set of parties.

Input: Each party P_i has its input x_i and one-time multi-key IT-MAC keys $\{\mathsf{K}_{i,j}\}_{j\in[n]}$, where each $\mathsf{K} \leftarrow \mathsf{MK.KeyGen}(1^s)$.

The Protocol:

- All parties emulate $\mathcal{F}_{\mathsf{ss\text{-}ua}}$ as given in Functionality 4.2.2.
- Each party P_i broadcasts its output $([y]_i^t, \tau_i)$.

Output: Every party P_i does the following:

1. Receive $([y]_j^t, \tau_j)$ for all $j \in [n]$.
2. For each $j \in [n]$, compute $b_{i,j} = \mathsf{MK.Verify}(\mathsf{K}_{i,j}, [y]_j^t, \tau_j)$.
3. Let $\mathcal{T} = \{k \mid b_{i,k} = 1\}$. If $|\mathcal{T}| \geq t + 1$, output $y = \mathsf{SS.Recon}(\{[y]_k^t\}_{k\in\mathcal{T}})$. Else, terminate the computation and output abort.

Theorem 4.2.4 (5-Round Malicious Fall-back with Fairness Compiler)**.** *Let there exist a 4-round malicious fall-back secure protocol in the CRS model that realizes $\mathcal{F}_{\mathsf{ss\text{-}ua}}$ tolerating $t < \frac{n}{2}$ malicious corruptions. Then, there exists a 5-round protocol $\Pi_{\mathsf{m\text{-}fair}}$ in the CRS model that securely realizes,*

- *Fair: $\mathcal{F}_{\mathsf{fair}}$ tolerating $t < \frac{n}{2}$ corruptions by an unbounded malicious adversary.*

– *UA:* $\mathcal{F}_{\mathsf{ua}}$ *tolerating* $t < n$ *corruptions by a PPT malicious adversary.*

We provide the proof sketch here. Formal proof can be found in Appendix E.3.
Proof Outline. The security of $\Pi_{\mathsf{m\text{-}fair}}$ is formally grounded in the properties of the underlying $\Pi_{\mathsf{m\text{-}ua}}$ protocol. We first consider the setting of $t < \frac{n}{2}$ corruptions, where $\Pi_{\mathsf{m\text{-}fair}}$ guarantees fairness. The security of the protocol in this case reduces to the security of the $\Pi_{\mathsf{m\text{-}ua}}$ protocol in the statistical setting. The parties obtain t-out-of-n Shamir shares of the function output by executing an instance of $\Pi_{\mathsf{m\text{-}ua}}$. These shares are authenticated using one-time multi-key IT-MACs i.e. each party also receives a tag on its share. When each party broadcasts its share, it also broadcasts this tag, which enables other parties to check if the share is correct. Since $t < \frac{n}{2}$, the shares held and broadcast by honest parties are sufficient to reconstruct the output, thus ensuring that all honest parties reconstruct the output. If the honest parties do not receive their shares from $\Pi_{\mathsf{m\text{-}ua}}$, the corrupt parties (numbering at most t) cannot reconstruct the output either, as they lack the required number of shares. This ensures fairness in the protocol.

When $t \geq \frac{n}{2}$ corruptions, the protocol transitions to providing security with UA. The security of the protocol again reduces to the computational case of $\Pi_{\mathsf{m\text{-}ua}}$. In this case, the adversary recovers all the corrupt parties' shares and can reconstruct the output without the honest shares. In this case, we show that the adversary cannot violate the correctness of the protocol, and the honest parties either abort or obtain the correct function output. This follows from the unforgeability of the one-time multi-key IT-MAC scheme. As explained above, all honest parties can detect incorrect shares, and thus, they all either recover the function output (if they get $\geq t + 1$ correct shares) or abort.

Acknowledgments. We thank Yuval Ishai and Benny Applebaum for their valuable insights and discussions towards the difficulty of presenting lower bounds. Anasuya Acharya was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 101124977 (DECRYPSYS). Divya Ravi is funded by NWO Talent Programme Veni (VI.Veni.242.295) and by NWO Gravitation Project Challenges in Cyber Security (CiCS) with file number 024.006.037. Aditya Patankar and Arpita Patra are supported by the Centre of Excellence in Cybersecurity (CySecK - FSID).

References

ABT18. Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect secure computation in two rounds. In *TCC 2018*, pages 152–174, 2018.

ACGJ18. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *Annual international cryptology conference*, pages 395–424. Springer, 2018.

ACGJ19. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In *EUROCRYPT 2019*, pages 532–561, 2019.

AHPV23. Anasuya Acharya, Carmit Hazay, Oxana Poburinnaya, and Muthuramakrishnan Venkitasubramaniam. Best of both worlds - revisiting the spymasters double agent problem. In *CRYPTO 2023*, pages 328–359, 2023.

AHV25. Anasuya Acharya, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. On achieving "best-in-the-multiverse" MPC. In *TCC 2025*, pages 119–151, 2025.

AKP20a. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The resiliency of mpc with low interaction: the benefit of making errors. In *Theory of Cryptography Conference*, pages 562–594. Springer, 2020.

AKP20b. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of perfect mpc with active security and optimal resiliency. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

AKP22. Benny Applebaum, Eliran Kachlon, and Arpita Patra. Round-optimal honest-majority mpc in minicrypt and with everlasting security. In *Theory of Cryptography Conference*, pages 103–120. Springer, 2022.

AKP23. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of statistical mpc with optimal resiliency. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1527–1536, 2023.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, 2012.

BM89. Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications, 1989.

CCG+20. Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In *TCC 2020*, pages 291–319, 2020.

CDvdG87. David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO 1987*, pages 87–119, 1987.

CF01. Ran Canetti and Marc Fischlin. Universally composable commitments. In *Annual International Cryptology Conference*, pages 19–40. Springer, 2001.

Cha89. David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO 1989*, pages 591–602, 1989.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

FHHW03. Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-threshold broadcast and detectable multi-party computation. In *EUROCRYPT*, pages 51–67, 2003.

FHW04. Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. Multi-party computation with hybrid security. In *EUROCRYPT 2004*, pages 419–438, 2004.

GIKR02. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO 2002*, pages 178–193, 2002.

GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO 2015*, pages 63–82, 2015.

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

GMW86. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO 1986*, pages 171–185, 1986.

GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Advances in Cryptology – EUROCRYPT 2018*, pages 468–499, Cham, 2018. Springer International Publishing.

HIKR23. Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In *Annual International Cryptology Conference*, pages 203–235. Springer, 2023.

HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO 2011*, pages 132–150, 2011.

HM97. Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC 1997*, pages 25–34, 1997.

IKK+11. Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput.*, 40(1):122–141, 2011.

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

Kat07. Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 11–20, 2007.

KM20. Dakshita Khurana and Muhammad Haris Mughees. On statistical security in two-party computation. In *TCC 2020*, pages 532–561, 2020.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of cryptology*, 22(2):161–188, 2009.

LRM10. Christoph Lucas, Dominik Raub, and Ueli M. Maurer. Hybrid-secure MPC: trading information-theoretic robustness for computational privacy. In *PODC 2010*, pages 219–228, 2010.

PR18. Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In *CRYPTO 2018*, pages 425–458, 2018.

PS21. Arpita Patra and Akshayaram Srinivasan. Three-round secure multiparty computation from black-box two-round oblivious transfer. In *Annual International Cryptology Conference*, pages 185–213. Springer, 2021.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Annual international cryptology conference*, pages 554–571. Springer, 2008.

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

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, 1986.

Supplementary Material

A Additional Preliminaries

Definition A.1. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ defined over a finite domain D are **statistically indistinguishable**, denoted $X \stackrel{s}{\equiv} Y$, if every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Delta(X_n, Y_n) < \frac{1}{p(n)}$$

where,

$$\Delta(X_n, Y_n) = \frac{1}{2} \cdot \sum_{\alpha\in D} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|$$

Definition A.2. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are **computationally indistinguishable**, denoted $X \stackrel{c}{\equiv} Y$, if for every PPT distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

Secure Multiparty Computation. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be a set of n parties. MPC involves the computation of a (possibly randomized) function that maps a tuple of inputs to a tuple of outputs (one for each party). This random process is referred to as a functionality $f : \{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^* \times \cdots \times \{0,1\}^*$ where $f = (f_1, \ldots, f_n)$. Each party P_i has input x_i. For simplicity and without loss of generality, we work over functionalities for which $x_i \in \{0,1\}$. For every tuple of inputs $\boldsymbol{x} = (x_i, \ldots, x_n)$, the vector output by f is a random variable $(f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ ranging over tuples of strings where P_i receives $f_i(\boldsymbol{x})$. The following notation describes a functionality:

$$(x_1, \ldots, x_n) \mapsto (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$$

We prove the security of our MPC protocols in the presence of semi-honest and malicious adversaries in the dishonest majority setting.

In the semi-honest setting, an adversary can control a subset of the participating parties and always follows the protocol specification, but may try to learn additional information from the transcript of messages received and its internal state. Let $f = (f_1, \ldots, f_n)$ be a multiparty functionality and Π be an n-party protocol computing f in t rounds. The view of party P_i in an execution of Π on inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ is,

$$\mathbf{View}_{\Pi,i}(\kappa, \boldsymbol{x}) = (\kappa, x_i, r_i, \{m^j_{k\to i}\}_{j\in[t], k\neq i\in[n]})$$

where κ is the security parameter, r_i is the content of P_i's internal random tape, and each $m^j_{k\to i}$ represents the message received from party P_k in round j. The output of P_i in an execution of Π on $\boldsymbol{x}$ is denoted $\mathsf{output}_{\Pi,i}(\boldsymbol{x})$ and can be computed from $\mathbf{View}_{\Pi,i}(\kappa, \boldsymbol{x})$. The set of corrupted parties is denoted by $Z \subset \mathcal{P}$ and the set of honest parties by $\overline{Z}$. We extend the view notation to capture any subset of parties, denoting by $\mathbf{View}_{\Pi,T}(\kappa, \boldsymbol{x})$ the joint view of all parties in $T \subseteq \mathcal{P}$ on $(\kappa, \boldsymbol{x})$.

Definition A.3. *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{C}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f in the presence of a* ***PPT semi-honest adversary*** *tolerating adversary structure $\mathcal{Z}_{\mathsf{C}}$ if there exists a PPT simulator* Sim *such that for every $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathsf{Sim}(1^{\kappa}, Z, \{x_i, f_i(\boldsymbol{x})\}_{i\in Z}), \{f_i(\boldsymbol{x})\}_{i\notin Z}\right\}_{\kappa\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}$$
$$\stackrel{\mathrm{c}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(\kappa,\boldsymbol{x}), \mathsf{output}_{\Pi,\overline{Z}}(\kappa,\boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}$$

where κ is the computational security parameter. The distribution is considered over the randomness of the simulator and random tapes of parties in the protocol.

Definition A.4. *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{S}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f in the presence of an* ***unbounded semi-honest adversary*** *tolerating adversary structure $\mathcal{Z}_{\mathsf{S}}$ if there exists a PPT simulator* Sim *such that for every $Z \in \mathcal{Z}_{\mathsf{S}}$,*

$$\left\{\mathsf{Sim}(1^{s}, Z, \{x_i, f_i(\boldsymbol{x})\}_{i\in Z}), \{f_i(\boldsymbol{x})\}_{i\notin Z}\right\}_{s\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}$$
$$\stackrel{\mathrm{s}}{\equiv}\left\{\mathbf{View}_{\Pi,Z}(s,\boldsymbol{x}), \mathsf{output}_{\Pi,\overline{Z}}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the simulator's randomness and the parties' random tapes in the protocol.

In the malicious setting, an adversary controlling a subset of the participating parties can have them deviate arbitrarily from the protocol specification. Security in this setting is defined by comparing an execution of the functionality in an ideal model and an execution of the protocol in the real model. Depending on the security guarantee required, the ideal model execution is defined differently for the case of security with abort and fairness.

Execution in the ideal model (Security with Abort). An ideal execution is an interaction between all the parties in $\mathcal{P}$ and a trusted party $\mathcal{F}$, wherein the parties submit their input to the trusted party that computes the output and returns it. An honest party receives input for the computation and forwards it to the trusted party. In contrast, a corrupt party can replace its input with an arbitrary value of the same length. On receiving these, $\mathcal{F}$ computes the output and first sends the outputs of the corrupt parties to the adversary. The adversary then decides whether the honest parties will receive their outputs from $\mathcal{F}$ or an abort symbol $\perp$. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \boldsymbol{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}^{\perp}_{f,\mathcal{A}(z),Z}(\kappa,\boldsymbol{x})$.

Execution in the ideal model (Fairness). This is a similar ideal execution as described above. Here again, honest parties receive inputs for the computation and forward them to the trusted party. A corrupt party can replace its input with an arbitrary value of the same length. If these inputs contain an abort symbol $\perp$, $\mathcal{F}$ does not compute the output, and none of the parties receive it. Instead, $\mathcal{F}$ sends $\perp$ to the honest parties. Otherwise, $\mathcal{F}$ computes the output and simultaneously sends it to *both the adversary and the honest parties*. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \boldsymbol{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}^{\mathsf{fair}}_{f,\mathcal{A}(z),Z}(\kappa, \boldsymbol{x})$.

Execution in the real model. This is an interaction directly among the parties in $\mathcal{P}$ wherein the honest parties follow the instructions in the protocol Π. The adversary $\mathcal{A}$ sends all messages in place of the corrupted parties and may follow any arbitrary strategy. Letting Π be a protocol that computes f as above, $\mathcal{A}$ be a non-uniform probabilistic machine, and let $Z \subset \mathcal{P}$ be the set of corrupted parties, the real execution of Π on inputs $(\kappa, \boldsymbol{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \boldsymbol{x})$.

Definition A.5. *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{C}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f with abort (resp. fairness) in the presence of **PPT malicious adversaries** tolerating adversary structure $\mathcal{Z}_{\mathsf{C}}$ if there exists a PPT adversary Sim in the ideal model such that for every PPT $\mathcal{A}$ in the real model and $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathbf{IDEAL}^{\perp}_{f,\mathsf{Sim}(z),Z}(\kappa, \boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

resp.,

$$\left\{\mathbf{IDEAL}^{\mathsf{fair}}_{f,\mathsf{Sim}(z),Z}(\kappa, \boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

where κ is the computational security parameter. The distribution is considered over the adversaries' randomness and all the parties' random tapes.

Definition A.6. *Let f and Π be as above, and let $\mathcal{Z}_{\mathsf{S}} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f with abort (resp. fairness) in the presence of **unbounded malicious adversaries** tolerating adversary structure $\mathcal{Z}_{\mathsf{S}}$ if there exists an adversary Sim in the ideal model such that for every computationally unbounded $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_{\mathsf{S}}$,*

$$\left\{\mathbf{IDEAL}^{\perp}_{f,\mathsf{Sim}(z),Z}(s, \boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \overset{\mathrm{s}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s, \boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

resp.,

$$\left\{\mathbf{IDEAL}^{\mathsf{fair}}_{f,\mathsf{Sim}(z),Z}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \stackrel{\mathrm{s}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the adversaries' randomness and all the parties' random tapes.

In this work, the final protocols that we consider satisfy *threshold* adversary structures. This is a special case of adversary structures $\mathcal{Z}$ that can be represented by a threshold $t \leq n$ and refers to $\mathcal{Z}$ containing any subset of n of size at most t.

MPC with fall-back security. Informally, a multi-party protocol satisfies fall-back security if it is secure when certain subsets of parties are corrupted by a computationally unbounded adversary. However, if more parties than that are corrupted, computational security is still guaranteed.

Definition A.7 (Semi-Honest Fall-Back Security)**.** *For a set of parties $\mathcal{P} = \{P_i\}_{i\in[n]}$ where P_i has input x_i, let $\Pi = \langle\mathcal{P}\rangle$ be an n-party protocol computing a function f with input $\boldsymbol{x} = \{x_i\}_{i\in[n]}$. Let $\mathcal{Z}_{\mathsf{C}} = 2^{\mathcal{P}}$ and $\mathcal{Z}_{\mathsf{S}} \subseteq \mathcal{Z}_{\mathsf{C}}$ be adversary structures for PPT and unbounded semi-honest adversaries, respectively. The protocol Π satisfies* ***fall-back security in the semi-honest setting tolerating*** *$\mathcal{Z}_{\mathsf{S}}$ if there exists a PPT Sim such that for every $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\begin{aligned}&\left\{\mathsf{Sim}(1^\kappa, Z, \{x_i, f_i(\boldsymbol{x})\}_{i\in Z}), \{f_i(\boldsymbol{x})\}_{i\notin Z}\right\}_{\kappa\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}\\ \stackrel{\mathrm{c}}{\equiv}&\left\{\mathbf{View}_{\Pi,Z}(\kappa,\boldsymbol{x}), \mathsf{output}_{\Pi,\overline{Z}}(\kappa,\boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}\end{aligned}$$

where κ is the computational security parameter; and there exists a PPT simulator Sim such that for every $Z \in \mathcal{Z}_{\mathsf{S}}$,

$$\begin{aligned}&\left\{\mathsf{Sim}(1^s, Z, \{x_i, f_i(\boldsymbol{x})\}_{i\in Z}), \{f_i(\boldsymbol{x})\}_{i\notin Z}\right\}_{s\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}\\ \stackrel{\mathrm{s}}{\equiv}&\left\{\mathbf{View}_{\Pi,Z}(s,\boldsymbol{x}), \mathsf{output}_{\Pi,\overline{Z}}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x}\in\{0,1\}^*}\end{aligned}$$

where s is the statistical security parameter. The distribution is considered over the randomness of the simulator and all random tapes of parties in the protocol.

Definition A.8 (Malicious Fall-Back Security)**.** *For a set of parties $\mathcal{P} = \{P_i\}_{i\in[n]}$ where P_i has input x_i, let $\Pi = \langle\mathcal{P}\rangle$ be an n-party protocol computing a function f with input $\boldsymbol{x} = \{x_i\}_{i\in[n]}$. Let $\mathcal{Z}_{\mathsf{C}} = 2^{\mathcal{P}}$ and $\mathcal{Z}_{\mathsf{S}} \subseteq \mathcal{Z}_{\mathsf{C}}$ be adversary structures for PPT and computationally unbounded malicious adversaries, respectively. The protocol Π satisfies* ***fall-back security in the malicious setting tolerating***

$\mathcal{Z}_{\mathsf{S}}$ if there exists a PPT adversary Sim *in the ideal model such that for every PPT $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_{\mathsf{C}}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(\kappa,\boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa,\boldsymbol{x})\right\}_{\kappa\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

where κ is the computational security parameter; and there exists an adversary Sim *in the ideal model such that for every computationally unbounded $\mathcal{A}$ in the real model and every $Z \in \mathcal{Z}_{\mathsf{S}}$,*

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim}(z),Z}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*} \overset{\mathrm{s}}{\equiv} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s,\boldsymbol{x})\right\}_{s\in\mathbb{N},\boldsymbol{x},z\in\{0,1\}^*}$$

where s is the statistical security parameter. The distribution is considered over the randomness of the simulator and all random tapes of parties in the protocol.

Secret-Sharing. A *secret-sharing scheme* (SS) is a cryptographic primitive used to store sensitive information in a distributed manner. It is defined as a pair of probabilistic polynomial-time (PPT) algorithms $\mathsf{SS} = (\mathsf{Share}, \mathsf{Recon})$, over a set of n parties P. Given a secret message m, the sharing algorithm $\mathsf{Share}(m)$ outputs a vector of n shares $(s_1, \ldots, s_n)$, where each share s_i is distributed to party $i \in P$. The scheme is defined with respect to an adversary structure $\mathcal{Z} \subseteq 2^P$, which specifies the subsets of parties that may be corrupted. The scheme satisfies the following properties:

- **Privacy:** For any subset $Z \in \mathcal{Z}$, the joint distribution of the shares $\{s_i\}_{i\in Z}$ reveals no information about the secret m, even against a computationally unbounded adversary.
- **Reconstruction:** For any subset $S \subseteq P$ such that $S \notin \mathcal{Z}$, the parties in S can jointly reconstruct the secret m using the algorithm Recon.

This is formalized in Definition A.9 below, for threshold adversary structures.

Definition A.9 (Threshold Secret-Sharing)**.** *For parameters $n, t \in \mathbb{N}$, where $t \leq n$, and message space $\mathcal{M}$, a tuple of PPT algorithms* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Recon})$ *is a* ***secret-sharing scheme*** *if the following holds:*

- *for any $m \in \mathcal{M}$, $[m] = \{[m]_i^t\}_{i\in[n]} \leftarrow \mathsf{Share}(m)$ denotes a t-out-of-n secret-sharing of some message m and for any $\mathcal{H} \subseteq [n]$ we use $[m]_{\mathcal{H}} = \{[m]_i^t\}_{i\in\mathcal{H}}$.*
- ***t-reconstruction:*** *for all $m \in \mathcal{M}$ and all $R \subseteq [n]$ with $|R| \geq t$ it always holds that,*

$$\mathsf{Recon}(\mathsf{Share}(m)_R) = m$$

- ***t-privacy:*** *for all $m^0, m^1 \in \mathcal{M}$ and all $\mathcal{C} \subseteq [n]$ with $|\mathcal{C}| < t$ it holds that,*

$$\mathsf{Share}(m^0)_{\mathcal{C}} \equiv \mathsf{Share}(m^1)_{\mathcal{C}}$$

Information-Theoretic instantiations of threshold secret-sharing schemes exist [Sha79] with linear sharing and reconstruction algorithms.

Message Authentication Codes (MACs). Message Authentication Code (MAC) is a symmetric cryptographic primitive used to ensure the integrity of a communicated message. A MAC scheme consists of three algorithms ($\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Verify}$). There is a shared secret key k between the signer and verifier. The sender of message m signs it by generating a tag $\sigma \leftarrow \mathsf{Tag}(k, m)$. The message is sent along with the tag. The receiver verifies the message and tag pair $\mathsf{Verify}(k, m, \sigma) = 1$.

Definition A.10 (Message Authentication Codes (MACs))**.** *For a message space $\mathcal{M}$ and key space $\mathcal{K}$, a MAC scheme is a tuple of PPT algorithms* ($\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Verify}$)*:*

- $k \leftarrow \mathsf{KeyGen}(1^\kappa)$ *is the key generation algorithm such that $k \in \mathcal{K}$ is sampled uniformly at random.*
- $\sigma \leftarrow \mathsf{Tag}(k, m)$ *is the tag generation algorithm that on input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ outputs a tag σ.*
- $\mathsf{Verify}(k, m, \sigma) \in \{0, 1\}$ *is the verification algorithm that on input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag σ, outputs 1 (accept) or 0 (reject).*
  The MAC scheme needs to satisfy the following properties:
- ***Correctness:*** *For all $k \in \mathcal{K}$ and $m \in \mathcal{M}$ it holds that,*

$$\mathsf{Verify}(k, m, \mathsf{Tag}(k, m)) = 1$$

- ***Unforgeability:*** *For every non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function μ such that, for all $k \in \mathcal{K}$, and oracle $\mathcal{O}(\cdot)$ such that $\mathcal{O}_k(m) = \mathsf{Tag}(k, m)$, it must hold that,*

$$\Pr\left[\mathsf{Verify}(k, m^*, \sigma^*) = 1 \cap m^* \notin \tau \middle| (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\kappa)\right] \leq \mu(\kappa)$$

  where τ is the list of all queries made by $\mathcal{A}$ to $\mathcal{O}$. If the above holds for any unbounded adversary $\mathcal{A}$, then the MAC scheme is called an information-theoretic MAC.

In our protocols, we use another variant of a MAC scheme called a one-time multi-key IT-MAC introduced in [ACGJ19]. This is formalized in the definition below.

Definition A.11 (One-Time Multi-Key IT-MAC [ACGJ19])**.** *A one-time multi-key IT-MAC scheme with message space $\mathcal{M}$, key space $\mathcal{K}$ and signature space $\mathcal{X}$ for a set of n parties $P = \{P_i\}_{i\in[n]}$ is a tuple of PPT algorithms* $\mathsf{MK} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $\mathsf{K} \leftarrow \mathsf{KeyGen}(1^s)$*: The key generation algorithm that takes the statistical security parameter s, and samples $\mathsf{K} \in \mathcal{K}$ uniformly at random.*
- $\tau \leftarrow \mathsf{MK.Sign}(\mathsf{K}_1, \ldots, \mathsf{K}_n, x)$*: The signing algorithm takes a set of keys $\mathsf{K}_1, \ldots, \mathsf{K}_n \in \mathcal{K}^n$, and a message bit $x \in \mathcal{M}$ as input and outputs a MAC $\tau \in \mathcal{X}$.*
- $b \leftarrow \mathsf{MK.Verify}(\mathsf{K}, x, \tau)$*: The verification algorithm takes a key $\mathsf{K} \in \mathcal{K}$, a message bit $x \in \mathcal{M}$ and a MAC value $\tau \in \mathcal{X}$ as input and outputs a bit $b \in \{0, 1\}$.*

We say that a one-time multi-key MAC scheme is $\varepsilon(s)$-statistically secure if it satisfies the following properties:

- ***Correctness:*** *For all $m \in \mathcal{M}$,*

$$\Pr\left[\tau \leftarrow \mathsf{Sign}(\{\mathsf{K}_i \leftarrow \mathsf{KeyGen}(1^s)\}_{i\in[n]}, m), \{\mathsf{Verify}(\mathsf{K}_i, m, \tau) = 1\}_{i\in[n]}\right] = 1$$

- ***Multi-Key Unforgeability:*** *For any unbounded adversary $\mathcal{A}$, there exists a negligible function μ such that the following holds:*

$$\Pr\left[\mathsf{Expt}^{\mathsf{forge}}_{\mathcal{A}}(1^s, 1^n) = 1\right] \leq \mu(s),$$

  where $\mathsf{Expt}^{\mathsf{forge}}_{\mathcal{A}}(1^s, 1^n)$ is defined as follows:
  - *The adversary $\mathcal{A}$ chooses a subset $S \subset [n]$, a set of keys $\{\mathsf{K}_i\}_{i\in[n]\setminus S}$ and a message $x \in \mathcal{M}$, and sends these to the challenger.*
  - *The challenger samples keys $\{\mathsf{K}_i \leftarrow \mathsf{MK.KeyGen}(1^s)\}_{i\in S}$ and computes $\tau \leftarrow \mathsf{MK.Sign}(\{\mathsf{K}_i\}_{i\in[n]}, x)$ and sends τ to $\mathcal{A}$.*
  - *The adversary $\mathcal{A}$ returns (x^*, τ^*).*
  - *The experiment outputs 1 if $\tau^* \neq \tau$ and $\mathsf{MK.Verify}(\mathsf{K}_i, x^*, \tau^*) = 1$ for some $i \in S$, and 0 otherwise.*

[ACGJ19] provided an instantiation of a one-time multi-key IT-MAC, which takes the message m as a bit i.e. $\mathcal{M} = \{0,1\}$. However, we relax the notation and call the $\mathsf{MK.Sign}$ algorithm on any string $x \in \{0,1\}^*$. Concretely, the tag may be viewed as a collection of tags on each bit of x, but for readability, we suppress this lower-level representation.

Commitment Scheme. A commitment scheme Com is a two-party protocol between a committer $\mathcal{C}$ with input x and a receiver $\mathcal{R}$, in which $\mathcal{C}$ samples randomness r and creates a *commitment* to the value x. This commitment should be *hiding*: no information about x should be revealed. At a later point in time, $\mathcal{C}$ can send a *decommitment* message to $\mathcal{R}$ showing that the value committed to was indeed x. The commitment scheme should also satisfy the binding property, which states $\mathcal{C}$ cannot deceive $\mathcal{R}$ into believing that the committed value was some different $x' \neq x$. The binding and hiding properties in the commitment schemes contradict each other, so a commitment cannot be both statistically binding and hiding. Commitment schemes can be designed to satisfy either the statistical binding or statistical hiding property, and we use both in our protocol design.

Definition A.12 (Statistically Binding Commitment)**.** *Let $\mathsf{Com}_{\mathsf{SB}}(m; r)$ be a commitment scheme for messages $m \in \{0,1\}^\ell$. It satisfies:*

- ***Statistical Binding:*** *For all messages $m_0, m_1 \in \{0,1\}^\ell$, it holds that*

$$\{\mathsf{Com}_{\mathsf{SB}}(m_0; r)\}_{r\in\{0,1\}^*} \cap \{\mathsf{Com}_{\mathsf{SB}}(m_1; s)\}_{s\in\{0,1\}^*} = \emptyset$$

– ***Computationally Hiding:*** *For all choices of* $m_0, m_1 \in \{0,1\}^\ell$*, it holds that*

$$\{\mathsf{Com}_{\mathsf{SB}}(m_0; r)\}_{r \in \{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{Com}_{\mathsf{SB}}(m_1; s)\}_{s \in \{0,1\}^*}$$

– ***Decommitment:*** *A decommitment to a commitment* τ *is a pair* (m, r) *such that* $\tau = \mathsf{Com}_{\mathsf{SB}}(m; r)$*.*

Definition A.13 (Statistical Hiding Commitment)**.** *A* statistically hiding commitment scheme *for a message space* $\{0,1\}^\ell$ *is a PPT algorithm* $\mathsf{Com}_{\mathsf{SH}}$ *satisfying:*

– ***Computationally Binding:*** *For every PPT adversary* $\mathcal{A}$*, there exists a negligible function* μ *such that*

$$\Pr\Big[\mathcal{A}(1^\kappa) \to (c, m_0, r_0, m_1, r_1) \ : \ \mathsf{Com}_{\mathsf{SH}}(m_0; r_0) = c \wedge \mathsf{Com}_{\mathsf{SH}}(m_1; r_1) = c \ \wedge \ m_0 \neq m_1\Big] \leq \mu(\kappa),$$

where $m_0, m_1 \in \{0,1\}^\ell$*.*

– ***Statistically Hiding:*** *For all* $m_0, m_1 \in \{0,1\}^\ell$*, it holds that*

$$\{\mathsf{Com}_{\mathsf{SH}}(m_0; r)\}_{r \in \{0,1\}^*} \equiv \{\mathsf{Com}_{\mathsf{SH}}(m_1; s)\}_{s \in \{0,1\}^*}$$

– ***Decommitment:*** *A decommitment to a commitment* c *is a pair* (m, r) *such that* $c = \mathsf{Com}_{\mathsf{SH}}(m; r)$*.*

Zero-Knowledge Proofs. Zero-Knowledge Proofs [CF01] are an interaction between a prover P and a verifier V such that the interaction convinces V that for a public value x, P has a secret witness w such that $(x, w) \in R$ for a public relation R but the interaction reveals nothing beyond that fact. We use a computational zero-knowledge proof protocol between an unbounded prover and a computationally bounded verifier.

Definition A.14 (Zero-Knowledge Proof)**.** *Let* μ *be a negligible function, and for computational security parameter* κ*, let* $\epsilon = \mu(\kappa)$*. The tuple* $\mathsf{ZKP} = (P, V)$ *is a zero-knowledge proof for a relation* R *if for every PPT* V*,*

– ***Perfect Completeness:*** *For every* $(x, w) \in R$*,*

$$\Pr[\langle P(w), V\rangle(x) = 1] \geq 1 - \epsilon$$

– ϵ***-Soundness:*** *For every* x *such that* $\nexists w$ *for which* $(x, w) \in R$*, and every malicious* P^**,*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \epsilon$$

– ***Computational Zero-Knowledge:*** *For every PPT* V^**, there is a PPT* S^**,*

$$\{\langle P(w), V^*\rangle(x)\}_{(x,w) \in R} \stackrel{c}{\equiv} \{S^*(x)\}_{(x,w) \in R}$$

Oblivious Transfer. A 1-out-of-2 OT is a two-party cryptographic protocol between a sender S with inputs $m_0, m_1 \in \{0,1\}^n$ and a receiver R with input b, where at the end of the protocol R receives m_b. This is formalized by the following two party functionality $\mathcal{F}_{\mathsf{OT}}$ between the sender and receiver:

$$\mathsf{OT} : ((m_0, m_1), b) \mapsto (\perp, m_b)$$

In our constructions, we use protocols Π_{OT} that securely realize this functionality $\mathcal{F}_{\mathsf{OT}}$ using 2-rounds. The protocol needs to tolerate either an unbounded adversary corrupting the sender or a PPT adversary corrupting the receiver.

Semi-Honest Secure OT. Such a 2-round protocol Π_{OT} as required exists in the plain model when security is required in the presence of semi-honest corruption [BM89], assuming the hardness of DDH. Going forward, we denote the algorithms in Π_{OT} as $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ where,

- R computes $(\mathsf{ots}_1, \omega) \leftarrow \mathsf{OT}_1(1^\kappa, b)$ and sends ots_1 to S.
- S computes $\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{ots}_1, (m_0, m_1))$ and sends ots_2 to R.
- R outputs $m_b \leftarrow \mathsf{OT}_3(\mathsf{ots}_2, (b, \omega))$.

Malicious Secure OT. When requiring security in the presence of malicious adversaries, such a 2-round Π_{OT} only exists assuming setup. In particular, constructions of maliciously secure 2-round OT providing statistical security against unbounded sender and computational security against PPT receiver are known in the CRS setting from [PVW08]. We use their construction in the malicious setting. Here, in addition to the semi-honest syntax, we denote a CRS generation algorithm by K_{OT} that on input 1^κ outputs a reference string σ. The protocols $\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3$ now additionally take σ as input.

Garbled Circuits. Bellare et al. [BHR12] abstract a garbling scheme as a cryptographic object containing four algorithms as given in Definition A.15. In the definition below, we let a Boolean circuit $\mathbf{C}$ represent the function $f : \{0,1\}^n \leftarrow \{0,1\}^m$.

Definition A.15 (Garbling Scheme [BHR12])**.** *A garbling scheme is a tuple* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *of four polynomial-time algorithms with the following syntax:*

- $(\mathbf{F}, \mathbf{e}, \mathbf{d}) \leftarrow \mathsf{Gb}(1^\kappa, \mathbf{C})$*: on input a security parameter (in unary) and a circuit* $\mathbf{C}$*, returns a garbling* $\mathbf{F}$*, input encoding function* $\mathbf{e}$*, and output decoding function* $\mathbf{d}$*.*
- $\mathbf{X} = \mathsf{En}(\mathbf{e}, \mathbf{x})$*: returns the encoding* $\mathbf{X}$ *for function input* $\mathbf{x}$*.*
- $\mathbf{Y} = \mathsf{Ev}(\mathbf{F}, \mathbf{X})$*: returns the output labels* $\mathbf{Y}$ *by evaluating* $\mathbf{F}$ *on* $\mathbf{X}$*.*
- $\{\perp, \mathbf{y}\} = \mathsf{De}(\mathbf{Y}, \mathbf{d})$*: returns either the failure symbol* $\perp$ *or a value* $\mathbf{y} = f(\mathbf{x})$*.*

These algorithms must satisfy the following properties:

– Correctness*: For every circuit* $\mathbf{C} : \{0,1\}^n \to \{0,1\}^m$ *and input* $\mathbf{x} \in \{0,1\}^n$,

$$\Pr[\mathbf{y} = \mathbf{C}(\mathbf{x}) : (\mathbf{F}, \mathbf{e}, \mathbf{d}) \leftarrow \mathsf{Gb}(\mathbf{C}),\ \mathbf{X} = \mathsf{En}(\mathbf{e}, \mathbf{x}),\ \mathbf{Y} = \mathsf{Ev}(\mathbf{F}, \mathbf{X}),\ \mathbf{y} = \mathsf{De}(\mathbf{d}, \mathbf{Y})] = 1$$

– Selective Privacy by Simulation*: there exists a PPT algorithm* Sim *such that for every two-stage PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *the following is negligible:*

$$\left| \Pr_{\substack{(\mathbf{C},\mathbf{x},\sigma)\leftarrow\mathcal{A}_0(1^\kappa) \\ (\mathbf{F},\mathbf{e},\mathbf{d})\leftarrow\mathsf{GS.Gb}(1^\kappa,\mathbf{C}) \\ \mathbf{X}=\mathsf{GS.En}(\mathbf{e},\mathbf{x})}} [\mathcal{A}_1(\sigma, (\mathbf{F}, \mathbf{X}, \mathbf{d})) = 1] - \Pr_{\substack{(\mathbf{C},\mathbf{x},\sigma)\leftarrow\mathcal{A}_0(1^\kappa) \\ (\mathbf{F},\mathbf{X},\mathbf{d})\leftarrow\mathsf{Sim}(1^\kappa,\mathbf{C},\mathbf{C}(\mathbf{x}))}} [\mathcal{A}_1(\sigma, (\mathbf{F}, \mathbf{X}, \mathbf{d})) = 1] \right|$$

Robust Additive Randomized Encodings. We re-state the definition of a robust additive randomized encoding (RARE) from [HIKR23] below.

Definition A.16 (Robust Additive Randomized Encodings [HIKR23])**.** *A RARE scheme for a function* $f : (\{0,1\}^*)^* \to \{0,1\}^*$ *is a triple of algorithms* $\Pi = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *with the following syntax:*

– $\mathsf{Setup}(1^\kappa, 1^n, 1^\ell) \to \mathsf{pp}$ *is a PPT setup algorithm that, given security parameter* κ*, number of parties* n*, and input length* ℓ*, generates public parameters* pp*. The public parameters include* κ, n, ℓ*, and an explicit description of an Abelian group* $\mathbb{G}$*.*
– $\mathsf{Encode}(\mathsf{pp}, i, x_i) \to \hat{x}_i$ *is a PPT encoding algorithm that maps an input* x_i *of party* i *to a group element from* $\mathbb{G}$*.*
– $\mathsf{Decode}(\mathsf{pp}, \hat{y})$ *is a PPT decoding algorithm that maps a group element* $\hat{y} \in \mathbb{G}$ *to an output* y*.*

We require the following properties:

– ***Correctness:*** *Π is correct, with a possible error of* $\epsilon = \epsilon(\kappa)$*, if for all* κ, n, ℓ *and* $x_1, \ldots, x_n \in \{0,1\}^\ell$*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, 1^n, 1^\ell); \\ \hat{x}_i \leftarrow \mathsf{Enc}(\mathsf{pp}, i, x_i); \\ \hat{y} = \sum_{i=1}^n \hat{x}_i \end{array} : \mathsf{Dec}(\mathsf{pp}, \hat{y}) = f(x_1, \ldots, x_n)\right] \geq 1 - \epsilon(\kappa),$$

Π is said to be statistically correct if ϵ is a negligible function.
– ***Statistical Robustness:*** *Π is said to be robust if there exists a simulator* Sim *with access to a residual function oracle, such that for some negligible function* $\delta(\cdot)$*, it holds for all* $\kappa, n, \ell, H \subset [n]$*, and inputs* $\mathbf{x} = (x_i : i \in H)$*, that*

$$\mathsf{SD}(\mathsf{Sim}^{f_{H,\mathbf{x}}}(1^\kappa, 1^n, H, 1^\ell), \Pi_H(1^\kappa, \mathbf{x})) \leq \delta(\kappa)$$

where $\mathsf{SD}(\cdot,\cdot)$ *is the statistical distance and* Π_H *is as follows:*

$$\Pi_H(1^\kappa, x_1, \ldots, x_n) := \left\{ \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, 1^n, 1^\ell); \hat{x}_i \leftarrow \mathsf{Enc}(\mathsf{pp}, i, x_i); \\ \hat{y}_H = \sum_{i \in H} \hat{x}_i;\ \textit{output}\ (\mathsf{pp}, \hat{y}) \end{array} \right\}$$

We use this as a building block for a 2-round semi-honest fall-back secure protocol in the plain model. Although instantiations for an RARE with statistical robustness are not yet known, this construction demonstrates the difficulty of proving a round complexity lower bound for 2 rounds.

B Additional Material for Semi-Honest Fall-back MPC

B.1 Designated-Party Protocol

In a designated-party protocol instance, only a single party, known as the designated party, receives the protocol's output. Such a protocol provides IT security when the designated party is honest, and computational security otherwise. Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be the complete set of parties participating in MPC. Without loss of generality, we denote P_1 as the designated party. The designated-party protocol provides fall-back security with respect to the adversary structure $\mathcal{Z}_{\mathsf{D}} = \{Z \subseteq \{P_i\}_{i\in[n]\setminus\{1\}}\}$, i.e., the adversary structure consisting of any subset of parties excluding the designated party P_1.

To reduce the overall round complexity of the semi-honest fall-back secure protocol, we reduce the round complexity of the designated-party protocol used to implement the next-message functions. Towards this, we use the ideas from [GS18] to design a 2-round designated-party protocol, denoted as $\Pi_{\mathsf{sh\text{-}des}}$ hereafter. The key idea is a technique that enables garbled circuits to talk to each other. This enables parties to avoid multiple rounds of interaction by sending a garbled circuit that interacts with the other parties on their behalf by the end of round 2. To design a designated-party protocol, we modify the 2-round protocol so that only the designated party receives garbled circuits from all the parties (as opposed to all parties receiving all garbled circuits as in the original construction). The designated party can then locally evaluate all these circuits to obtain the output. Furthermore, since the construction provides only computational security, additional modifications are needed to ensure statistical security when the designated party is honest. We first recall the details of the original 2-round compiler and then describe these modifications in detail.

The compiler works for protocols satisfying a certain syntactic structure, and such protocols are referred to as conforming protocols. It is shown that any MPC protocol can be transformed into a conforming protocol while preserving the correctness and security properties of the original protocol. A conforming protocol Φ is defined by functions pre, post, and computations steps denoted as $\Phi_1, \Phi_2, \cdots \Phi_T$. These conforming protocols proceed as follows:

- Pre-processing phase: For each $i \in [n]$, party P_i computes

$$(z_i, v_i) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$$

  where pre is a randomized algorithm. The algorithm pre takes as input the index i of the party and its input x_i, and outputs $z_i \in \{0,1\}^{\ell/n}$ and $v_i \in \{0,1\}^\ell$ (where ℓ is a parameter of the protocol). P_i retains v_i as its secret

information and broadcasts z_i to every other party. In practice, z_i acts as one-time-pad encryption of this party's input x_i. We require that

$$v_{i,k} = 0 \quad \text{for all } k \in [\ell] \setminus \left\{(i-1)\frac{\ell}{n} + 1, \ldots, i\frac{\ell}{n}\right\}$$

– Computation phase: For each $i \in [n]$, party P_i sets

$$\mathsf{st}_i := (z_1 \,\|\, \cdots \,\|\, z_n) \oplus v_i$$

Next, for each $t \in \{1, \ldots, T\}$, the parties proceed as follows:

1. Parse action Φ_t as (i, f, g, h) where $i \in [n]$ and $f, g, h \in [\ell]$.
2. Party P_i computes one NAND gate:

$$\mathsf{st}_{i,h} = \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g}),$$

   and broadcasts $\mathsf{st}_{i,h} \oplus v_{i,h}$ to every other party.
3. Every party P_j for $j \neq i$ updates $\mathsf{st}_{j,h}$ to the bit value received from P_i.

We require that for all $t, t' \in [T]$ such that $t \neq t'$, if $\Phi_t = (\cdot, \cdot, \cdot, h)$ and $\Phi_{t'} = (\cdot, \cdot, \cdot, h')$, then $h \neq h'$. Let $A_i \subseteq [T]$ denote the set of rounds in which party P_i sends a bit:

$$A_i := \{t \in [T] \mid \Phi_t = (i, \cdot, \cdot, \cdot)\}.$$

– Output phase: For each $i \in [n]$, party P_i outputs $\mathsf{post}(\mathsf{st}_i)$

Such a conforming protocol can be compiled to a 2-round protocol as follows. In the first round of the compiled protocol, every party performs the pre-processing stage of Φ and broadcasts its computed messages. In the second round, each party sends a collection of garbled circuits that non-interactively realize the computation phase of Φ. The parties can locally reconstruct the complete transcript of the computation phase of Φ. Using this information, every party can determine the final output of the protocol. To enable only a single designated party, say P_1, to obtain the output, we modify this protocol by having all parties send their garbled circuits only to P_1. Party P_1 then locally evaluates these circuits to obtain the output.

As stated earlier, this compiler only provides computational privacy and does not give us the desired information-theoretic security when the designated party is honest. To satisfy this security guarantee required by the designated-party protocol, we replace the generic OT scheme in their compiler with an OT scheme that is statistically secure against an unbounded sender. Upon receiving the circuits for each round of the underlying conforming protocol from all the parties, P_1 evaluates the circuits locally and obtains its output. Note that P_1 also generates garbled circuits as in the original compiler, but these are used only for its local computation and not sent to any other party. When the designated party is honest, the adversary only sees OT_1 messages corresponding to the honest parties' inputs in the protocol. By the security property of the OT scheme, these

inputs remain statistically hidden, and thus, the protocol achieves statistical security in this case. The correctness of the computation follows directly from the correctness of the [GS18] compiler. This compiler for obtaining a 2-round designated-party protocol is formalized in Protocol B.1.1.

Protocol B.1.1: 2-Round Designated-Party Protocol $\Pi_{\mathsf{sh\text{-}des}}$

Setting: Let $P = \{P_i\}_{i\in[n]}$ be the set of parties. Let Π' be the n-party conforming semi-honest MPC protocol computing the same function as that of $\Pi_{\mathsf{sh\text{-}des}}$ with P_1 as the designated party. Π' is assumed to provide computational security against a dishonest majority.

Building Blocks:

- Garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ (Definition A.15).
- Semi-honest 2-round statistically secure against unbounded sender OT $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

Input: Each party P_i holds input (x_i, r_i).

The Protocol:

- **Round 1:** Each party P_i does:
  1. Compute $(z_i, v_i) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$.
  2. For each t where $\Phi_t = (i, f, g, h)$ (A_i is the set of such values of t) compute:
  $$\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\kappa, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta})$$
  for each $\alpha, \beta \in \{0,1\}$.
  3. Send $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i, \alpha,\beta\in\{0,1\}})$ to all parties.
- **Round 2:** Each party P_i does:
  1. Compute $\mathsf{st}_i := (z_1 \| \ldots \| z_n) \oplus v_i$.
  2. Initialize labels $\overline{\mathsf{L}}^{i,T+1} := \{\mathsf{L}_{k,0}^{i,T+1}, \mathsf{L}_{k,1}^{i,T+1}\}_{k\in[\ell]}$, where for each $k \in [\ell], b \in \{0,1\}$, $\mathsf{L}_{k,b}^{i,T+1} := 0^\kappa$.
  3. For each t from T down to 1:
     (a) Parse $\Phi_t = (i^*, f, g, h)$ and let P be as described in Functionality B.1.2.
     (b) If $i = i^*$, compute
     $$(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{L}}^{i,t}) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{P}[i, \Phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{L}}^{i,t+1}])$$
     (c) If $i \neq i^*$, for all $\alpha, \beta \in \{0,1\}$, set
     $$\mathsf{ots}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha,\beta}, \mathsf{L}_{h,0}^{i,t+1}, \mathsf{L}_{h,1}^{i,t+1})$$
     and compute
     $$(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{L}}^{i,t}) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{P}[i, \Phi_t, v_i, \bot, \{\mathsf{ots}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{L}}^{i,t+1}])$$
  4. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,1}\}_{k\in[\ell]}\right)$ to P_1.

– **Evaluation:** Party P_1 does:
  1. Collect labels $\{\mathsf{L}_k^{j,1}\}_{k\in[\ell]}$ from all P_j.
  2. For $t = 1$ to T:
     (a) Parse $\Phi_t = (i^*, f, g, h)$.
     (b) Compute $((\alpha,\beta,\gamma), \omega, \{\mathsf{L}_k^{i^*,t+1}\}_{k\in[\ell]}) \leftarrow \mathsf{Ev}(\widetilde{\mathsf{P}}^{i^*,t}, \{\mathsf{L}_k^{i^*,t}\}_{k\in[\ell]})$
     (c) Set $\mathsf{st}_{1,h} := \gamma \oplus v_{1,h}$.
     (d) For each $j \neq i^*$:
        i. Compute $(\mathsf{ots}_2, \{\mathsf{L}_k^{j,t+1}\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Ev}(\widetilde{\mathsf{P}}^{j,t}, \{\mathsf{L}_k^{j,t}\}_{k\in[\ell]})$.
        ii. Recover $\mathsf{L}_h^{j,t+1}$ using $\mathsf{L}_h^{j,t+1} := \mathsf{OT}_3(\mathsf{ots}_2, \omega)$
  3. Compute output $y_1 \leftarrow \mathsf{post}(1, \mathsf{st}_1)$.

Functionality B.1.2: Function P based on the Conforming Protocol

Input: st_i.
Hardcoded. The index i of the party, the action $\Phi_t = (i^*, f, g, h)$, the secret value v_i, the strings $\{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}$,$\{\mathsf{ots}_{2,t,\alpha,\beta}\}_{\alpha,\beta}$ and a set of labels $\overline{\mathsf{L}} = \{\mathsf{L}_{k,0}, \mathsf{L}_{k,1}\}_{k\in[\ell]}$.
1. If $i = i^*$ then:

(a) Compute $\mathsf{st}_{i,h} := \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$, $\alpha := \mathsf{st}_{i,f} \oplus v_{i,f}$, $\beta := \mathsf{st}_{i,g} \oplus v_{i,g}$, and $\gamma := \mathsf{st}_{i,h} \oplus v_{i,h}$.
(b) Output $((\alpha,\beta,\gamma), \omega_{t,\alpha,\beta}, \{\mathsf{L}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]})$.

2. else:

(a) Output $(\mathsf{ots}_{2,t,\mathsf{st}_{i,f},\mathsf{st}_{i,g}}, \{\mathsf{L}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]\setminus\{h\}})$.

We denote the protocol $\Pi_{\mathsf{sh\text{-}des}}$ applied to compute functionality f as $\Pi^f_{\mathsf{sh\text{-}des}}$. If party P_i executing it is the designated party, this is represented by $\Pi^{f_i}_{\mathsf{sh\text{-}des}}$.

Theorem B.1.3. *Let $\mathcal{Z}_\mathsf{D}$ be the set of all subsets of $\{P_i\}_{i\in[n]\setminus\{1\}}$. If $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ is a semi-honest 2-round OT protocol that is statistically secure against a corrupt sender and $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is a garbling scheme for circuits, then $\Pi_{\mathsf{sh\text{-}des}}$ is fall-back secure, tolerating $\mathcal{Z}_\mathsf{D}$ for semi-honest adversaries in the plain model.*

We provide an outline of the proof here. The formal proof for Theorem B.1.3 can be found in Appendix D.1.
Proof Outline. We provide an informal argument for the fall-back security of the protocol $\Pi_{\mathsf{sh\text{-}des}}$, obtained by compiling an underlying computationally secure conforming protocol using the modified compiler. We begin with the case of an unbounded adversary corrupting a subset of parties in $\{P_i\}_{i\in[n]\setminus 1}$. Observe that the only messages such an adversary receives from an honest party P_j are the OT_1 messages corresponding to its inputs and the values z_j, which are computed during the pre step of Φ and communicated in the first round. The z_j values are one-time pad encryptions of the corresponding inputs x_j and therefore reveal no information on their own. Furthermore, since we use an OT scheme that provides

statistical security against a corrupt sender, the adversary obtains no information about the inputs through the OT instances.

We next consider a PPT adversary corrupting an arbitrary subset of parties. Security in this setting follows directly from the security of the original compiler [GS18] and the underlying conforming protocol. Since security against adversaries that do not corrupt the designated party already follows from the statistical case, it suffices to consider the case where the designated party is corrupted. In this case, for an honest party P_j, the adversary receives the OT_1 messages corresponding to P_j's input, the value z_j, and the garbled circuits that P_j sends to P_1. This is exactly the information received by an adversary during an execution of the 2-round compiled protocol of [GS18]. Hence, security follows directly from the security of their compiler and the underlying protocol.

B.2 Possibility of 2-Round Fall-back Protocol

We describe one possible approach for obtaining a 2-round semi-honest fall-back secure protocol in the plain model based on robust additive randomized encodings (RARE). We design a 2-round protocol for function $f(x_1, \cdots, x_n)$ that relies on statistical RARE, a hypothetical variant of RARE. On a high level, each party P_i encodes its input x_i and masks this encoding using an additive share of 0, and broadcasts this masked encoding. To prevent residual attacks[9] with respect to the corrupt parties' inputs, parties commit to their inputs and share these commitments. For any function f, we define the RARE encoding for an augmented functionality g, which additionally takes these commitment shares as inputs, and checks if the commitments are valid before giving the output $y = f(x_1, \ldots, x_n)$. Below, we describe the functionality g and the 2-round protocol.

Functionality B.2.1: RARE function g based on Functionality f

---

Inputs: Each party P_i has:

- Inputs x_i, r_i, r'_i.
- Shares $\{[c_j]_i^t\}_{j\in[n]}$ where $c_j \leftarrow \mathsf{Com}_{\mathsf{SB}}(x_j; r_j)$.
- $\{c'_j\}_{j\in[n]}$ where $c'_j \leftarrow \mathsf{Com}_{\mathsf{SH}}(x_j; r'_j)$.

Computation:

1. For all $j \in [n]$, reconstruct $c_j = \mathsf{SS.Recon}([c_j]_1^t, \ldots, [c_j]_n^t)$ and check if $c_j = \mathsf{Com}_{\mathsf{SB}}(x_j; r_j)$. If not, set output $y = \perp$.
2. Check if all parties input the same $\{c'_j\}_{j\in[n]}$. For all $j \in [n]$, check if $c'_j = \mathsf{Com}_{\mathsf{SH}}(x_j; r'_j)$. If either condition fails, set output $y = \perp$.
3. If both the above checks pass, compute $y = f(x_1, \ldots x_n)$.

---

[9] Residual attacks refer to the case where a semi-honestly corrupt party can attempt to derive additional information from a protocol by locally computing on messages received from other parties, with more than one version of its internal state.

Protocol B.2.2: 2-Round Fall-Back Secure $\Pi_{\mathsf{sh\text{-}fb\text{-}2r}}$ for $t < \frac{n}{2}$

Setting: Let $P = \{P_i\}_{i\in[n]}$ be set of parties. The parties wish to compute a function $f(x_1, \ldots, x_n)$.
Building Blocks:

- A statistically binding commitment scheme $\mathsf{Com}_{\mathsf{SB}}$.
- A statistically hiding commitment scheme $\mathsf{Com}_{\mathsf{SH}}$.
- A statistical RARE $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ for g (Functionality B.2.1).

Inputs: Party P_i has input x_i.
The Protocol:

- **Round 1:**
  Each party P_i does the following:
  1. Compute $c_i = \mathsf{Com}_{\mathsf{SB}}(x_i; r_i)$ and generate t-out-of-n Shamir sharing of c_i as $\{[c_i]^t_j\}_{j\in[n]} \leftarrow \mathsf{SS.Share}(c_i)$.
  2. Compute and broadcast $c'_i = \mathsf{Com}_{\mathsf{SH}}(x_i; r'_i)$.
  3. Generate and send additive secret shares of 0 as $\{[0]^n_{i,j}\}_{j\in[n]}$.
  4. Receive $([c_j]^t_i, c'_j, [0]^n_{j,i})$ from P_j for all $j \in [n]$.
- **Round 2:**
  Each party P_i does the following:
  1. Compute $\hat{X}_i \leftarrow \mathsf{Encode}(\mathsf{pp}, X_i)$, where $X_i = (x_i, r_i, r'_i, \{[c_j]^t_i, c'_j\}_{j\in[n]})$.
  2. Compute $[0]^n_i = \sum_{j=1}^n [0]^n_{j,i}$ and broadcast $\hat{X}_i + [0]^n_i$.
  3. Receive $\hat{X}_j + [0]^n_j$ from P_j for all $j \in [n]$.
  4. Obtain the output as $y \leftarrow \mathsf{Decode}(\mathsf{pp}, \hat{Y})$, where $\hat{Y} = \sum_{j=1}^n \hat{X}_j + [0]^n_j = \sum_{j=1}^n \hat{X}_j$

Theorem B.2.3. *Let* $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *be a statistical RARE for* g*, and* $\mathsf{Com}_{\mathsf{SB}}$ *and* $\mathsf{Com}_{\mathsf{SH}}$ *are statistically binding and statistically hiding commitment schemes respectively, then* $\Pi_{\mathsf{sh\text{-}fb\text{-}2r}}$ *is a fall-back secure protocol tolerating* $t < \frac{n}{2}$ *corruptions against semi-honest adversaries in the plain model.*

We outline the proof below. The formal proof can be found in Appendix D.3.
Proof Outline. The correctness of the protocol follows from the correctness of the RARE scheme. Denote the set of corrupt parties as Z. Since an honest party P_j broadcasts $\hat{X}_j + [0]^n_j$, the adversary $\mathcal{A}$ can only recover $\sum_{P_j\notin Z} \hat{X}_j$, and the individual $\hat{X}_j$s remain hidden. We now analyze possible residual attacks, where a party P_i locally changes $\hat{X}_j$ to some $\hat{X}'_j$ corresponding to input $X'_j \neq X_j$ and tries to learn something about the honest parties' inputs. For $P_j \notin Z$, it follows from the statistical robustness that even an unbounded adversary $\mathcal{A}$ can't compute such a $\hat{X}'_j$. For $P_i \in Z$, $\mathcal{A}$ can compute $\hat{X}'_i$ for any input X'_i; however, we ensure through the use of commitments that this input substitution will fail. The parties give both statistically binding and statistically hiding commitments on their inputs. The binding commitments remain t-out-of-n secret shared to ensure

the actual commitment, and therefore, the committed value remains hidden from an unbounded adversary. These shared commitments, although helpful in the unbounded case, fail to stop residual attacks in the computational case because the adversary's shares determine the commitment, and hence, the adversary can change the corrupt parties' shares according to any commitment on an arbitrary input. To prevent this, the parties additionally broadcast a statistically hiding commitment. The computational binding property of such a commitment prevents a PPT adversary from changing the inputs.

C Additional Material for Malicious Fall-back Secure MPC

C.1 Malicious Designated-Party Protocol

Similar to the semi-honest designated-party protocol, the maliciously secure designated-party protocol provides information-theoretic security to the designated party if it is honest, and computational security otherwise. Denoting the designated party as P_1, the security guarantees provided by such a protocol are as follows:

1. Information-theoretic security against an unbounded malicious adversary corrupting any subset of parties excluding the designated party P_1.
2. Computational security against a malicious PPT adversary corrupting an arbitrary subset of parties (possibly including the designated party P_1).

More formally, the designated-party protocol provides fall-back security with respect to the adversary structure $\mathcal{Z}_{\mathsf{D}} = \{Z \subseteq \{P_i\}_{i\in[n]\setminus\{1\}}\}$, i.e. the adversary structure consisting of any subset of parties excluding the designated party P_1. As in the semi-honest fall-back secure protocol, the parties compute each next-message function of the maliciously statistically secure protocol (the virtual protocol) through an instance of the designated-party protocol, where the designated party is the party whose next-message function is being computed.

To design a 2-round designated-party protocol, we begin with an arbitrary-round maliciously fall-back secure conforming protocol. We then use the modified compiler $\Pi_{\mathsf{m\text{-}des}}$ to squeeze this into a 2-round designated-party protocol. The resulting construction closely follows the semi-honest case, with the main difference being that the underlying conforming protocol is now the maliciously fall-back secure protocol of [AHPV23]. As in the semi-honest setting, our compiler is obtained by modifying the compiler of [GS18] in two ways. First, we replace the OT scheme with one that is statistically secure against an unbounded sender [PVW08] and second, all parties send their garbled circuits only to the designated party, who performs the computation locally.

In the resulting maliciously secure designated-party protocol, correctness follows from that of the conforming protocol, while privacy against an unbounded adversary (assuming the designated party is honest) follows from the security of the OT scheme. Security against PPT adversaries follows directly from [GS18].

We now give a formal description of the protocol. Without loss of generality, we consider P_1 to be the designated party.

Protocol C.1.1: 2-Round Designated-Party Protocol $\Pi_{\mathsf{m\text{-}des}}$ (UA)

Setting: Let $P = \{P_i\}_{i\in[n]}$ be the set of parties. Let Φ be an n-party conforming maliciously fall-back secure protocol tolerating $t < \frac{n}{2}$ corruptions, where only P_1 obtains the output.

Building Blocks:

- Garbling Scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ (Definition A.15).
- Maliciously secure (with equivocal receiver security) 2-round OT scheme secure against an unbounded sender $(\mathsf{K_{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

CRS: For each $t \in T$, $\alpha, \beta \in \{0,1\}$, sample $\sigma_{t,\alpha,\beta} \leftarrow \mathsf{K_{OT}}(1^\kappa)$ and output $\{\sigma_{t,\alpha,\beta}\}_{t\in[T],\alpha,\beta\in\{0,1\}}$ as the CRS.

The Protocol:

- **Round 1:** Each party P_i does:
  1. Compute $(z_i, v_i) \leftarrow \mathsf{pre}(1^\kappa, i, x_i)$.
  2. For each t where $\Phi_t = (i, f, g, h)$ (A_i is the set of such values t), for each $\alpha, \beta \in \{0,1\}$

$$\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(\sigma_{t,\alpha,\beta}, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta})$$

  3. Send $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ to all parties.
- **Round 2:** Each party P_i does:
  1. Compute $\mathsf{st}_i := (z_1 \| \cdots \| z_n) \oplus v_i$.
  2. Initialize labels $\overline{\mathsf{L}}^{i,T+1} := \{\mathsf{L}^{i,T+1}_{k,0}, \mathsf{L}^{i,T+1}_{k,1}\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$, $\mathsf{L}^{i,T+1}_{k,b} := 0^\kappa$.
  3. For each t from T down to 1:
     (a) Parse $\Phi_t = (i^*, f, g, h)$ and let P be as in Functionality B.1.2.
     (b) If $i = i^*$, compute

$$(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{L}}^{i,t}) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{P}[i, \Phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{L}}^{i,t+1}]).$$

     (c) If $i \neq i^*$, compute

$$\mathsf{ots}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta}, \mathsf{L}^{i,t+1}_{h,0}, \mathsf{L}^{i,t+1}_{h,1})$$

     and compute

$$(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{L}}^{i,t}) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{P}[i, \Phi_t, v_i, \bot, \{\mathsf{ots}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{L}}^{i,t+1}]).$$

  4. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{L}^{i,1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}\right)$ to P_1.
- **Evaluation:** Party P_1 does:
  1. Collect labels $\{\mathsf{L}^{j,1}_k\}_{k\in[\ell]}$ from all P_j.
  2. For $t = 1$ to T:
     (a) Parse $\Phi_t = (i^*, f, g, h)$.

(b) Compute $((\alpha, \beta, \gamma), \omega, \{\mathsf{L}_k^{i^*,t+1}\}_{k\in[\ell]}) \leftarrow \mathsf{Ev}(\widetilde{\mathsf{P}}^{i^*,t}, \{\mathsf{L}_k^{i^*,t}\}_{k\in[\ell]})$.
(c) Set $\mathsf{st}_{1,h} := \gamma \oplus v_{1,h}$.
(d) For each $j \neq i^*$:
i. Compute $(\mathsf{ots}_2, \{\mathsf{L}_k^{j,t+1}\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Ev}(\widetilde{\mathsf{P}}^{j,t}, \{\mathsf{L}_k^{j,t}\}_{k\in[\ell]})$.
ii. Recover $\mathsf{L}_h^{j,t+1} := \mathsf{OT}_3(\mathsf{ots}_2, \omega)$
3. Compute output $y_1 \leftarrow \mathsf{post}(1, \mathsf{st}_1)$.

Theorem C.1.2. *Let $\mathcal{Z}_\mathsf{D}$ be the set of all subsets of $\{P_i\}_{i\in[n]\setminus\{1\}}$. If $\mathsf{OT} = (\mathsf{K}_\mathsf{OT}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ is a maliciously secure 2-round OT protocol that is statistically secure against a corrupt sender, and $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is a garbling scheme for circuits, then $\Pi_{\mathsf{m\text{-}des}}$ is fall-back secure tolerating $\mathcal{Z}_\mathsf{D}$ for malicious adversaries and guaranteeing UA in the CRS model.*

We outline the proof here. The complete proof in given in Appendix E.1.

Proof Outline. We provide an informal justification for the fall-back security of the protocol $\Pi_{\mathsf{m\text{-}des}}$, obtained by compiling a malicious fall-back secure protocol using the modified compiler. First, consider an unbounded adversary corrupting a subset of the parties in $\{P_i\}_{i\in[n]\setminus\{1\}}$. In this setting, the only information received from an honest party P_j consists of the OT_1 messages corresponding to its inputs, together with the value z_j generated during the pre phase of Φ and sent in the first round. Since z_j is simply a one-time pad masking of the input x_j, it reveals no information on its own. Furthermore, the OT primitive we use provides statistical security against a malicious sender, ensuring that the OT interaction does not leak any additional information about the inputs.

Now consider a PPT adversary corrupting a subset of the parties. In this case, security follows from the guarantees of the original compiler of [GS18] together with the security of the underlying fall-back secure conforming protocol. It suffices to consider the case in which the designated party is corrupted, since all other cases are already covered by the statistical argument above. When the designated party is corrupt, for any honest party P_j, the adversary learns the OT_1 messages corresponding to P_j's input, the masked value z_j, and the garbled circuits sent by P_j to P_1. This is precisely the view obtained by the adversary in the 2-round compiled protocol of [GS18]. Hence, security follows directly from the analysis of their compiler and the security of the underlying protocol.

D Deferred Semi-Honest Proofs

D.1 Proof of Semi-Honest Designated Protocol (Theorem B.1.3)

Proof. **Information-Theoretic Security to Designated Party.** First, we show that there exists a PPT simulator Sim_S that, for any subset of parties $Z \in \mathcal{Z}_\mathsf{D}$ corrupted by an unbounded adversary $\mathcal{A}$, can output a view whose distribution is statistically close to the view in the real protocol in the plain model. The PPT simulator Sim_S operates as follows:

1. $\mathsf{Sim_S}$ has the input of all the corrupt parties $\{x_i\}_{P_i \in Z}$. Let $\mathcal{R}$ be the domain of each party's randomness. $\mathsf{Sim_S}$ samples randomness $\{r_i \leftarrow \mathcal{R}\}_{P_i \in Z}$ for all corrupt parties.
2. For each t where $\Phi_t = (j, f, g, h)$ and $P_j \notin Z$, $\mathsf{Sim_S}$ randomly samples $r_j \leftarrow \{0,1\}^m$ and sets $\hat{z}_j = r_j \| 0^{\ell'}$.
3. Denote $A_j = \{t \mid \Phi_t = (j, f, g, h)\}$. For each $P_j \notin Z$, $\mathsf{Sim_S}$ generates the $\hat{\mathsf{ots}}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\kappa, 0; \hat{\omega}_{t,\alpha,\beta})$ for $t \in A_j, \alpha, \beta \in \{0,1\}$.
4. $\mathsf{Sim_S}$ outputs the simulated view as:

$$\{x_i, r_i\}_{P_i \in Z}, \{\hat{z}_j, \{\hat{\mathsf{ots}}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{P_j \notin Z}$$

We prove that the real view and the simulated view are identically distributed through a series of hybrids. Let $|Z| = d$. To simplify notation, we assume that the parties $P_1, \ldots, P_{n-d}$ are honest. We now consider the following $n - d + 2$ hybrids:

– Hybrid H_0: This is the real view of the parties in the protocol

$$\mathsf{H}_0 = \{\{x_i, r_i\}_{P_i \in Z}, \{z_j, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{j \in [n-d]}\}$$

– Hybrid H_k: In this hybrid, the OT_1 messages sent by the first k honest parties are replaced with those generated by $\mathsf{Sim_S}$. Thus, the distribution in this hybrid is

$$\mathsf{H}_k = \{\{x_i, r_i\}_{P_i \in Z}, \{z_j, \hat{\mathsf{ots}}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{j \le k}, \{z_j, \mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{j \in [n-d] \setminus [k]}\}$$

– Hybrid H_{n-d}: In this hybrid, the OT_1 messages received from all the honest parties are replaced with those generated by $\mathsf{Sim_S}$. Thus, the distribution in this hybrid is

$$\mathsf{H}_{n-d} = \{\{x_i, r_i\}_{P_i \in Z}, \{z_j, \{\hat{\mathsf{ots}}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{j \in [n-d]}\}$$

– Hybrid H_{n-d+1}: In this hybrid, the $\{z_j\}_{P_j \notin Z}$ messages are replaced with uniformly sampled values $\{\hat{z}_j\}_{P_j \notin Z}$. The distribution in this hybrid is the distribution of the simulated view.

$$\mathsf{H}_{n-d+1} = \{\{x_i, r_i\}_{P_i \in Z}, \{\hat{z}_j, \{\hat{\mathsf{ots}}_{1,t,\alpha,\beta}\}_{t \in A_j, (\alpha,\beta) \in \{0,1\}}\}_{j \in [n-d]}\}$$

Claim D.1.1. *Assuming that the OT protocol* OT *is semi-honest IT-secure against an unbounded adversary corrupting the sender, the view in the hybrid distribution* H_{j-1} *and* H_j *is identically distributed for all* $j \in [n-d]$.

Proof. Each pair of adjacent hybrids differs only in the OT_1 messages. From the security of the OT protocol, we have the following

$$\{\mathsf{OT}_1(1^\kappa, 0; \omega_0)\} \stackrel{s}{\equiv} \{\mathsf{OT}_1(1^\kappa, 1; \omega_1)\}$$

where the two distributions are defined over the randomness ω_0 and ω_1 respectively. Thus, we have that

$$\{\mathsf{OT}_1(1^\kappa, v_{j,h} \oplus \mathsf{NAND}(v_{j,f} \oplus \alpha, v_{j,g} \oplus \beta); \omega_{t,\alpha,\beta})\} \stackrel{s}{\equiv} \{\mathsf{OT}_1(1^\kappa, 0; \hat{\omega}_{t,\alpha,\beta})\}$$

for all $\alpha, \beta \in \{0,1\}$, $t \in A_j$ and $P_j \notin Z$.

Thus, the hybrids H_0 and H_{n-d} are statistically indistinguishable. To see that the hybrids H_{n-d} and H_{n-d+1} are identically distributed, observe that $z_j = x_j \oplus r_j \| 0^{\ell'}$. Since r_j is uniformly distributed, this is identically distributed as $\hat{z}_j$. Thus $\mathsf{H}_0 \stackrel{s}{\equiv} \mathsf{H}_{n-d+1}$. □

Computational Security. It remains to handle the case of a computational adversary. The computational security immediately follows from the simulator provided by [GS18], and we adopt their proof in the computational setting. The only change is that only the designated party gets the garbled circuit and the output, while the other parties receive $\perp$. We describe it below for completeness.

Let $\mathcal{A} \subseteq [n]$ be a semi-honest adversary corrupting a subset of parties $Z \subset P$. Let $|Z| = d$. We construct a PPT simulator Sim_C interacting with ideal functionality $\mathcal{F}$ and internally running $\mathcal{A}$. The simulator uses the semi-honest conforming-protocol simulator Sim_Φ and the garbling simulator Sim_GC as subroutines. The gates are indexed from $\{1, \ldots, T\}$. For gate t, we denote the action $\Phi_t = (i^*, f, g, h)$ where i^* is the designated party for round t.

- Initialization: We consider P_1 to be the designated party without loss of generality. The simulator Sim_C uses the inputs of corrupted parties $\{x_i\}_{i \in Z}$ and the output y if $P_1 \in Z$ of the ideal functionality $\mathcal{F}$ to generate a simulated view for the adversary. For each $P_i \in Z$, Sim_C sends $(\mathsf{input}, \{P_1, \ldots, P_n\}, P_i, x_i)$ to the ideal functionality $\mathcal{F}$. The functionality $\mathcal{F}$ outputs the function output y to party P_1, else sends $\perp$. We consider only the case where $P_1 \in Z$, since $P_1 \notin Z$ security is ensured from the prior case.
  Sim_C runs $\mathsf{Sim}_\Phi(1^\kappa, \{x_i\}_{i \in Z}, y)$ to obtain $\{z_i\}_{i \notin Z}$, the random tapes for corrupted parties, and the transcript of computation denoted by $\tau \in \{0,1\}^T$, where τ_t represents the bit sent in the t-th round of computation of Φ. It also obtains the value st^*, where each st^*_i for $i \in [n]$ and $k \in [\ell]$ satisfies $\mathsf{st}^*_{i,k} = \mathsf{st}_{i,k} \oplus v_{i,k}$. Sim_C then starts the real-world adversary $\mathcal{A}$ with inputs $\{z_i\}_{i \notin Z}$ and random tape from Sim_Φ.
- Round 1 messages from Sim_C to $\mathcal{A}$: For each $P_i \notin Z$, $t \in A_i$, and $\alpha, \beta \in \{0,1\}$, the simulator generates $\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\kappa, \tau_t; \omega_{t,\alpha,\beta})$. For each $P_i \notin Z$, Sim_C sends $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta \in \{0,1\}})$ to $\mathcal{A}$ on behalf of P_i.
- Round 1 messages from $\mathcal{A}$ to Sim_C: For each $P_i \in Z$, Sim_C receives from $\mathcal{A}$ the tuple $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta \in \{0,1\}})$ corresponding to the corrupted party P_i.
- Round 2 messages from Sim_C to $\mathcal{A}$: For each $P_i \notin Z$, Sim_C generates the second-round message for P_i as follows:
  1. For each $k \in [\ell]$, set $\mathsf{L}_k^{i,T+1} := 0^\kappa$.

2. For each round t from T down to 1:
   (a) Parse Φ_t as (i^*, f, g, h).
   (b) Set $\alpha^* := \mathsf{st}_f^*$, $\beta^* := \mathsf{st}_g^*$, and $\gamma^* := \mathsf{st}_h^*$.
   (c) If $i = i^*$, then compute

$$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\big(1^\kappa, ((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{L}_k^{i,t+1}\}_{k\in[\ell]})\big).$$

   (d) If $i \neq i^*$, then set

$$\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(1^\kappa, \mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{L}_h^{i,t+1}, \mathsf{L}_h^{i,t+1})$$

   and compute

$$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\big(1^\kappa, (\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{L}_k^{i,t+1}\}_{k\in[\ell]\setminus\{h\}})\big).$$

3. Finally, send $\big((\widetilde{\mathsf{P}}^{i,t})_{t\in[T]}, \{\mathsf{L}_k^{i,1}\}_{k\in[\ell]}\big)$ to P_1.

– Round 2 messages from $\mathcal{A}$ to Sim_C: For every $P_i \in Z$, Sim_C receives the second-round messages from $\mathcal{A}$ on behalf of corrupted parties. Upon receiving all such messages, for each $P_i \notin Z$, Sim_C sends $(\mathsf{generateOutput}, \{P_1, \ldots, P_n\}, P_i)$ to the ideal functionality.

We show that no PPT adversary $\mathcal{A}$ can distinguish whether it is interacting with the real execution or the ideal execution with the simulator Sim_C described. The proof is by a hybrid argument that walks from the real world distribution to the ideal world distribution through $T+1$ intermediate hybrids, where T is the number of gates in the conforming protocol Φ.

Let H_0 denote the distribution of the real execution.

For $t \in \{1, \ldots, T\}$, define hybrid H_t by changing how the OT messages and garbled circuits that affect the t-th action $\Phi_t = (i^*, f, g, h)$ are produced; everything else remains as in the previous hybrid. Finally, obtain H_{T+1} by replacing the remaining honest parties randomness and the transcript with values produced by Sim_Φ (run on the corrupted parties inputs together with the ideal output). Hybrid H_{T+1} matches the ideal world distribution produced by the simulator Sim_C.

We show that each adjacent pair of hybrids is computationally indistinguishable. Fix an index t and let $\Phi_t = (i^*, f, g, h)$. Execute the protocol Φ (or equivalently consult Sim_Φ) to obtain the computation-transcript bits $\tau \in \{0,1\}^t$. Let st^* denote the local state at the end of execution. Denote the gate-specific bit by τ_t, and let $\alpha^*, \beta^*, \gamma^*$ be the simulated truth values for the two input wires and the output wire of gate t (the corresponding entries in st^*).

For the t-th gate, we replace the real generation of the following objects by simulator generated ones:

- the Round-1 OT_1 transcripts $\mathsf{ots}_{1,t,\alpha,\beta}$ that honest parties would produce,
- the OT_2 encodings that honest parties would normally produce or embed in their garbled circuits. The garbled circuits and labels $\widetilde{\mathsf{P}}^{i,t}$ and $\{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}$ that honest parties would send in Round 2.

Claim D.1.2. *Assuming the semi-honest security of the garbled-circuit scheme and the two-round* OT *protocol, for all* $t \in 1, \ldots T$ *hybrids* H_{t-1} *and* H_t *are computationally indistinguishable.*

We perform the replacement via three sub-hybrids, denoted $\mathsf{H}_{t,1}$, $\mathsf{H}_{t,2}$, $\mathsf{H}_{t,3}$. Each sub-step is chosen so that it can be justified by a single primitive guarantee: the garbling simulator, OT_2 encoding security, and OT_1 receiver security, respectively. When $i^* \in Z$, we just skip the steps.

- Hybrid $\mathsf{H}_{t,1}$: replacing garbled circuits by simulated garbled circuits. In $\mathsf{H}_{t,1}$ we substitute the honest parties' garbled circuits and input labels that participate in gate t with outputs from the garbling simulator. $\mathsf{Sim}_{\mathsf{GC}}$. Concretely
  If $i = i^*$ then

  $$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa, ((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k\in[\ell]})\Big)$$

  If $i \neq i^*$ then

  $$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa, (\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k\in[\ell]\setminus\{h\}})\Big)$$

  where $\{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.
  The computational indistinguishability between hybrids $\mathsf{H}_{t,1}$ and H_{t-1} follows from $n - d$ invocations of security of the garbling scheme.
- Hybrid $\mathsf{H}_{t,2}$: We skip this hybrid if there is no $i \neq i^*$ such that $P_i \notin Z$. In this hybrid we change how $\mathsf{Sim}_{\mathsf{C}}$ generates $\mathsf{ots}^i_{2,t,\alpha,\beta}$ on behalf of honest parties where $i \neq i^*$. In the real protocol, each honest sender prepares four OT_2 responses (one per choice (α, β)) to P_1. In $\mathsf{H}_{t,2}$, the simulator has honest senders produce only the single OT_2 response corresponding to the simulated choice (α^*, β^*), which is now generated as $\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{ots}^i_{1,t,\alpha^*,\beta^*}, \mathsf{L}^{i,t+1}_{h,\tau_t}, \mathsf{L}^{i,t}_{h,\tau_t})$ instead of $\mathsf{OT}_2(\mathsf{ots}^i_{1,t,\alpha^*,\beta^*}, \mathsf{L}^{i,t+1}_{h,0}, \mathsf{L}^{i,t+1}_{h,1})$. Indistinguishability of $\mathsf{H}_{t,1} \to \mathsf{H}_{t,2}$ is argued based on the sender's security of the underlying semi-honest OT protocol.
- Hybrid $\mathsf{H}_{t,3}$: We skip this hybrid if $i^* \in Z$. In this hybrid, we replace Round-1 OT_1 transcripts with simulated ones. Finally, we replace the OT_1 transcripts that honest parties sent for gate t by transcripts computed using the simulated choice bit τ_t. That is, for each honest party, we set

  $$\mathsf{ots}_{1,t,\alpha,\beta} := \mathsf{OT}_1\big(1^\kappa,\ \tau_t;\ \omega_{t,\alpha,\beta}\big)$$

  using freshly simulated randomness $\omega_{t,\alpha,\beta}$ instead of generating $\mathsf{OT}_1\big(1^\kappa, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta}\big)$. Because the OT primitive in our construction provides statistical protection of the receiver's choice-bit against a corrupt unbounded sender, a corrupted sender cannot distinguish whether the OT_1 transcripts were generated with the true choice bit or with the simulated bit. Therefore the transition $\mathsf{H}_{t,2} \to \mathsf{H}_{t,3}$ is statistically indistinguishable.

After these three sub-steps, we obtain hybrid H_t. Therefore H_{t-1} and H_t are indistinguishable: $\mathsf{H}_{t,1} \to \mathsf{H}_{t,2}$ is computationally indistinguishable under the garbling security, while $\mathsf{H}_{t,2} \to \mathsf{H}_{t,3}$ is statistically indistinguishable due to the OT_1 receiver-side guarantee. Applying the above replacement for every $t = 1, \ldots, T$ yields H_T. To reach H_{T+1} we replace the way the global transcript τ, the honest-parties preprocessing values $\{z_i\}_{i \notin Z}$, and the simulated shared state st^* are produced: instead of deriving them from honest parties' real inputs, we obtain these values by running Sim_{Φ} on the corrupted parties' inputs together with the ideal output y provided by $\mathcal{F}$. By the semi-honest security of Φ, this change is computationally indistinguishable. The distribution H_{T+1} therefore equals the ideal world distribution produced by $\mathsf{Sim}_{\mathsf{C}}$.
For every $t \in \{1, \ldots, T\}$ the hybrids H_{t-1} and H_t are indistinguishable to any PPT distinguisher. Combining all hybrid steps and the final replacement yields that H_0 and H_{T+1} are computationally indistinguishable. Since H_{T+1} is the simulator's ideal world distribution, no PPT adversary $\mathcal{A}$ can distinguish the real execution from the ideal execution except with negligible probability, completing the proof.

□

D.2 Proof of Semi-Honest Fallback Protocol (Theorem 3.4)

Proof. The proof of the theorem consists of two parts. The first part will show the existence of the PPT simulator $\mathsf{Sim}_{\mathsf{C}}$ that for any subset of parties $Z \in \mathcal{Z}_{\mathsf{C}} = 2^P$ can generate a view that is computationally indistinguishable from the view in the real protocol. The second part shows the existence of a simulator $\mathsf{Sim}_{\mathsf{S}}$ that, for any subset of $t < \frac{n}{2}$, can output a view that is statistically indistinguishable from the view in the real protocol. We introduce some notation that is used to prove the security of the protocol in the plain model.

1. Just as Π_{sh} can be written as the set of functions $\{\mathsf{NM}_i^j\}_{i \in [n], j \in [3]}$, the simulator $\mathsf{Sim}_{\mathsf{sh}}$ can also be written as,

$$\mathsf{Sim}_{\mathsf{sh}} = \{\mathsf{NM}_i^{j*}\}_{i \in [n], j \in [3]}$$

For each corrupted party $P_i \in Z$, the simulated next-message functions are as in the real protocol Π_{sh} i.e. $\{\mathsf{NM}_i^{j*}\}_{j \in [3]} = \{\mathsf{NM}_i^j\}_{j \in [3]}$. However, for each honest party $P_i \notin Z$, for each round $j \in [3]$, the next-message function has the same structure as the real function but produces simulated messages that do not depend on the party's input:

$$\{\hat{T}_{ik}^j\}_{k \in [n]} \leftarrow \mathsf{NM}_i^{j*}\left(\widehat{\mathsf{View}}_{\mathsf{sh},i}^{\,j-1}\right)$$

Here $\widehat{\mathsf{View}}_{\mathsf{sh},i}^{\,j-1}$ is the simulated view that P_i receives up to round $j-1$.

2. The real protocol $\Pi_{\mathsf{sh\text{-}fb}}$ is composed of the set of protocols $\{\Pi^i_{\mathsf{sh\text{-}des}} = (\Pi^{G_i}_{\mathsf{sh\text{-}des}}, \Pi^{H_i}_{\mathsf{sh\text{-}des}})\}_{i\in[n]}$ corresponding to Π_{sh}, where each $\Pi^i_{\mathsf{sh\text{-}des}}$ works with P_i as the designated party. From theorem B.1.3, we know that for a PPT semi-honest adversary corrupting $Z \subset P$, there exists a simulator, denoted as $\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{C}}$, that when provided with inputs and outputs of the corrupt parties can produce a view that is computationally indistinguishable from the view in the real execution of the sub-protocol. We define View^Z_i as the view of the adversary Z in the real $\Pi^i_{\mathsf{sh\text{-}des}}$ instance, and $\widehat{\mathsf{View}}^Z_i$ as the view of the adversary in the simulated instance, i.e. the view generated by $\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{C}}$. Then, the following holds

$$\left\{\widehat{\mathsf{View}}^Z_i\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{r}\in R^n} \overset{c}{\equiv} \left\{\mathsf{View}^Z_i\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{r}\in R^n}$$

From theorem B.1.3, we also have that for an unbounded adversary corrupting $Z \subseteq P \setminus \{P_i\}$, there exists a PPT simulator denoted as $\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{S}}$, that when provided with inputs and outputs of the corrupt parties can produce a view that is statistically indistinguishable from the view in the real execution of the sub-protocol. We overload notation and denote this view as $\widehat{\mathsf{View}}^Z_i$ as well. In our proofs, $\widehat{\mathsf{View}}^Z_i$ denotes the view generated either by $\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{S}}$ or $\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{C}}$, depending on the corruption case being considered.

$$\left\{\widehat{\mathsf{View}}^Z_i\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{r}\in R^n} \overset{s}{\equiv} \left\{\mathsf{View}^Z_i\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{r}\in R^n}$$

With these necessary blocks, we can prove the theorem in the plain model.

Computational Security. We now consider the case of security against a PPT semi-honest adversary that corrupts any $t < n$ parties. For the proof, we consider the instances $\Pi^{G_i}_{\mathsf{sh\text{-}des}}$ and $\Pi^{H_i}_{\mathsf{sh\text{-}des}}$ as a single instance $\Pi^i_{\mathsf{sh\text{-}des}}$. The simulator, denoted as $\mathsf{Sim}_{\mathsf{C}}$, has the inputs and outputs of all corrupt parties as $\{x_i, y_i\}_{P_i\in Z}$. It makes black-box access to simulators in $\{\mathsf{Sim}^k_{\mathsf{sh\text{-}des},\mathsf{C}}\}_{k\in[n]}$ and for each $P_i \in Z$, does the following:

- $\mathsf{Sim}_{\mathsf{C}}$ samples random tape r_i for P_i and computes

$$\{T^1_{ij}\}_{j\in[n]} = \mathsf{NM}^1_i(x_i, r_i)$$

- $\mathsf{Sim}_{\mathsf{C}}$ invokes the garbled circuit simulator to obtain the simulated circuit $\widehat{C_i}$ and labels $\{\widehat{\mathsf{L}}_{j,i,h}\}_{j\in[n],h\in[\ell]}$ as

$$(\widehat{C_i}, \{\widehat{\mathsf{L}}_{j,i,h}\}_{j\in[n],h\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(y_i)$$

- $\mathsf{Sim}_{\mathsf{C}}$ randomly samples additive secret shares of the labels, denoted as

$$\{\widehat{\mathsf{L}}^b_{j,k,i,h}\}_{j,k\in[n],h\in[\ell],b\in\{0,1\}}$$

– For all $k \in [n]$, $\mathsf{Sim}_{\mathsf{C}}$ calls the simulator of the designated protocol to generate the view.

$$\mathsf{Sim}^{k}_{\mathsf{sh-des},\mathsf{C}}(\{T^{1}_{ik}, \{\widehat{\mathsf{L}}^{b}_{k,j,i,h}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}\}_{P_i\in Z}, z_k)$$

where $z_k = (\widehat{C_k}, \{\widehat{\mathsf{L}}_{k,j,h}\}_{j\in[n],h\in[\ell]})$ if $P_k \in Z$ and $z_k = \perp$ otherwise. Denote this view as $\widehat{\mathsf{View}}^{Z}_{k}$.

– $\mathsf{Sim}_{\mathsf{C}}$ generates the view of round 3 in the protocol by sending $\{\widehat{\mathsf{L}}_{k,i,h}\}_{k\notin Z,h\in[\ell]}$.

This completes the simulation. We denote the entire random tape of P_i, along with the label shares sampled by $\mathsf{Sim}_{\mathsf{C}}$ during the simulation, as R_i. The view of the adversary in the above execution of $\mathsf{Sim}_{\mathsf{C}}$ is distributed as,

$$\left\{\{x_i, y_i, R_i, \{\widehat{\mathsf{L}}_{k,i,h}\}_{P_k\notin Z,h\in[\ell]}\}_{P_i\in Z}, \{\widehat{\mathsf{View}}^{Z}_{k}\}_{k\in[n]}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

We now prove that the above view is computationally indistinguishable from the view of the adversary in the protocol. The simulated view differs from the real view in the protocol as the view of the next-message protocols is composed of calls to $\mathsf{Sim}^{i}_{\mathsf{sh-des},\mathsf{C}}$, and the simulated garbled circuits in the simulation. To show that the output of $\mathsf{Sim}_{\mathsf{C}}$ is indistinguishable from the real view, we consider 3 hybrids, and then argue their indistinguishability by considering n intermediate hybrids between each consecutive pair of hybrids:

– Hybrid H_0: This is the view obtained from a real execution of the protocol.

$$\left\{\{x_i, y_i, R_i, \{\mathsf{L}_{k,i,h}\}_{P_k\notin Z,h\in[\ell]}\}_{P_i\in Z}, \{\mathsf{View}^{Z}_{k}\}_{k\in[n]}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

– Hybrid H_1: In this hybrid, for all $i \in [n]$, we replace the execution of all the $\Pi^{i}_{\mathsf{sh-des}}$ instances with corresponding instances of $\mathsf{Sim}^{i}_{\mathsf{sh-des},\mathsf{C}}$.

$$\left\{\{x_i, y_i, R_i, \{\mathsf{L}_{k,i,h}\}_{P_k\notin Z,h\in[\ell]}\}_{P_i\in Z}, \{\mathsf{View}'^{Z}_{i}\}_{i\in[n]}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

– Hybrid H_2: In this hybrid, we replace the garbled circuit and the labels i.e. $(C_j, \{\mathsf{L}_{j,k,h}\}_{k\in[n],h\in[\ell]})$ with the circuit and labels generated by $\mathsf{Sim}_{\mathsf{GC}}$ i.e. $(\widehat{C_k}, \{\widehat{\mathsf{L}}_{k,j,h}\}_{j\in[n],h\in[\ell]})$.

$$\left\{\{x_i, y_i, R_i, \{\widehat{\mathsf{L}}_{k,i,h}\}_{P_k\notin Z,h\in[\ell]}\}_{P_i\in Z}, \{\widehat{\mathsf{View}}^{Z}_{k}\}_{k\in[n]}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

Here, we define the intermediate view View'^{Z}_{k} as

$$\mathsf{View}'^{Z}_{k} \leftarrow \mathsf{Sim}^{k}_{\mathsf{sh-des},\mathsf{C}}(\{T^{1}_{ik}, \{\mathsf{L}^{b}_{k,j,i,h}\}_{j\in[n],h\in[\ell],b\in\{0,1\}}\}_{P_i\in Z}, z_k)$$

where $z_k = (C_k, \{\mathsf{L}_{k,j,h}\}_{j\in[n],h\in[\ell]})$ if $P_k \in Z$ and $z_k = \perp$ otherwise. The difference between this intermediate view and the simulated view is that the intermediate view is generated by calling $\mathsf{Sim}^{k}_{\mathsf{sh-des},\mathsf{C}}$ on the garbled circuits and labels obtained

in the real execution, while the simulated view is generated by calling $\mathsf{Sim}^{k}_{\mathsf{sh\text{-}des},\mathsf{C}}$ on the output of $\mathsf{Sim}_{\mathsf{GC}}$.
We now define the intermediate hybrid $\mathsf{H}_{0,j}$ for $j \in [n]$ as

$$\left\{\{x_i, y_i, R_i, \{\mathsf{L}_{k,i,h}\}_{P_k \notin Z, h\in[\ell]}\}_{P_i\in Z}, \{\mathsf{View}'^{Z}_{k}\}_{k\le j}, \{\mathsf{View}^{Z}_{k}\}_{k>j}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

and similarly, define the intermediate hybrid $\mathsf{H}_{1,j}$ as

$$\left\{\{x_i, y_i, R_i, \{\mathsf{L}_{k,i,h}\}_{P_k \notin Z, h\in[\ell]}\}_{P_i\in Z}, \{\widehat{\mathsf{View}}^{Z}_{k}\}_{k\le j}, \{\mathsf{View}'^{Z}_{k}\}_{k>j}\right\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$$

Claim D.2.1. *Assuming that the protocol $\Pi_{\mathsf{sh\text{-}des}}$ is secure in the presence of a PPT semi-honest adversary corrupting any subset $Z \subset P$ of the parties, the view in the hybrid distribution $\mathsf{H}_{0,j}$ is computationally indistinguishable from that in $\mathsf{H}_{0,j-1}$.*

Proof. We can observe that in the hybrid distributions defined above, the pair of adjacent hybrids $\mathsf{H}_{0,j}$ and $\mathsf{H}_{0,j-1}$ differ only in the view of the execution of the sub-protocol $\Pi^{j}_{\mathsf{sh\text{-}des}}$. In $\mathsf{H}_{0,j}$, this view is the simulated view View'^{Z}_{j} output by $\mathsf{Sim}^{j}_{\mathsf{sh\text{-}des},\mathsf{C}}$ while in $\mathsf{H}_{0,j-1}$, this is the view of the parties in Z in the real execution of $\Pi^{j}_{\mathsf{sh\text{-}des}}$.

We show that if there exists a PPT distinguisher $\mathcal{D}$ that can distinguish between the hybrid distributions $\mathsf{H}_{0,j}$ and $\mathsf{H}_{0,j-1}$ with non-negligible advantage ε, then $\mathcal{D}$ can be used in a black-box manner by a PPT adversary $\mathcal{A}$ to distinguish between the distributions $\{\mathsf{View}'^{Z}_{j}\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$ and $\{\mathsf{View}^{Z}_{j}\}_{\kappa\in\mathbb{N},\, \boldsymbol{R}\in R^n}$ from the real execution of the sub-protocol. The adversary $\mathcal{A}$ operates as follows:

- $\mathcal{A}$ is given the index j, the set of corrupted parties $Z \subset P$, and the pair $(\boldsymbol{x} = (x_1, ..., x_n), f(\boldsymbol{x}))$. It samples randomness $\boldsymbol{R} = (R_1, ..., R_n) \leftarrow R^n$ for all parties.
- For the instances $\Pi^{j'}_{\mathsf{sh\text{-}des}}$ where $j' < j$, it generates the view using the simulator, namely $\mathsf{View}'^{Z}_{j'}$.
- For the instances $\Pi^{j'}_{\mathsf{sh\text{-}des}}$ where $j' > j$, it generates the view according to the real protocol execution, namely $\mathsf{View}^{Z}_{j'}$.
- Corresponding to the j-th instance, $\mathcal{A}$ sends all the inputs and outputs to the challenger $\mathcal{C}$ that samples a bit b. If $b = 0$, $\mathcal{C}$ computes,

  $$\mathsf{View}'^{Z}_{j} \leftarrow \mathsf{Sim}^{j}_{\mathsf{sh\text{-}des},\mathsf{C}}(\{T^{1}_{ij}, \{\mathsf{L}^{b}_{j,k,i,h}\}_{k\in[n],h\in[\ell],b\in\{0,1\}}\}_{P_i\in Z}, z_j)$$

  where $z_j = (C_j, \{\mathsf{L}_{j,k,h}\}_{k\in[n],h\in[\ell]})$ if $P_j \in Z$ and $z_j = \perp$ otherwise.
  Else, it computes View^{Z}_{j} to be the views of the parties in Z in the real execution of $\Pi^{j}_{\mathsf{sh\text{-}des}}$.

In the above construction, the adversary $\mathcal{A}$ sees the simulated view if $b = 0$, and the actual view when $b = 1$. It thus achieves a distinguishing advantage greater than or equal to that of $\mathcal{D}$'s. Thus, no PPT $\mathcal{D}$ can have a non-negligible

advantage, since this contradicts the assumed security of the protocol $\Pi^{i}_{\mathsf{sh\text{-}des}}$ against a PPT semi-honest adversary corrupting any subset $Z \subset P$ of the parties. This implies that the hybrid distributions $\mathsf{H}_{0,j}$ and $\mathsf{H}_{0,j-1}$ are computationally indistinguishable. □

Since each pair of adjacent hybrids in the above sequence is computationally indistinguishable, we conclude that the view in H_1 is computationally indistinguishable from the real view H_0.
We now prove that the hybrids H_1 and H_2 are computationally indistinguishable.

Claim D.2.2. *Assuming that the garbling scheme is private in the presence of a PPT semi-honest adversary corrupting any subset $Z \subset P$ of the parties, the view in the hybrid distribution $\mathsf{H}_{1,j}$ is computationally indistinguishable from that in $\mathsf{H}_{1,j-1}$.*

Proof. We can observe that in the hybrid distributions defined above, the pair of adjacent hybrids $\mathsf{H}_{1,j}$ and $\mathsf{H}_{1,j-1}$ differ only in the garbled circuits and the labels used for the simulation. Note that if $P_j \notin Z$, the two hybrids $\mathsf{H}_{1,j-1}$ and $\mathsf{H}_{1,j}$ are the same since C_j is not used by $\mathsf{Sim}_{\mathsf{C}}$. We argue the indistinguishability between subsequent hybrids assuming $P_j \in Z$. In $\mathsf{H}_{1,j-1}$, this is the simulated view View'^{Z}_{j} generated on the circuit and labels in the real execution, while in $\mathsf{H}_{1,j}$, this is the simulated view $\widehat{\mathsf{View}}^{Z}_{j}$ generated on the simulated circuit and the labels.

We show that if there exists a PPT distinguisher $\mathcal{D}$ that can distinguish between the hybrid distributions $\mathsf{H}_{1,j}$ and $\mathsf{H}_{1,j-1}$ with non-negligible advantage ε, then $\mathcal{D}$ can be used in a black-box manner by a PPT adversary $\mathcal{A}$ to distinguish between the distributions $\{C_j, \{\mathsf{L}_{j,k,h}\}_{k\in[n],h\in[\ell]}\}$ and $\{\widehat{C_j}, \{\widehat{\mathsf{L}}_{j,k,h}\}_{k\in[n],h\in[\ell]}\}$ with advantage at least equal to ε. The adversary $\mathcal{A}$ operates as follows:

- $\mathcal{A}$ is given the index j, the set of corrupted parties $Z \subset P$, and the pair $(\boldsymbol{x} = (x_1, ..., x_n), f)$. It samples randomness $\boldsymbol{R} = (R_1, ..., R_n) \leftarrow R^n$ for all parties.
- For the instances of $\Pi^{j'}_{\mathsf{sh\text{-}des}}$ where $j' < j$, it calls $\mathsf{Sim}_{\mathsf{GC}}$ to obtain $(\widehat{C_{j'}}, \{\widehat{\mathsf{L}}_{j',k,h}\}_{k\in[n],h\in[\ell]})$, namely $\widehat{\mathsf{View}}^{Z}_{j'}$.
- For the instances of $\Pi^{j'}_{\mathsf{sh\text{-}des}}$ where $j' > j$, it simulates the view using $(C_{j'}, \{\mathsf{L}_{j',k,h}\}_{k\in[n],h\in[\ell]})$, namely $\mathsf{View}'^{Z}_{j'}$.
- Corresponding to the j-th instance, $\mathcal{A}$ sends all the inputs and outputs to the challenger $\mathcal{C}$ that samples a bit b. If $b = 0$, $\mathcal{C}$ computes,

$$(\widehat{C_j}, \{\widehat{\mathsf{L}}_{j,k,h}\}_{k\in[n],h\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(y_j)$$

  Else, it computes $(C_j, \{\mathsf{L}_{j,k,h}\}_{k\in[n],h\in[\ell]})$ as in the real protocol.

In the above construction, the adversary $\mathcal{A}$ achieves the same (non-negligible) distinguishing advantage as $\mathcal{D}$. However, this contradicts the assumed privacy of the garbling scheme against a PPT semi-honest adversary. Consequently,

such an adversary $\mathcal{A}$ cannot exist, and hence neither can a distinguisher $\mathcal{D}$. It follows that the hybrid distributions $\mathsf{H}_{1,j}$ and $\mathsf{H}_{1,j-1}$ are computationally indistinguishable. □

Since no pair of adjacent hybrids in the above sequence can be distinguished, we conclude that the view in H_2 is computationally indistinguishable from the simulated view H_1. Therefore, the view generated by Sim_C is computationally indistinguishable from the view in the real execution.

Information-Theoretic Security. We now consider the case of IT security against an unbounded semi-honest adversary corrupting at most $t < \frac{n}{2}$ parties. Let $Z_V \subset P$ denote the set of corrupted parties, where $|Z_V| < \frac{n}{2}$. Since the number of corrupt parties $< \frac{n}{2}$, the Π_{sh} is secure against an unbounded semi-honest adversary corrupting Z_V. We denote the overall view of parties in Z_V in the designated-party protocol instance $\Pi^k_{\mathsf{sh\text{-}des}}$ as $\mathsf{View}^{Z_V}_k$, and in the corresponding simulated instance (generated by $\mathsf{Sim}^k_{\mathsf{sh\text{-}des},\mathsf{S}}$) as $\widehat{\mathsf{View}}^{Z_V}_k$. A designated instance computing the simulated next-message function NM^{r*}_j instead of the actual next-message function NM^r_j is denoted as $\Pi^{j*}_{\mathsf{sh\text{-}des}}$. The overall view of parties in Z_V in such a designated-party protocol instance $\Pi^{k*}_{\mathsf{sh\text{-}des}}$ as $\mathsf{View}^{*Z_V}_k$, and in the corresponding simulated instance as $\widehat{\mathsf{View}}^{*Z_V}_k$.

The simulator Sim_S for $\Pi_{\mathsf{sh\text{-}fb}}$, which makes black-box calls to the simulators in $\{\mathsf{Sim}^i_{\mathsf{sh\text{-}des},\mathsf{S}}\}_{i\in[n]}$, operates as follows:

1. For $P_i \in Z_V$, Sim_S has the input of the corrupt parties x_i and the function output $f(\boldsymbol{x}) = y_i$. It samples random tape r_i for P_i. It samples all labels together with their additive shares. i.e.
$$\{\widehat{\mathsf{L}}^b_{i,j,k,h}, \widehat{\mathsf{L}}^b_{j,i,k,h}\}_{P_j\notin Z_V, k\in[n], h\in[\ell], b\in\{0,1\}}$$
2. For $P_j \notin Z_V$, Sim_S computes the simulated messages of the honest parties as
$$\{\hat{T}^1_{ji}\}_{P_i\in Z_V} = \mathsf{NM}^{1*}_j(x_j, r_j)$$
For $P_i \in Z_V$, it simply computes the next-message function as in the real protocol i.e.
$$\{T^1_{ij}\}_{j\in[n]} = \mathsf{NM}^1_i(x_i, r_i)$$
3. For each $P_i \in Z_V$, Sim_S executes $\Pi^i_{\mathsf{sh\text{-}des}}$ instances with the honest parties' inputs as $\{\hat{T}^1_{ji}, \{\widehat{\mathsf{L}}^b_{i,j,k,h}, \widehat{\mathsf{L}}^b_{j,i,k,h}\}_{k\in[n],h\in[\ell]}\}_{P_j\notin Z_V}$.
4. For each $P_i \in Z_V, P_j \notin Z_V$, Sim_S generates the simulated view as
$$\widehat{\mathsf{View}}^{*Z_V}_j \leftarrow \mathsf{Sim}^j_{\mathsf{sh\text{-}des},\mathsf{S}}(\hat{T}^1_{ij}, \{\mathsf{L}^b_{j,k,i,h}\}_{k\in[n],h\in[\ell],b\in\{0,1\}})$$
5. For each $P_i \notin Z_V$, Sim_S uses the simulated next-message functions to obtain the simulated messages $\{\hat{T}^2_{ij}\}_{P_j\in Z_V}$. It further selects the correct labels according to these simulated values. In more detail, it computes
$$\{\hat{T}^2_{ij}\}_{P_j\in Z_V} \leftarrow \mathsf{NM}^{2*}_i(\{\{\hat{T}^1_{ji}\}_{P_j\notin Z_V}, \{T^1_{ji}\}_{P_j\in Z_V}\})$$

6. For $\forall P_i \in Z_V, P_j \notin Z_V$, let $\hat{T}^2_{ji} = z_1 \| ... \| z_\ell$. It then computes $\widehat{\mathsf{L}}_{j,i} = \{\widehat{\mathsf{L}}^{z_h}_{j,i,h}\}_{h\in[\ell]}$, and includes $\widehat{\mathsf{L}}_{j,i}$ in the view of P_i.

At the end of the simulation, the adversary's view consists of

$$\left\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j\notin Z_V}\}_{P_i\in Z_V}, \{\widehat{\mathsf{View}}_k^{*Z_V}\}_{P_k\notin Z_V}\right\}_{k\in\mathbb{N},\, \boldsymbol{R}\in\mathbb{R}^n}$$

where R_i is the random tape r_i along with the label shares sampled by Sim_S. It remains to argue that the above view is statistically close to the real view of the adversary in protocol 3.1. For this, consider the following hybrids:

– Hybrid H_0. This is the distribution of the output of Sim_S,

$$\left\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j\notin Z_V}\}_{P_i\in Z_V}, \{\widehat{\mathsf{View}}_k^{*Z_V}\}_{P_k\notin Z_V}\right\}_{k\in\mathbb{N},\, \boldsymbol{R}\in\mathbb{R}^n}$$

– Hybrid H_1. An intermediate hybrid distribution, where the next-message functions are still those of Sim_sh, but each of the simulated designated-party instances with an honest designated party is replaced with an actual designated-party protocol instance:

$$\left\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j\notin Z_V}\}_{P_i\in Z_V}, \{\mathsf{View}_k^{*Z_V}\}_{P_k\notin Z_V}\right\}_{k\in\mathbb{N},\, \boldsymbol{R}\in\mathbb{R}^n}$$

Basically, each $\Pi^j_{\mathsf{sh\text{-}des}}$ instance for an honest P_j computes the simulated next-message functions NM_j^{r*}. This differs from H_0, where this $\Pi^j_{\mathsf{sh\text{-}des}}$ is simulated, and not actually executed.

– Hybrid H_2. The distribution of the view of the parties in Z_V in a real execution of Protocol 3.1:

$$\left\{\{x_i, y_i, R_i, \mathsf{View}_i^{Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j\notin Z_V}\}_{P_i\in Z_V}, \{\mathsf{View}_k^{Z_V}\}_{P_k\notin Z_V}\right\}_{k\in\mathbb{N},\, \boldsymbol{R}\in\mathbb{R}^n}$$

Claim D.2.3. *Assuming that for each $i \in [n]$, the protocol $\Pi^i_{\mathsf{sh\text{-}des}}$ in Protocol B.1.1 is secure in the presence of a semi-honest unbounded adversary that can corrupt any subset of $P \setminus \{P_i\}$, the view in the hybrid distributions H_0 and H_1 are statistically close.*

Proof. Let $(n-d)$ be the number of honest parties. Without loss of generality, we assume that the parties $P_1, ..., P_{n-d}$ are honest. In order to show that the hybrids H_0 and H_1 are statistically close, consider the following set of $n - d + 1$ hybrids.

- Hybrid $\mathsf{H}_{0,0}$. This is a hybrid that is distributed identically to H_0:

$$\left\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j\notin Z_V}\}_{P_i\in Z_V}, \{\widehat{\mathsf{View}}_k^{*Z_V}\}_{P_k\notin Z_V}\right\}_{k\in\mathbb{N},\, \boldsymbol{R}\in\mathbb{R}^n}$$

- Hybrid $\mathsf{H}_{0,j}$. For each $j \in [n-d]$, this hybrid distribution is derived using the next-message functions of the simulated virtual protocol $\mathsf{Sim}_{\mathsf{sh}}$, where for the sub-protocols upto $\Pi^{j}_{\mathsf{sh\text{-}des}}$, a view of the parties in Z_V of the real execution of the protocol is included, instead of a simulated one as in H_0. The rest of the view is generated as in H_0:

$$\Big\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j \notin Z_V}\}_{P_i \in Z_V}, \{\mathsf{View}_k^{*Z_V}\}_{k \le j} \\ , \{\widehat{\mathsf{View}}_k^{*Z_V}\}_{k \in [n-d] \setminus [j]}\Big\}_{k \in \mathbb{N},\, \boldsymbol{R} \in \mathbb{R}^n}$$

- Hybrid $\mathsf{H}_{0,n-d}$. The last hybrid distribution above is distributed the same way as in H_1:

$$\Big\{\{x_i, y_i, R_i, \mathsf{View}_i^{*Z_V}, \{\widehat{\mathsf{L}}_{j,i}\}_{P_j \notin Z_V}\}_{P_i \in Z_V}, \{\mathsf{View}_k^{*Z_V}\}_{P_k \notin Z_V}\Big\}_{k \in \mathbb{N},\, \boldsymbol{R} \in \mathbb{R}^n}$$

  In the above set of hybrids, the adjacent pairs of hybrids $\mathsf{H}_{0,j-1}$ and $\mathsf{H}_{0,j}$ differ only in the view of one execution of the sub-protocol $\Pi^{j*}_{\mathsf{sh\text{-}des}}$, where $\mathsf{NM}_i^{2^*} \in \mathsf{Sim}_{\mathsf{sh}}$. In $\mathsf{H}_{0,j-1}$, the view of this execution is the simulated view $\widehat{\mathsf{View}}_j^{*Z_V}$. In $\mathsf{H}_{0,j}$, this is $\mathsf{View}_j^{*Z_V}$ as in the real execution. The rest of the hybrid is distributed the same way.
  We already know that the simulated view produced by $\mathsf{Sim}^{j}_{\mathsf{sh\text{-}des},\mathsf{S}}$ is distributed in a way that is statistically close to the real view of the protocol $\Pi^{j*}_{\mathsf{sh\text{-}des}}$. Let ε be the statistical difference between the distributions of the simulated and real executions, respectively. Then the statistical difference between the hybrids is no more than ε. It also follows from the triangle inequality of statistical distance that the difference between the simulated view H_0 output by $\mathsf{Sim}_{\mathsf{S}}$ and the hybrid H_1 is at most $(n-d)\varepsilon$. Therefore, these hybrid distributions are statistically close.

□

Claim D.2.4. *Assuming that the protocol Π_{sh} is secure in the presence of a semi-honest unbounded adversary corrupting $t < \frac{n}{2}$, the views in the hybrid distributions H_1 and H_2 are statistically close.*

Proof. We already know that the distribution of the view produced by $\mathsf{Sim}_{\mathsf{S}}$ is statistically close to that in the real execution of the protocol Π_{sh}. Let δ be the statistical difference between these two views. Since this is the only difference between the two hybrids H_1 and H_2, the statistical difference between them is also at most δ. Therefore, these hybrid distributions are statistically close. So it follows that the view produced by $\mathsf{Sim}_{\mathsf{S}}$ is statistically close to that of the real execution of protocol 3.1 for an adversary corrupting $t < \frac{n}{2}$. □

□

D.3 Proof of 2-Round Fall-back Protocol (Theorem B.2.3)

Proof. We first show a simulator $\mathsf{Sim}_{\mathsf{S}}$ that for any subset of corruptions $t < \frac{n}{2}$ generates a view that is statistically close to the view in the real protocol. The second component will show the existence of $\mathsf{Sim}_{\mathsf{C}}$ that for any subset of $Z \in \mathcal{Z}_{\mathsf{C}} = 2^P$ generates a view computationally indistinguishable from the view in the real protocol.

Let Z ($|Z| < \frac{n}{2}$) represent the subset of parties corrupted by an unbounded adversary $\mathcal{A}$. $\mathsf{Sim}_{\mathsf{S}}$ receives the inputs of the corrupted parties as $\{x_i\}_{i \in Z}$ and the output $y = f(x_1, \ldots, x_n)$. $\mathsf{Sim}_{\mathsf{S}}$ does the following:

1. For every P_i, where $P_i \in Z$,
   - Sample randomness $\tilde{r}_i$ and $\tilde{r}'_i$.
   - Compute $\tilde{c}_i = \mathsf{Com}_{\mathsf{SB}}(x_i, \tilde{r}_i)$ and $\tilde{c}'_i = \mathsf{Com}_{\mathsf{SH}}(x_i, \tilde{r}'_i)$.
   - Generate t-out-of-n Shamir sharing of $\tilde{c}_i$ as $\{[\tilde{c}_i]^t_j\}_{j \in [n]} \leftarrow \mathsf{SS.Share}(\tilde{c}_i)$.
   - Generate additive sharing of 0 as $\{[0]^n_{i,j}\}_{j \in [n]}$.
2. For every honest $P_j \notin Z$,
   - Sample randomness $\tilde{r}_j$ and compute $\tilde{c}_j := \mathsf{Com}_{\mathsf{SB}}(0; \tilde{r}_j)$
   - Generate t-out-of-n Shamir sharing of $\tilde{c}_j$ as $\{[\tilde{c}_j]^t_k\}_{k \in [n]} \leftarrow \mathsf{SS.Share}(\tilde{c}_j)$.
   - Compute $\tilde{c}'_j := \mathsf{Com}_{\mathsf{SH}}(0; \tilde{r}'_j)$
   - Compute the additive shares of 0 as $\{[0]^n_{j,k}\}_{k \in [n]}$.
3. Obtain $(\mathsf{pp}, \hat{y}, \hat{y}_H = \sum_{P_j \notin Z} \hat{X}_j) \leftarrow \mathsf{Sim}_{\mathsf{RARE}}(1^\kappa, \mathcal{P} \setminus Z, y)$.
4. $\mathsf{Sim}_{\mathsf{S}}$ emulates the $f_{H,\mathbf{x}}$ oracle as follows: If $\mathsf{Sim}_{\mathsf{RARE}}$ queries $\{x_i\}_{P_i \in Z}$ output y else output $\perp$.
5. For all $k \in [n]$, compute $[0]^n_k = \sum_{j=1}^n [0]^n_{j,k}$ and $[0]_H = \sum_{P_j \notin Z} [0]^n_j$.
6. Compute a $n - |Z|$ additive sharing of $\hat{y}_H + [0]_H$ as $\{\hat{X}'_j\}_{P_j \notin Z}$.
7. Output the view of the adversary as

$$(\{\{[\tilde{c}_j]^t_i, [0]^n_{j,i}\}_{P_i \in Z,}, \tilde{c}'_j\}_{j \in [n]}, \{\tilde{r}_i, \tilde{r}'_i\}_{P_i \in Z}, \mathsf{pp}, \{\hat{X}'_j\}_{P_j \notin Z})$$

We now prove that the simulated view is statistically indistinguishable from the real view. Let hybrid H_0 denote the simulated view generated by $\mathsf{Sim}_{\mathsf{S}}$. Hybrid H_1 is obtained by replacing the statistically binding commitments of 0 with commitments on the actual inputs x_j for every $P_j \notin Z$. Since $\mathcal{A}$ corrupts at most $t < \frac{n}{2}$ parties, it observes at most t shares, and hence, the actual commitment, and therefore, the input remain perfectly hidden. This implies that the hybrids H_0 and H_1 are identically distributed. We now define hybrid H_2 where the statistically hiding commitments on 0 are replaced with commitments on the actual inputs x_j for every $P_j \notin Z$. The statistical indistinguishability of these two hybrids follows from the statistical hiding property of the commitment scheme.

Next, we define hybrid H_3, where $\mathsf{Sim}_{\mathsf{RARE}}$ gets oracle access to the actual residual oracle $f_{H,\boldsymbol{x}}$. We now prove that outputs provided by the oracle $f_{H,\boldsymbol{x}}$ and those provided by $\mathsf{Sim}_{\mathsf{S}}$ in hybrid H_2 are statistically indistinguishable. To see this, we show that $f_{H,\boldsymbol{x}}$ outputs $\perp$ if the adversary queries anything other than the committed inputs $\{x_i\}_{P_i \in Z}$. Since the statistically binding commitment is t-out-of-n Shamir shared, the honest shares completely determine this commitment.

Thus, each corrupt party is bound to the input it commits to, and hence, any input other than $\{x_i\}_{P_i \in Z}$ returns $\perp$.

Next, we define hybrid H_4 by replacing the simulated encodings $\{\hat{X}'_j\}_{P_j \notin Z}$ with the real encodings $\{\hat{X}_j\}_{P_j \notin Z}$. By the statistical robustness of the underlying RARE scheme, we get $\mathsf{H}_3 \overset{\text{s}}{\equiv} \mathsf{H}_4$. Since H_4 is identical to the real view, this completes the proof of the statistical case.

The simulator Sim_C operates identically to the simulator Sim_S described in the statistical case. Let H_0 denote the simulated execution generated by Sim_C. Again, hybrid H_1 is obtained by replacing $\tilde{c}_j = \mathsf{Com}_{\mathsf{SB}}(0; \tilde{r}_j)$ with $c_j = \mathsf{Com}_{\mathsf{SB}}(x_j; r'_j)$ for every honest P_j. From the computational hiding of $\mathsf{Com}_{\mathsf{SB}}$, it follows that $\mathsf{H}_0 \overset{\text{c}}{\equiv} \mathsf{H}_1$. As before, hybrid H_2 is obtained by replacing the statistically hiding commitments on 0 with the actual input x_j for every honest P_j. From the statistically hiding property, it follows that $\mathsf{H}_1 \overset{\text{s}}{\equiv} \mathsf{H}_2$.

Next, we define hybrid H_3, where $\mathsf{Sim}_{\mathsf{RARE}}$ gets oracle access to the actual residual oracle $f_{H,\boldsymbol{x}}$. We now prove that outputs provided by the oracle $f_{H,\boldsymbol{x}}$ and those provided by Sim_S in hybrid H_2 are computationally indistinguishable. To see this, we show that $f_{H,\boldsymbol{x}}$ outputs $\perp$ if the adversary queries anything other than the committed inputs $\{x_i\}_{P_i \in Z}$. Recall that each party broadcasts a statistically hiding commitment on its input, and this commitment is then taken from all parties as input. Thus, the oracle $f_{H,\boldsymbol{x}}$ outputs a non-$\perp$ value on inputs $\{x'_i\}_{P_i \in Z}$ only if each P_i provides valid openings corresponding to the input x'_i. The computational binding property of these commitment schemes therefore implies that the adversary on any query other than $\{x_i\}_{P_i \in Z}$, receives $\perp$ except with negligible probability. Thus, the hybrids $\mathsf{H}_2 \overset{\text{c}}{\equiv} \mathsf{H}_3$.

We define hybrid H_4 by replacing the simulated encodings $\{\hat{X}'_j\}_{P_j \notin Z}$ with the real encodings $\{\hat{X}_j\}_{P_j \notin Z}$. By the statistical robustness of the underlying RARE scheme, $\mathsf{H}_3 \overset{\text{s}}{\equiv} \mathsf{H}_4$. Therefore, it follows that $\mathsf{H}_0 \overset{\text{c}}{\equiv} \mathsf{H}_4$, which proves the security of the protocol.

□

E Deferred Malicious Proofs

E.1 Proof of Malicious Designated Protocol (Theorem C.1.2)

Proof. **Information-Theoretic to Designated Party.** First, we show that there exists a PPT simulator Sim_S that, for any unbounded adversary $\mathcal{A}$ controlling a subset of parties $Z \in \mathcal{Z}_\mathsf{D}$, can output a view whose distribution is statistically close to the view in the real protocol. Let H denote the set of honest parties. We consider the designated party, i.e. P_1 to be honest, since the protocol does not give any guarantees in case of the designated party being unbounded corrupt. The PPT simulator Sim_S operates as follows:

- **Generation of the CRS:** Sim_S generates the CRS as follows:
  1. For each $P_j \notin Z, t \in A_j, \alpha, \beta \in \{0,1\}$, generate the CRS $\sigma_{t,\alpha,\beta}$.

2. For each $P_i \in Z, \alpha, \beta \in \{0,1\}$ and $t \in A_i$ generate $(\sigma_{t,\alpha,\beta}, \tau_{t,\alpha,\beta}) \leftarrow \mathsf{Ext}_1(1^\kappa)$ (using the extractor of the OT protocol).
3. Output the CRS as $\{\sigma_{t,\alpha,\beta}\}_{t,\alpha,\beta}$.

– For each t where $\Phi_t = (j, f, g, h)$ and $P_j \notin Z$, Sim_S generates

$$\hat{\mathsf{ots}}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\kappa, 0; \hat{\omega}_{t,\alpha,\beta})$$

where $\hat{\omega}_{t,\alpha,\beta}$ is randomly sampled.
– Sim_S invokes Sim_Φ, that for each t s.t. $\Phi_t = (j, f, g, h)$ and $P_j \notin Z$, outputs $\hat{z}_j$.
– Sim_S sends $(\hat{z}_j, \{\hat{\mathsf{ots}}_{1,t,\alpha,\beta}\}_{t\in A_j,\alpha,\beta\in\{0,1\}})$ to $\mathcal{A}$ on behalf of honest P_j.
– Sim_S receives $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ for all $P_i \in Z$. For each $t \in A_i, \alpha, \beta \in \{0,1\}$, it extracts $b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta})$. The extracted input $b_{t,\alpha,\beta}$ is provided by Sim_S to Sim_Φ [10].
– At some point in the computation, Sim_Φ returns the extracted inputs $\{x_i\}_{P_i\in Z}$ of the corrupted parties. Sim_S forwards these extracted inputs to the ideal functionality.

As proved in the case of a semi-honest adversary, the view of the adversary in the ideal world and the real execution are statistically indistinguishable. The only messages that $\mathcal{A}$ receives from an honest party P_j are $(z_j, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$. Since the conforming protocol Φ is fall-back secure, it follows that z_j is statistically indistinguishable from $\hat{z}_j$. Also, since the OT scheme provides IT security to the receiver, $\mathsf{ots}_{1,t,\alpha,\beta}$ statistically hides the choice bit. Thus, the two views are statistically close.

Computational Security. It remains to handle the case of a computational adversary. The computational security follows from the simulator provided by [GS18]. For completeness, we provide the proof below. The only change is the designated party, which we assume without loss of generality as P_1 gets the output, while the other parties receive $\perp$. We only need to consider the case when P_1 is part of the corrupted parties, otherwise the security will follow from the statistical case. We now consider the case of a computational adversary corrupting any arbitrary subset of parties. Let $\mathcal{A} \subseteq [n]$ be a malicious adversary corrupting a subset of parties $Z \subset P$. We provide the description of the ideal world adversary Sim_C that simulates the view of the real world adversary $\mathcal{A}$. The simulator will internally use the malicious simulator Sim_Φ for Φ, the extractor $(\mathsf{Ext}_1, \mathsf{Ext}_2)$ implied by the sender security of the 2-round OT, the simulator $\mathsf{Sim}_{\mathsf{Eq}}$, which is obtained from equivocal receiver security, and the simulator $\mathsf{Sim}_{\mathsf{GC}}$ for the garbling scheme for circuits.

– Faithful Execution: Faithful execution is an execution that is consistent with these extracted values. In the first round of the protocol $\mathcal{A}$ provides z_i

[10] As in the computational case, a Faithful execution is done for Sim_Φ to complete its simulation. Since Φ is fall-back secure, Sim_Φ generates a statistically indistinguishable transcript.

for every $P_i \in Z$ and $\mathsf{ots}_{1,t,\alpha,\beta}$ for every $t \in \cup_{i\in[n]\setminus h}$ and $\alpha, \beta \in \{0,1\}$. These values can be extracted by an Ext_2. We define an interactive procedure $\mathsf{Faithful}\big(i, \{z_i\}_{i\in[n]}, \{b_{t,\alpha,\beta}\}_{t\in A_{i,\alpha,\beta}}\big)$ that on input $i \in [n]$, $\{z_i\}_{i\in[n]}$, $\{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ produces protocol Φ message on behalf of party P_i (acting consistently with the extracted values) as follows:

1. Set $\mathsf{st}^* := z_1 \| \ldots \| z_n$.
2. For $t \in \{1, \ldots, T\}$:
   (a) Parse $\Phi_t = (i^*, f, g, h)$.
   (b) If $i \neq i^*$ then it waits for a bit from P_i and sets st^*_h to be the received bit once it is received.
   (c) Set $\mathsf{st}^* := b_{t,\mathsf{st}^*_f,\mathsf{st}^*_g}$ and output it to all the other parties.

– Generating CRS: Simulator Sim_C generates the common reference string as follows:

1. For each $P_i \notin Z$, $t \in A_i$, $\alpha, \beta \in \{0,1\}$ set
$$(\sigma_{t,\alpha,\beta}, (\mathsf{ots}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{\mathsf{Eq}}(1^\kappa)$$
with the equivocal simulator.
2. Similarly for each $P_i \in Z$, $\alpha, \beta \in \{0,1\}$ and $t \in A_i$, generate
$$(\sigma_{t,\alpha,\beta}, \tau_{t,\alpha,\beta}) \leftarrow \mathsf{Ext}_1(1^\kappa)$$
using the extractor of the OT protocol.
3. Output the CRS $\{\sigma_{t,\alpha,\beta}\}_{t,\alpha,\beta}$.

– Initialization: Sim_C executes the simulator $\mathsf{Sim}_\Phi(1^\kappa)$ to obtain $\{z_i\}_{P_i\notin Z}$. We describe how Sim_C provides its messages to Sim_Φ and $\mathcal{A}$.

– Round-1 messages from Sim_C to $\mathcal{A}$: For each $P_i \notin Z$, Sim_C sends to the adversary $\mathcal{A}$ $\big(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}\big)$ on behalf of the honest party P_i.

– Round-1 messages from $\mathcal{A}$ to Sim_C: Corresponding to every $P_i \in Z$, Sim_C receives from the adversary $\mathcal{A}$ the value $\big(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}\big)$ on behalf of the corrupted party P_i. Next, for each $P_i \in Z$, $t \in A_i$, and $\alpha, \beta \in \{0,1\}$ extract the value $b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta})$.

– For every corrupted party $P_i \in Z$, Sim_C sends z_i values to Sim_Φ. This begins the computation phase of Φ by Sim_Φ. For every corrupted party $P_i \in Z$ Sim_C generates messages for P_i for Sim_Φ using the procedure $\mathsf{Faithful}\big(i, \{z_i\}_{i\in[n]}, \{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta}\big)$. Sim_Φ will eventually return the extracted inputs $\{x_i\}_{P_i\in Z}$ of the corrupted parties. For each $P_i \in Z$, Sim_C sends $(\mathsf{input}, \{P_1, \ldots, P_n\}, P_i, x_i)$ to the ideal functionality implementing f and P_i obtains the output y if $P_i = P_1$, else obtains $\perp$, which is provided to Sim_Φ. Let $\mathcal{T} \in \{0,1\}^t$ where $\mathcal{T}_t$ is the bit sent in the t-th round of the computation phase of Φ in this execution. Let st^* be the state value at the end of the execution of one of the corrupted parties. Also, for each $t \in \cup_{P_i\notin Z} A_i$ and $\alpha, \beta \in \{0,1\}$ set $\omega_{t,\alpha,\beta} := \omega^{\mathcal{T}_t}_{t,\alpha,\beta}$

– Round-2 messages from Sim_C to $\mathcal{A}$: For each $P_i \notin Z$, the simulator Sim_C generates the second round message on behalf of party P_i as follows:

1. For each $k \in [\ell]$ set $\mathsf{L}^{i,T+1}_k := 0^\kappa$.

2. For each t from T down to 1,
   (a) Parse Φ_t as (i^*, f, g, h).
   (b) Set $\alpha^* := \mathsf{st}_f^*$, $\beta^* := \mathsf{st}_g^*$, and $\gamma^* := \mathsf{st}_h^*$.
   (c) If $i = i^*$ then compute

$$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa, ((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{L}_k^{i,t+1}\}_{k\in[\ell]})\Big).$$

   (d) If $i \neq i^*$ then set

$$\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}\mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{L}_h^{i,t+1}, \mathsf{L}_h^{i,t+1})$$

   and compute

$$(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa, (\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{L}_k^{i,t+1}\}_{k\in[\ell]\setminus\{h\}})\Big).$$

3. Send $(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{L}_k^{i,1}\}_{k\in[\ell]})$ to party P_1.

– Round-2 messages from $\mathcal{A}$ to $\mathsf{Sim}_{\mathsf{C}}$: For every $P_i \in Z$, $\mathsf{Sim}_{\mathsf{C}}$ obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, $\mathsf{Sim}_{\mathsf{C}}$ executes the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties to see that the execution of garbled circuits proceeds consistently with the expected faithful execution. If the computation succeeds then for each $P_i \notin Z$, $\mathsf{Sim}_{\mathsf{C}}$ sends $(\mathsf{generateOutput}, \{P_1 \dots P_n\}, P_i)$ to the ideal functionality.

No adversary $\mathcal{A}$ can distinguish whether it interacts with a real world adversary $\mathcal{A}$ or an ideal world adversary $\mathsf{Sim}_{\mathsf{C}}$. The proof proceeds via a hybrid argument. We prove this via a hybrid argument with $T + 3$ hybrids.

– Hybrid H_0. This hybrid corresponds to the real world execution.
– Hybrid H_1. In this hybrid, we modify the generation of the common random string for the corrupted parties. Instead of sampling it honestly, the simulator generates it using the OT extractor.
  For every $P_i \in Z$, $\alpha, \beta \in \{0, 1\}$ and $t \in A_i$ the simulator samples

$$(\sigma_{t,\alpha,\beta}, \tau_{t,\alpha,\beta}) \leftarrow \mathsf{Ext}_1(1^\kappa)$$

  When the adversary sends the first round of OT messages

$$(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t,\alpha,\beta})$$

  on behalf of party P_i, the simulator extracts

$$b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta})$$

  Indistinguishability between H_0 and H_1 follows from the sender security and equivocal receiver security of the 2-round OT protocol.

– Hybrid H_t for $t \in \{1, \ldots, T+1\}$. Let $\Phi_t = (i^*, f, g, h)$ denote the t-th action of protocol Φ. The simulator internally executes Φ using honest party inputs and randomness and computes the resulting local state st^*. Let $\alpha^* := \mathsf{st}^*_f, \beta^* := \mathsf{st}^*_g, \gamma^* := \mathsf{st}^*_h$.
Hybrid H_t differs from H_{t-1} in three ways. For each $t > 1$, the hybrid H_t differs from H_{t-1} only in the way messages related to the t-th action $\Phi_t = (i^*, f, g, h)$ are generated. In particular, we modify the distributions of the OT transcripts (from both rounds) and the garbled circuits corresponding to this step.
Fix any $P_i \in Z$. In the mental experiment, the simulator $\mathsf{Sim}_{\mathsf{C}}$ completes an execution of Φ using honest inputs and randomness for honest parties, while simulating corrupted parties. It samples $\{z_i\}_{P_i \notin Z}$ and generates all protocol messages accordingly. Let $\alpha^* := \mathsf{st}^*_f, \quad \beta^* := \mathsf{st}^*_g, \quad \gamma^* := \mathsf{st}^*_h$ denote the local states at the end of the execution. We now describe the changes introduced in H_t:
  - If $P_{i^*} \in Z$, no modification is applied.
  - Otherwise, $\mathsf{Sim}_{\mathsf{C}}$ alters the generation of messages for P_{i^*} as follows:
    1. For each $\alpha, \beta \in \{0, 1\}$, compute
    $$(\sigma_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta}, \omega^1_{t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}) \leftarrow \mathsf{Sim}_{\mathsf{Eq}}(1^\kappa)$$
    and set $\omega_{t,\alpha^*,\beta^*}$ to $\omega^{\mathcal{T}_t}_{t,\alpha^*,\beta^*}$.
    2. The garbled circuit is generated via
    $$(\widetilde{\mathsf{P}}^{i^*,t}, \{\mathsf{L}^{t+1}_k\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\big(1^\kappa, (\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{L}^{i^*,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}\big),$$
    where $\{\mathsf{L}^{t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ are honestly generated input labels.
  - For every other honest party P_i with $i \neq i^*$, we modify:
    1. Instead of generating all four OT messages, only a single value is produced:
    $$\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2\big(\sigma_{t,\alpha^*,\beta^*}, \mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{L}^{i,t+1}_{h,\mathcal{T}}, \mathsf{L}^{i,t+1}_{h,\mathcal{T}}\big)$$
    2. The corresponding garbled circuit is sampled as
    $$(\widetilde{\mathsf{P}}^{i^*,t}, \{\mathsf{L}^{i,t}_k\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\big(1^\kappa, \mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{L}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]\setminus\{h\}}\big)$$
    where $\{\mathsf{L}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i^*,t+1}$.
– Hybrid H'_{T+1}. In this hybrid, the output phase is modified, the garbled circuits provided by adversary $\mathcal{A}$ on behalf of corrupt parties are evaluated, and consistency with the transcript $\mathcal{T}$ is checked. If all checks pass, we instruct parties $\notin Z$ to output y. else, they output $\perp$. This transition relies on the authenticity of garbled inputs.
– Hybrid H_{T+2}. In this hybrid, we just change how the transcript $\mathcal{T}$, random coins of malicious parties and the value of st^* are generated. Instead of generating it using honest party inputs in execution with a faithful execution

of Φ, we generate it via the simulator $\mathsf{Sim}_{\mathsf{\Phi}}$. In other words we execute the simulator $\mathsf{Sim}_{\mathsf{\Phi}}$ where messages on behalf of each corrupted party P_i are generated using $\mathsf{Faithful}(i, \{z_i\}_{P_i \in Z}, \{b_{t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta}$. The indistinguishability between H_{T+1} and H_{T+2} follows directly from the malicious security of the protocol Φ. The hybrid H_{T+2} is the same as ideal execution.

Claim E.1.1. *Assuming malicious security of the 2-round OT protocol and the security of the garbling scheme, for every $t \in \{2, \ldots, T+1\}$, the hybrids H_{t-1} and H_t are computationally indistinguishable.*

Proof. We follow the same notation as earlier. Let $\mathsf{\Phi}_t = (i^*, f, g, h)$, and denote by st_{i^*} the internal state of party P_{i^*} after round t. Define

$$\alpha^* := \mathsf{st}_{i^*,f} \oplus v_{i^*,f}, \quad \beta^* := \mathsf{st}_{i^*,g} \oplus v_{i^*,g}, \quad \gamma^* := \mathsf{st}_{i^*,h} \oplus v_{i^*,h}.$$

We prove the claim through a sequence of intermediate hybrids $\mathsf{H}_{t,1}, \mathsf{H}_{t,2}, \mathsf{H}_{t,3}$.

- Hybrid $\mathsf{H}_{t,1}$: This hybrid modifies H_t by letting the simulator $\mathsf{Sim}_{\mathsf{C}}$ generate garbled circuits for all $P_i \notin Z$ using the garbling simulator, instead of constructing them honestly.
  Concretely, rather than sampling $\big(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k \in [\ell]}\big)$
  via the real garbling algorithm, they are produced using $\mathsf{Sim}_{\mathsf{GC}}$ so that the circuit output is embedded.
  We distinguish two cases:
  - If $i = i^*$, then

    $$\big(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k \in [\ell]}\big) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa,\ \big((\alpha^*, \beta^*, \gamma^*),\ w_{t,\alpha^*,\beta^*},\ \{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k \in [\ell]}\big)\Big).$$

  - If $i \neq i^*$, then

    $$\big(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{L}_k^{i,t}\}_{k \in [\ell]}\big) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\Big(1^\kappa,\ \big(\mathsf{ots}_{2,t,\alpha^*,\beta^*}^i,\ \{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k \in [\ell] \setminus \{h\}}\big)\Big).$$

  In both cases, $\{\mathsf{L}_{k,\mathsf{st}_{i,k}}^{i,t+1}\}_{k \in [\ell]}$ are honestly generated labels for the next circuit $\widetilde{P}_i^{t+1}$.
  The transition from H_{t-1} to $\mathsf{H}_{t,1}$ is justified by repeatedly applying the security of the garbling scheme (once per $P_i \notin Z$).
- Hybrid $\mathsf{H}_{t,2}$: If all $i \neq i^*$ belongs to Z, this step is not needed. Otherwise, we modify how OT outputs are generated for each honest party $i \notin Z \setminus \{i^*\}$.
  Instead of preparing all four values $\mathsf{ots}_{2,t,\alpha,\beta}^i$ for $(\alpha, \beta) \in \{0,1\}^2$, the simulator only generates one of the four values namely $\mathsf{ots}_{2,t,\alpha^*,\beta^*}^i$ and which is now generated as $\mathsf{OT}_2\big(\sigma_{t,\alpha^*,\beta^*},\ \mathsf{ots}_{1,t,\alpha^*,\beta^*},\ \mathsf{L}_{h,\mathcal{T}}^{i,t+1},\ \mathsf{L}_{h,\mathcal{T}}^{i,t+1}\big)$.
  The indistinguishability between $\mathsf{H}_{t,1}$ and $\mathsf{H}_{t,2}$ follows from the sender security of the malicious OT protocol.

– Hybrid $\mathsf{H}_{t,3}$: This step is only relevant if $P_{i^*} \notin Z$. Here we adjust how the simulator generates the round-1 message for party P_{i^*}.
Instead of fixing these values directly, the simulator invokes an equivocal simulator:

$$(\sigma_{t,\alpha,\beta}, (\mathsf{ots}_{1,t,\alpha,\beta}, , (\mathsf{ots}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{\mathsf{Eq}}(1^\kappa).$$

and then sets $\omega_{t,\alpha^*,\beta^*} := \omega^{\mathcal{T}_t}_{t,\alpha^*,\beta^*}$ rather than $\omega^{v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta)}_{t,\alpha,\beta}$.

We now argue indistinguishability between $\mathsf{H}_{t,2}$ and $\mathsf{H}_{t,3}$ using the equivocal receiver security. We interact with the equivocal security challenger four times. For each α, β, we obtain $\sigma_{t,\alpha,\beta}$, $\mathsf{ots}_{1,t,\alpha,\beta}$ and $\omega^{v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta)}_{t,\alpha,\beta}$. We use this to generate the first round message of the protocol and the second round messages of the protocol. Note that if these values were generated honestly, then the distribution produced is identical to $\mathsf{H}_{t,2}$. Else, it is distributed identically to $\mathsf{H}_{t,3}$. We observe that $\mathsf{H}_{t,3}$ is same as H_t. Each transition thus preserves computational indistinguishability, which concludes the argument. □

□

E.2 Proof of Malicious UA Protocol (Theorem 4.1.7)

Proof. For brevity, we denote a party P_i's label and randomness share together as $X^b_{x,i} = (\mathsf{L}^b_{x,i}, R^b_{x,i})$.

Statistical Security. Denote the unbounded adversary as $\mathcal{A}$, the set of corrupt parties as Z, the simulator of the designated protocol as $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{S}}$, and the simulator of the virtual protocol as $\mathsf{Sim}_{\mathsf{V}}$. Then, the simulator for protocol 4.1.3 $\mathsf{Sim}_{\mathsf{S}}$ does the following:

1. For all $P_j \notin Z$, $\mathsf{Sim}_{\mathsf{S}}$ randomly samples the additive shares of the labels and randomness as

   $$\hat{\Sigma}_j = \{\mathsf{L}^b_{i,k,j,h}, \mathsf{L}^b_{i,\mathsf{BC},j,h,r}, R^b_{i,k,j,h}, R^b_{i,\mathsf{BC},k,j,h,r}\}_{i,k \in [n], h \in [\ell], b \in \{0,1\}, r \in [2]}$$

   It also samples the additive shares of random tapes as $r_{i,j}$ for $P_i \in Z$ along with the MAC keys $\{\mathsf{K}^b_{k,\mathsf{BC},i,j,h,r}\}_{i,k \in [n], h \in [\ell], b \in \{0,1\}}$ and participates in the designated protocol instances $\Pi^i_{\mathsf{m\text{-}des}}$.
2. $\mathsf{Sim}_{\mathsf{S}}$ computes and broadcasts commitments on $\hat{\Sigma}_j$ and $\{r_{k,j}\}_{k \neq j}$ for all $P_j \notin Z$. It computes $\mathsf{zk}_{j,i,1}$ as in the real protocol with respect to these commitments, and sends them to $P_i \in Z$. It also receives $\mathsf{zk}_{i,j,1}$ from $\mathcal{A}$.
3. For all designated instances $\Pi^j_{\mathsf{m\text{-}des}}$ where $P_j \notin Z$, $\mathsf{Sim}_{\mathsf{S}}$ invokes $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{S}}$ to generate the view of the corrupt parties $P_i \in Z$, and extract the corrupt parties' additive shares. In more detail, $\mathsf{Sim}_{\mathsf{S}}$ extracts the additive shares of the labels and randomness as

   $$\{\{\mathsf{L}^b_{k,\mathsf{BC},i,h,r}, R^b_{j,\mathsf{BC},k,i,h,r}, R^b_{k,\mathsf{BC},j,i,h,r}\}_{k \in [n], h \in [\ell], b \in \{0,1\}, r \in [2]}$$
   $$\{\mathsf{L}^b_{k,j,i,h}, \mathsf{L}^b_{j,k,i,h}, R^b_{k,j,i,h}, R^b_{j,k,i,h}\}_{P_k \in Z, h \in [\ell], b \in \{0,1\}}$$

   $\mathsf{Sim}_{\mathsf{S}}$ also extracts the MAC keys as $\{\mathsf{K}^b_{k,\mathsf{BC},j,i,h,r}\}_{k \in [n], h \in [\ell], b \in \{0,1\}, r \in [2]}$.

4. For $P_i \in Z, P_j \notin Z$, $\mathsf{Sim_S}$ receives the $\mathsf{zk}_{j,i,2}$ from $\mathcal{A}$, and sends $\mathsf{zk}_{i,j,2}$ to $\mathcal{A}$.
5. For $P_i \in Z, P_j \notin Z$, using the shares extracted in step 3 and sampled in step 1, $\mathsf{Sim_S}$ computes the pairwise labels and randomness $\{(\mathsf{L}^0_{i,j,h}, \mathsf{L}^1_{i,j,h}), (R^0_{i,j,h}, R^1_{i,j,h})\}_{h\in[\ell]}$, the corrupt parties' broadcast labels and randomness $\{(\mathsf{L}^0_{i,\mathsf{BC},h,1}, \mathsf{L}^1_{i,\mathsf{BC},h,1}), (R^0_{i,\mathsf{BC},j,h,1}, R^1_{i,\mathsf{BC},j,h,1})\}_{h\in[\ell]}$ and the honest parties' broadcast labels and randomness as $\{(\mathsf{L}^0_{j,\mathsf{BC},h,1}, \mathsf{L}^1_{j,\mathsf{BC},h,1}), (\boldsymbol{R}^0_{j,\mathsf{BC},h,1}, \boldsymbol{R}^1_{j,\mathsf{BC},h,1})\}_{h\in[\ell]}$. To simplify notation, we denote $X^b_{i,j,h} = (\mathsf{L}^b_{i,j,h}, R^b_{i,j,h})$ and $X^b_{i,\mathsf{BC},h,r} = (\mathsf{L}^b_{i,\mathsf{BC},h,r}, \boldsymbol{R}^b_{i,\mathsf{BC},h,r})$.
6. For $P_i \in Z, P_j \notin Z$, $\mathsf{Sim_S}$ calls $\mathsf{Sim_V}$ to receive simulated pairwise messages $\hat{T}_{ji}$ and broadcast messages $\hat{\mathsf{b}}^1_j$. It selects the labels and randomness according to $\hat{T}_{ji}, \hat{\mathsf{b}}^1_j$ and sends these to $\mathcal{A}$. More formally, it sends $\{X^{\hat{T}_{ji}[h]}_{j,i,h}, X^{\hat{\mathsf{b}}^1_j[h]}_{j,\mathsf{BC},h,1}\}_{h\in[\ell]}$ to $\mathcal{A}$, where $x[h]$ denotes the h-th bit of x. $\mathsf{Sim_S}$ also computes encrypted tags $\{\hat{w}_{k,\mathsf{BC},j,h,1}\}_{k\in[n]}$ corresponding to the broadcast messages as in the protocol, and sends them to $\mathcal{A}$. It also sends $\mathsf{zk}_{j,i,3}$ to $\mathcal{A}$.
7. For all $P_i \in Z, P_j \notin Z, h \in [\ell]$, $\mathsf{Sim_S}$ receives the labels, randomness and encrypted tags from the adversary. The pairwise labels along with the corresponding randomness are denoted as $\tilde{X}_{i,j,h}$, the broadcast labels along with the corresponding randomness are denoted as $\tilde{X}_{i,\mathsf{BC},h,1}$, and the encrypted tags are denoted as $\{\tilde{w}_{k,\mathsf{BC},i,h,1}\}_{k\in[n]}$. $\mathsf{Sim_S}$ performs the following checks:
   - Check if $\tilde{X}_{i,j,h} \in \{X^0_{i,j,h}, X^1_{i,j,h}\}$
   - Check if $\tilde{\mathsf{L}}_{i,\mathsf{BC},h,1} = \mathsf{L}^b_{i,\mathsf{BC},h,1}$ and if $\tilde{R}_{i,\mathsf{BC},j,h,1} = R^b_{i,\mathsf{BC},j,h,1}$ for some $b \in \{0,1\}$.
   - For all $k \in [n]$, $\mathsf{Sim_S}$ uses the encrypted tags $\{\tilde{w}_{k,\mathsf{BC},i,h,1}\}_{k\in[n]}$ and performs the multi-key MAC check as in Round 3 of the protocol.

   If any of the above checks fail, $\mathsf{Sim_S}$ sends abort to $\mathcal{F}_{\mathsf{ua}}$ and $\mathcal{A}$.
8. For all $P_i \in Z, P_j \notin Z$, $\mathsf{Sim_S}$ sends T_{ij} to $\mathsf{Sim_V}$, where for all $h \in [\ell]$, $T_{ij}[h] = b$ such that $\tilde{X}^{T_{ij}[h]}_{i,j,h} = X^b_{i,j,h}$. It similarly sends b^1_i to $\mathsf{Sim_V}$, where for all $h \in [\ell]$, $\mathsf{b}^1_i[h] = c$ such that $\tilde{X}^{\mathsf{b}^1_i[h]}_{i,\mathsf{BC},h,1} = X^c_{i,\mathsf{BC},h,1}$.
9. For all $P_i \in Z, P_j \notin Z$, $\mathsf{Sim_S}$ receives $\mathsf{zk}_{i,j,3}$. As explained in step 3, Sim_S extracts the entire input of each $P_i \in Z$ in the designated-party protocol instances $\{\Pi^j_{\mathsf{m\text{-}des}}\}_{P_j\notin Z}$. If any $P_i \in Z$ has used inconsistent label or randomness shares, $\mathsf{Sim_S}$ sends abort to $\mathcal{A}$ and the functionality $\mathcal{F}_{\mathsf{ua}}$.
10. For $P_i \in Z$, $\mathsf{Sim_S}$ computes the broadcast labels and randomness $\{(\mathsf{L}^b_{i,\mathsf{BC},h,2}, \{R^b_{i,\mathsf{BC},j,h,2}\}_{P_j\notin Z})\}_{h\in[\ell],b\in\{0,1\}}$ from the shares sampled in step 1 and those extracted from the corrupt parties in step 3 respectively. It similarly computes the broadcast labels and randomness $\{(X^0_{j,\mathsf{BC},h,2}, X^1_{j,\mathsf{BC},h,2})\}_{h\in[\ell]}$ for $P_j \notin Z$.
11. For $P_j \notin Z$, $\mathsf{Sim_S}$ calls $\mathsf{Sim_V}$ to receive simulated broadcast messages $\hat{\mathsf{b}}^2_j$. It selects the labels according to the simulated messages and sends these to $\mathcal{A}$. More formally, it sends $\{X^{\hat{\mathsf{b}}^2_j[h]}_{j,\mathsf{BC},h,2}\}_{h\in[\ell]}$ to $\mathcal{A}$, where $x[h]$ denotes the h-th bit of x. It also computes and sends the encrypted tags $\{\hat{w}_{k,\mathsf{BC},j,h,2}\}_{k\in[n]}$ corresponding to the broadcast messages and sends them to $\mathcal{A}$.

12. For all $P_i \in Z$, $\mathsf{Sim_S}$ receives the labels and the randomness corresponding to the messages b_i^2 from $\mathcal{A}$, denoted as $\tilde{X}_{i,\mathsf{BC},h,2}^{\mathsf{b}_i^2[h]}$. It also receives the encrypted tags $\{\tilde{w}_{k,\mathsf{BC},i,h,2}\}_{k\in[n]}$. It performs the following checks:
    - Check if $\tilde{L}_{i,\mathsf{BC},h,2} = L_{i,\mathsf{BC},h,2}^b$ and if $\tilde{R}_{i,\mathsf{BC},j,h,2} = R_{i,\mathsf{BC},j,h,2}^b$ for some $b \in \{0,1\}$ and all $P_j \notin Z$.
    - $\mathsf{Sim_S}$ uses the encrypted tags $\{\tilde{w}_{k,\mathsf{BC},i,h,2}\}$ and performs the multi-key MAC checks as in Round 4 of the protocol.
13. For all $P_i \in Z$, $\mathsf{Sim_S}$ sends b_i^2 to $\mathsf{Sim_V}$, where for all $h \in [\ell]$, $\mathsf{b}_i^2[h] = c$ such that $\tilde{\mathsf{L}}_{i,\mathsf{BC},h,2}^{\mathsf{b}_i^2[h]} = \mathsf{L}_{i,\mathsf{BC},h,2}^c$.
14. At some point in the above interaction with $\mathsf{Sim_V}$, $\mathsf{Sim_V}$ extracts the inputs $\{x_i\}_{P_i\in Z}$ and sends them to $\mathsf{Sim_S}$. $\mathsf{Sim_S}$ forwards these inputs to the functionality $\mathcal{F}_{\mathsf{ua}}$ and gets the output y, which is forwarded to $\mathsf{Sim_V}$ in order for it to complete the simulation.

We now prove that the simulated view of the adversary is statistically close to the real view of the adversary in the protocol. We introduce intermediate hybrids $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ that are used to prove the indistinguishability, and denote the real view and the simulated view as hybrid H_0 and H_4 respectively. We start by defining H_1, which differs from H_0 in that all the designated-party protocol instances $\{\Pi_{\mathsf{m\text{-}des}}^j\}_{P_j\notin Z}$ with honest designated parties are replaced with their corresponding simulated views generated by $\mathsf{Sim}_{\mathsf{m\text{-}des,S}}$. Denoting the number of corrupt parties as d, we introduce $n-d+1$ hybrids as $\mathsf{H}_{0,0},\ldots,\mathsf{H}_{0,n-d}$. Without loss of generality, we assume that parties $P_1,\ldots P_{n-d}$ are honest. In hybrid $\mathsf{H}_{0,k}$ for $k \in [n-d]\cup\{0\}$, the first k honest parties' designated-party protocol instances $\{\Pi_{\mathsf{m\text{-}des}}^j\}_{j\le k}$ are simulated by $\mathsf{Sim}_{\mathsf{m\text{-}des,S}}$, while the rest of the honest parties' instances are executed as in the real protocol. The hybrids $\mathsf{H}_{0,0}$ and $\mathsf{H}_{0,n-d}$ are identical to H_0 and H_1 respectively. We now show that the hybrids $\mathsf{H}_{0,k}$ and $\mathsf{H}_{0,k+1}$ are statistically indistinguishable. To argue this, we first note that the only difference between these two hybrids is the way the view of the adversary is generated in $(k+1)$-th designated-party protocol instance - in $\mathsf{H}_{0,k}$ it is generated as in the real protocol $\Pi_{\mathsf{m\text{-}des}}^{k+1}$, while in $\mathsf{H}_{0,k+1}$, it is simulated by $\mathsf{Sim}_{\mathsf{m\text{-}des,S}}$. From the security guarantees of the designated protocol, the simulated view is statistically indistinguishable from the real view when the designated party is honest, and hence, $\mathsf{H}_{0,k}$ and $\mathsf{H}_{0,k+1}$ are statistically indistinguishable. This further implies that H_0 and H_1 are statistically indistinguishable.

Next, we define H_2, which is same as H_1, but with the honest parties' messages being obtained from $\mathsf{Sim_V}$ rather than from the real execution. This means that the labels sent by the honest parties to the corrupt parties are selected based on the simulated messages output by $\mathsf{Sim_V}$. From the IT security of Π_{m}, we have that the real messages and the simulated messages are statistically indistinguishable and thus, H_1 and H_2 are statistically indistinguishable as well.
We next introduce hybrid H_3 which differs from H_2 with respect to the way pairwise label authenticity (i.e. for labels corresponding to the pairwise messages sent from corrupt parties to honest parties) is ensured: in H_3, as in the simulated view, $\mathsf{Sim_S}$ extracts both the labels and compares them with the label sent by $\mathcal{A}$.

In the hybrid H_2 however, the authenticity is ensured by the tag pairs associated with each input label. We now show that these two hybrids are statistically indistinguishable. Denote the number of labels sent by the adversary $\mathcal{A}$ to the honest parties as N. We define a series of $N+1$ hybrids $\mathsf{H}_{2,0}, \ldots, \mathsf{H}_{2,N}$ such that in hybrid $\mathsf{H}_{2,k}$, the authenticity of the first k labels is checked by comparing the extracted labels as done by Sim_S while the authenticity of the rest of the labels is checked using MACs as in the real execution. Note that the hybrid $\mathsf{H}_{2,0}$ is identical to the hybrid H_2, and the hybrid $\mathsf{H}_{2,N}$ corresponds to the hybrid H_3. We now show that the hybrids $\mathsf{H}_{2,k}$ and $\mathsf{H}_{2,k+1}$ are statistically indistinguishable. Assuming that the k-th pair of labels corresponds to pairwise messages, denote the labels and randomness as L_0, L_1, R_0, R_1, and the corresponding MAC keys and encrypted tags as $(K_0, w_0), (K_1, w_1)$. First, we observe that the two hybrids can lead to different outputs if and only if $\mathcal{A}$ sends $(L', R') \notin ((L_0, R_0), (L_1, R_1))$ such that

$$\mathsf{MAC.Verify}(K_0, L', w_0 \oplus R') = 1 \vee \mathsf{MAC.Verify}(K_1, L', w_1 \oplus R') = 1$$

In such a case, in the simulated view, $\mathsf{Sim_S}$ broadcasts abort while the real protocol proceeds further. This case, however, corresponds to an IT-MAC forgery and hence occurs with negligible probability. A similar argument follows in the case of broadcast labels, where we rely on the unforgeability of one-time multi-key MACs. We note that in this case, a rushing adversary may see an encrypted tag before it broadcasts its label and randomness; however, the unforgeability game of one-time multi-key MACs allows the adversary $\mathcal{A}$ to obtain a tag on one message before it attempts to output a forgery. This implies that the hybrids H_k and H_{k+1} are statistically indistinguishable, and hence, H_2 and H_3 are statistically indistinguishable.

Lastly, we define hybrid H_4, where $\mathsf{Sim_S}$ extracts the label and randomness shares of the corrupt parties, and aborts if they have used inconsistent shares. In hybrid H_3, this step is done by actually running a zero-knowledge protocol, where the adversary $\mathcal{A}$ proves it has used consistent shares, and honest parties either abort or continue depending on the zero-knowledge protocol output. By the soundness of the zero-knowledge proofs, we conclude that these two hybrids are statistically indistinguishable. This implies that the real view H_0 and the simulated view H_4 are statistically indistinguishable, and hence, the protocol is secure.

Computational Security. Let the PPT adversary be represented as $\mathcal{A}$ and the simulator of the designated protocol as $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$. The set of parties corrupted by $\mathcal{A}$ is denoted as Z. The simulator $\mathsf{Sim_C}$ for protocol 4.1.3 does the following steps:

1. For all designated-party protocol instances $\Pi^k_{\mathsf{m\text{-}des}}$ with $k \in [n]$ and corrupt parties $P_i \in Z$, $\mathsf{Sim_C}$ invokes $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$ to simulate the instances. $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$ and extracts P_i's additive shares of the labels and randomness as:

$$\hat{\Sigma}_i = \{\mathsf{L}^b_{j,k,i,h}, \mathsf{L}^b_{j,\mathsf{BC},i,h,r}, R^b_{j,k,i,h}, R^b_{j,\mathsf{BC},k,i,h,r}\}_{j,k\in[n],h\in[\ell],b\in\{0,1\},r\in[2]}$$

It also extracts the IT-MAC keys $\{K^b\}$ and the multi-key MAC keys $\{\mathsf{K}^b_{j,\mathsf{BC},k,i,h,r}\}_{j,k\in[n],h\in[\ell],r\in[2]}$. Additionally, from the designated-party protocol instances $\{\Pi^i_{\mathsf{m\text{-}des}}\}_{P_i\in Z}$, $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$ extracts the inputs $\{x_i\}_{P_i\in Z}$, and sends these to $\mathsf{Sim}_{\mathsf{C}}$.

2. $\mathsf{Sim}_{\mathsf{C}}$ computes and broadcasts commitments of 0 on behalf of the honest parties' inputs. It then uses $\mathsf{Sim}_{\mathsf{ZK}}$ to simulate the zero-knowledge proofs from the honest parties to the corrupt parties.
3. $\mathsf{Sim}_{\mathsf{C}}$ forwards $\{x_i\}_{P_i\in Z}$ to $\mathcal{F}_{\mathsf{ua}}$, and receives the output y.
4. For all $P_i \in Z$, $\mathsf{Sim}_{\mathsf{C}}$ generates the garbled circuit for the third next-message function NM^3_i as

$$\hat{C}_{2,i}, \{\{\hat{\mathsf{L}}_{j,\mathsf{BC},h,2}\}_{j\in[n]}, \hat{\mathsf{L}}_{i,\mathsf{st},h}\}_{h\in[\ell]} \leftarrow \mathsf{Sim}_{\mathsf{GC}}(y)$$

5. For all $i,j \in [n], h \in [\ell]$, $\mathsf{Sim}_{\mathsf{C}}$ randomly samples $\hat{R}^0_{j,\mathsf{BC},i,h,2}$, the keys $\{\mathsf{K}_{j,\mathsf{BC},i,k,h,2}\}_{P_k\notin Z}$ and computes

$$\hat{w}^0_{j,\mathsf{BC},i,h,2} = \hat{R}^0_{j,\mathsf{BC},i,h,2} \oplus \mathsf{MK.Sign}(\{\mathsf{K}_{j,\mathsf{BC},i,k,h,2}\}_{k\in[n]}, \hat{\mathsf{L}}_{j,\mathsf{BC},h,2})$$

It samples $\hat{w}^1_{j,\mathsf{BC},i,h,2}$ randomly, and randomly permutes $(\hat{w}^0_{j,\mathsf{BC},i,h,2}, \hat{w}^1_{j,\mathsf{BC},i,h,2})$. Denote this tuple as $\hat{w}_{j,\mathsf{BC},i,h,2}$.

6. $\mathsf{Sim}_{\mathsf{C}}$ computes the garbled circuit for the second next-message function NM^2_i as

$$\hat{C}_{1,i}, \{\hat{\mathsf{L}}_{j,\mathsf{BC},h,1}, \hat{\mathsf{L}}_{j,i,h}\}_{j\in[n],h\in[\ell]} \leftarrow \mathsf{Sim}_{\mathsf{GC}}(\{\hat{\mathsf{L}}_{i,\mathsf{BC},h,2} \| \hat{\boldsymbol{R}}_{i,\mathsf{BC},h,2}, \hat{\mathsf{L}}_{i,\mathsf{st},h}\}_{h\in[\ell]})$$

where $\boldsymbol{R}_{i,\mathsf{BC},h,2}$ is randomly sampled for $h \in [\ell]$.

7. For all $i,j \in [n], h \in [\ell]$, $\mathsf{Sim}_{\mathsf{C}}$ randomly samples $\hat{R}^0_{j,\mathsf{BC},i,h,1}$, the keys $\{\mathsf{K}_{j,\mathsf{BC},i,k,h,1}\}_{P_k\notin Z}$ and computes

$$\hat{w}^0_{j,\mathsf{BC},i,h,1} = \hat{R}^0_{j,\mathsf{BC},i,h,1} \oplus \mathsf{MK.Sign}(\{\mathsf{K}_{j,\mathsf{BC},i,k,h,1}\}_{k\in[n]}, \hat{\mathsf{L}}_{j,\mathsf{BC},h,1})$$

It samples $\hat{w}^1_{j,\mathsf{BC},i,h,1}$ randomly, and permutes $\hat{w}_{j,\mathsf{BC},i,h,1} = (\hat{w}^0_{j,\mathsf{BC},i,h,1}, \hat{w}^1_{j,\mathsf{BC},i,h,1})$ randomly.

8. $\mathsf{Sim}_{\mathsf{C}}$ randomly samples $\hat{R}^0_{j,i,h}$ and computes $\hat{w}^0_{j,i,h} = \hat{R}^0_{j,i,h} \oplus \mathsf{MAC.Tag}(K^b_{j,i,h}, \hat{L}_{j,i,h})$, where b is randomly sampled. It samples $\hat{w}^1_{j,i,h}$ randomly, and permutes $(\hat{w}^0_{j,i,h}, \hat{w}^1_{j,i,h})$ randomly. Denote this tuple as $\hat{w}_{j,i,h}$.
9. $\mathsf{Sim}_{\mathsf{C}}$ randomly samples labels $\hat{\mathsf{L}}_{i,j}$ and randomness $\hat{R}_{i,j}$ for $P_j \notin Z$. It sets the output of the F_i instance in $\Pi_{\mathsf{m\text{-}des}}$ as $(\{\hat{\mathsf{L}}_{i,j}, \hat{R}_{i,j}\}_{j\in[n]}, \hat{\mathsf{L}}_{i,\mathsf{BC},1}), \{\hat{\boldsymbol{R}}_{i,\mathsf{BC},h,1}\}_{h\in[\ell]}$, the output of the G_i instance as $(\hat{C}_{1,i}, \{\hat{w}_{j,i,h}, \hat{w}_{j,\mathsf{BC},i,h,1}\}_{j\in[n],h\in[\ell]})$, and the output of the H_i instance as $(\hat{C}_{2,i}, \{\hat{w}_{j,\mathsf{BC},i,h,2}\}_{j\in[n],h\in[\ell]})$.
10. $\mathsf{Sim}_{\mathsf{C}}$ completes the remaining simulation of $\mathsf{Sim}^j_{\mathsf{m\text{-}des},\mathsf{C}}$ for all $j \in [n]$ by forwarding the messages of $\mathsf{Sim}^j_{\mathsf{m\text{-}des},\mathsf{C}}$ and $\mathcal{A}$ to each other.
11. For $P_i \in Z, P_j \notin Z, h \in [\ell]$, $\mathsf{Sim}_{\mathsf{C}}$ sends the simulated labels and randomness $\{\hat{\mathsf{L}}_{j,i,h}, \hat{R}_{j,i,h}, \hat{\mathsf{L}}_{j,\mathsf{BC},h,1}, \hat{\boldsymbol{R}}_{j,\mathsf{BC},k,h,1}\}$ to $\mathcal{A}$ and broadcasts $\{\hat{w}_{k,\mathsf{BC},j,h,1}\}_{k\in[n]}$.

12. For all $P_i \in Z$, $\mathsf{Sim_C}$ receives the labels $\{\tilde{\mathsf{L}}_{i,\mathsf{BC},h,1}, \{\tilde{\mathsf{L}}_{i,j,h}\}_{P_j \notin Z}\}_{h\in[\ell]}$ and the randomness $\{\tilde{R}_{i,j,h}, \tilde{\boldsymbol{R}}_{i,\mathsf{BC},h,1}\}$, along with the encrypted tags $\{\tilde{w}_{k,\mathsf{BC},i,h,1}\}_{k\in[n],h\in[\ell]}$ from $\mathcal{A}$. For any $P_j \notin Z, h \in [\ell]$, if

$$(\tilde{\mathsf{L}}_{i,\mathsf{BC},h,1} \neq \hat{\mathsf{L}}_{i,\mathsf{BC},h,1}) \vee (\tilde{\mathsf{L}}_{i,j,h} \neq \hat{\mathsf{L}}_{i,j,h}) \vee (\tilde{R}_{i,j,h} \neq \hat{R}_{i,j,h}) = 1$$

or if $\hat{R}_{i,\mathsf{BC},j,h,1} \neq \tilde{R}_{i,\mathsf{BC},j,h,1}$ or $\hat{w}_{k,\mathsf{BC},i,h,1} \notin \tilde{w}_{k,\mathsf{BC},i,h,1}$, it broadcasts abort and terminates.

13. $\mathsf{Sim_C}$ participates in the ZK instances initiated by the corrupt parties. As explained in step 1, it extracts the input Σ'_i used by each $P_i \in Z$ in the designated-party protocol instances $\{\Pi^j_{\mathsf{m\text{-}des}}\}_{P_j \notin Z}$. If any $P_i \in Z$ has used inconsistent inputs in these instances, $\mathsf{Sim_C}$ broadcasts abort and terminates.
14. For $P_j \notin Z$, $\mathsf{Sim_C}$ sends the simulated labels $\{\hat{\mathsf{L}}_{j,\mathsf{BC},h,2}\}_{h\in[\ell]}$ to $\mathcal{A}$. It also sends broadcasts $\{\hat{w}_{k,\mathsf{BC},j,h,1}\}_{k\in[n],h\in[\ell]}$.
15. $\mathsf{Sim_C}$ receives the labels $\{\tilde{\mathsf{L}}_{i,\mathsf{BC},h,2}\}_{h\in[\ell]}$ along with the encrypted tags $\{\tilde{w}_{k,\mathsf{BC},i,h,2}\}_{k\in[n],h\in[\ell]}$ from $\mathcal{A}$. For any $P_i \in Z, h \in [\ell]$, if $\tilde{\mathsf{L}}_{i,\mathsf{BC},h,2} \neq \hat{\mathsf{L}}_{i,\mathsf{BC},h,2}$ or if $\hat{R}_{i,\mathsf{BC},j,h,1} \neq \tilde{R}_{i,\mathsf{BC},j,h,1}$ or $\hat{w}_{k,\mathsf{BC},i,h,2} \notin \tilde{w}_{k,\mathsf{BC},i,h,2}$, it broadcasts abort and terminates.

Let $\mathcal{A}$ be a PPT adversary corrupting an arbitrary set of parties. We show that the view of $\mathcal{A}$ in the real execution is computationally indistinguishable from its view generated in the simulated execution produced by $\mathsf{Sim_C}$. The view of the adversary in the real protocol consists of its views in the instances of $\{\Pi^i_{\mathsf{m\text{-}des}}\}_{i\in[n]}$ and the labels, along with the encrypted tags received from the honest parties. In the simulated view, each instance $\Pi^i_{\mathsf{m\text{-}des}}$ for $i \in [n]$ is simulated, and the labels sent by the honest parties as well as the encrypted tags are also generated by the simulator. For the instances $\Pi^i_{\mathsf{m\text{-}des}}$ where $P_i \in Z$, the simulator also generates a simulated garbled circuit as the output of these instances. We now prove the computational indistinguishability of the real view and the simulated view via a sequence of hybrids.

We start with hybrid H_0, which corresponds to the real execution of the protocol. In hybrid H_1, the commitments in the real execution are replaced with commitments on 0. By the computational hiding property of the commitment scheme, hybrids H_0 and H_1 are computationally indistinguishable. Next, we define H_2, where the zero-knowledge proof instances $\{\mathsf{ZK}_{j,i}\}_{P_j \notin Z, P_i \in Z}$ are replaced with their simulations. From the computational zero-knowledge property, it follows that H_1 and H_2 are computationally indistinguishable. This is followed by hybrid H_3, where each of the $\Pi_{\mathsf{m\text{-}des}}$ instances is replaced with its simulation. We define $n+1$ intermediate hybrids $\{\mathsf{H}_{2,k}\}_{k\in[n]\cup\{0\}}$, where $\mathsf{H}_{2,0}$ corresponds to H_2 and $\mathsf{H}_{2,n}$ corresponds to H_3. $\mathsf{H}_{2,k}$ is defined as follows:

Hybrids $\mathsf{H}_{2,k}$ *for* $k \in [n] \cup \{0\}$. In hybrid $\mathsf{H}_{2,k}$, the first k instances of $\Pi_{\mathsf{m\text{-}des}}$ are simulated using $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$, while the remaining instances are executed as in the real protocol. The only difference between $\mathsf{H}_{2,k-1}$ and $\mathsf{H}_{2,k}$ is with respect to the k-th $\Pi_{\mathsf{m\text{-}des}}$ instance. In $\mathsf{H}_{2,k-1}$, this instance is executed as in the real protocol, while in $\mathsf{H}_{2,k}$, it is simulated using $\mathsf{Sim}_{\mathsf{m\text{-}des},\mathsf{C}}$. This implies that if there exists

a PPT distinguisher that distinguishes these hybrids, then there exists a PPT distinguisher that can break the computational security of $\Pi_{\mathsf{m\text{-}des}}$. Thus, it holds that $\mathsf{H}_{2,k-1} \approx_c \mathsf{H}_{2,k}$, and therefore, $\mathsf{H}_2 \approx_c \mathsf{H}_{3,n} = \mathsf{H}_3$.

We now define hybrid H_4 by replacing each of the garbled circuits and labels output to the corrupt parties by the designated-party protocol with simulated garbled circuits and labels.

Hybrid H_4 In this hybrid, for all $P_i \in Z$, the garbled circuits $\hat{C}_{1,i}, \hat{C}_{2,i}$ and their corresponding input labels, which are the outputs of the instance $\Pi^i_{\mathsf{m\text{-}des}}$, are simulated using $\mathsf{Sim}_{\mathsf{GC}}$. The only difference between H_3 and H_4 is in the generation of the garbled circuits and labels. By the simulation security of the garbling scheme, the simulated garbled circuits and labels generated using $\mathsf{Sim}_{\mathsf{GC}}$ are computationally indistinguishable from a real garbled circuit corresponding to the same function and inputs. To show the indistinguishability of H_3 and H_4, we introduce $n-d+1$ intermediate hybrids, where d is the number of honest parties. In the hybrid $\mathsf{H}_{3,k}$, the garbled circuits for the first k corrupt parties are simulated, and generated as in the protocol for the remaining $n-d-k$ corrupt parties. Thus, the only difference between hybrid $\mathsf{H}_{3,k-1}$ and $\mathsf{H}_{3,k}$ is the way the garbled circuit of the k-th corrupt party is generated. From the privacy of the garbling scheme, it follows that hybrids $\mathsf{H}_{3,k-1}$ and $\mathsf{H}_{3,k}$ are computationally indistinguishable, and thus, hybrids H_3 and H_4 are computationally indistinguishable.

The differences between the hybrid H_4 and the simulated view correspond to the way the encrypted tags are computed, the way the authenticity of the labels sent by the adversary is checked, and the way the zero-knowledge proof instances from a corrupt party to an honest party are executed. In the simulated view, the $\mathsf{Sim}_{\mathsf{C}}$ computes the encrypted tag on the simulated label and samples the other tag randomly, while in hybrid H_4, the encrypted tags are computed as in the protocol. We first introduce hybrid H_5, which differs from H_4 in that the tags corresponding to the labels sent in the protocol are now computed as in the simulated view (i.e. the tag is now computed on the simulated label instead of the actual label). From the privacy of the garbling scheme, it follows that these two hybrids are indistinguishable.

Next, we introduce hybrid H_6, where the tag corresponding to the other labels, i.e. the labels which aren't sent in the protocol, are randomly sampled as in the simulated view. Consider one such label, say L^{1-b}, where L^b is sent in the protocol. For such a label, the corresponding randomness R^{1-b} is never opened. Also, since the view generated is (computationally) independent of the honest shares of R^{1-b}, it follows that R^{1-b} is pseudorandom. Thus, the corresponding encrypted tag w^{1-b} is pseudorandom, and hence, hybrids H_5 and H_6 are computationally indistinguishable.

The difference between H_6 and the simulated view is the way the label authenticity is checked, and the way the consistency of the corrupt parties' inputs is checked. We handle the difference in checking the consistency first. We introduce a hybrid H_7, which, as in the simulation, extracts the corrupt parties' shares and aborts of they have used inconsistent shares. This is different from H_6, where the protocol aborts or continues depending on the output of

the zero-knowledge proof instances (corrupt party as prover, honest party as verifier). By the soundness of the zero-knowledge proof, we conclude that H_6 and H_7 are indistinguishable. Next, we handle the difference in checking the label authenticity. In H_7, this is done as in the real protocol by checking the pair of tags for each party, while in the simulated view, $\mathsf{Sim_C}$ checks if the adversary provides the simulated label, and if not, it aborts. We now show that if the adversary provides any label other than the simulated label set by $\mathsf{Sim_C}$, the MAC check with the encrypted tags fails except with negligible probability. Since the MACs we use provide statistical guarantees, the indistinguishability of H_7 and the simulated view follows as argued in the statistical case. Thus, the real view and the simulated view are computationally indistinguishable. □

E.3 Proof of Malicious Fair Protocol (Theorem 4.2.4)

Proof. **Statistical Security.** We first prove that when $t < \frac{n}{2}$, $\Pi_{\mathsf{m\text{-}fair}}$ realizes the functionality $\mathcal{F}_{\mathsf{fair}}$. Denoting the set of corrupt parties as Z, the simulator $\mathsf{Sim_f}$ does the following:

1. Emulate $\mathcal{F}_{\mathsf{ss\text{-}ua}}$ as follows:
   - Receive inputs $x_i, \{\mathsf{K}_{i,j}\}_{j\in[n]}$ for each $P_i \in Z$ from $\mathcal{A}$.
   - Sample $\{[y]_i^t\}_{P_i\in Z}$ randomly and $\mathsf{K}_{j,i} \leftarrow \mathsf{MK.KeyGen}(1^s)$ for all $P_i \in Z, P_j \notin Z$.
   - Compute $\tau_i = \mathsf{MK.Sign}(\{\mathsf{K}_{j,i}\}_{j\in[n]}, [y]_i^t)$ for $P_i \in Z$.
   - Send output $\{([y]_i^t, \tau_i)\}_{P_i\in Z}$ to $\mathcal{A}$.
2. If $\mathcal{A}$ sends abort, then $\mathsf{Sim_f}$ sends $\perp$ to $\mathcal{F}_{\mathsf{fair}}$ and $\mathcal{A}$, and terminates.
3. Otherwise, receive y from $\mathcal{F}_{\mathsf{fair}}$ and extrapolate a $t-$degree polynomial $Y(x)$ using the points $\{(0, y), \{i, [y]_i^t\}_{P_i\in Z}\}$.
4. For all $P_j \notin Z$, compute $[y]_j^t = Y(j)$ and $\tau_j = \mathsf{MK.Sign}(\{\mathsf{K}_{k,j}\}_{k\in[n]}, [y]_j^t)$, and send these values to $\mathcal{A}$. Note that if the honest parties receive the output from $\mathcal{F}_{\mathsf{ss\text{-}ua}}$, then they can reconstruct the output y using just their shares. If $\mathcal{A}$ sends the wrong shares for any corrupt party, the honest parties detect it except with negligible probability and discard the share. Thus, the correctness follows from the unforgeability of the one-time multi-key IT-MAC scheme.

Computational Security. We now prove that when $\frac{n}{2} \leq t < n$, then $\Pi_{\mathsf{m\text{-}fair}}$ gives security with UA. The privacy of the protocol follows from that of the $\Pi_{\mathsf{m\text{-}ua}}$ protocol, which is used to realize the $\mathcal{F}_{\mathsf{ss\text{-}ua}}$ protocol. We now analyze the correctness. As argued in the statistical case, from the unforgeability of the MAC scheme, wrong shares sent by the corrupt parties are discarded except with negligible probability. This ensures that if an honest party recovers output y', then $y = y'$ except with negligible probability. Further, since these shares are broadcasted, all honest parties either reconstruct y or abort the protocol.

□