Round-Optimal Secure Multiparty Computation with Honest Majority

Prabhanjan Ananth[1], Arka Rai Choudhuri[2], Aarushi Goel[2], and Abhishek Jain[2]

[1] Massachusetts Institute of Technology, USA,
`prabhanjan@csail.mit.edu`
[2] Johns Hopkins University, USA,
{`achoud,aarushig,abhishek`}`@cs.jhu.edu`

Abstract. We study the exact round complexity of secure multiparty computation (MPC) in the honest majority setting. We construct several *round-optimal* n-party protocols, tolerating any $t < \frac{n}{2}$ corruptions.

1. **Security with abort:** We give the first construction of two round MPC that achieves security with abort against *malicious* adversaries in the plain model. Our protocol achieves unconditional security for NC^1 computations, and requires one-way functions for general computations.
2. **Guaranteed output delivery:** We also construct protocols that achieve security with guaranteed output delivery: **(i)** Against *fail-stop* adversaries, we construct two round MPC either in the (bare) public-key infrastructure model with no additional assumptions, or in the plain model assuming two-round semi-honest oblivious transfer. In three rounds, however, we can achieve security assuming only one-way functions (or unconditionally, for NC^1 computations). **(ii)** Against *malicious* adversaries, we construct three round MPC in the plain model, assuming public-key encryption and Zaps.
   Previously, such protocols were only known based on specific learning assumptions and required the use of common reference strings.

All of our results are obtained via general compilers that may be of independent interest.

1 Introduction

The notion of secure multiparty computation (MPC) [30,19] is fundamental in cryptography. Informally speaking, an MPC protocol allows mutually distrusting parties to jointly evaluate a function over their private inputs in such a manner that the protocol execution does not leak anything beyond the function output.

A fundamental measure of efficiency in MPC is *round complexity*, i.e., the number of rounds of communication between the parties. Protocols with smaller round complexity are more desirable so as to minimize the effect of network latency, which in turn decreases the time complexity of the protocol. Over the last three decades, the round complexity of MPC has been extensively studied in various security models.

MPC with Honest Majority. In this work, we study the exact round complexity of MPC in the *honest majority* setting, where an adversary may corrupt up to t parties out of $n = 2t+1$ total parties. We seek to construct round-optimal protocols in the plain model without any trusted setup assumptions.

The study of MPC in the honest majority model was initiated in the works of [5,9]. We recall the main security notions that have been studied over the years in this model:

- **Security with Abort:** In this notion, an adversary may learn the function output but prevent the honest parties from doing so by prematurely aborting the protocol. This is the most well-studied notion in the dishonest majority setting, where four rounds are known to be necessary for security against malicious adversaries [15,26]. Interestingly, this lower bound does not hold in the honest majority setting, which opens doors to achieving this notion in fewer rounds.
- **Guaranteed Output Delivery:** This notion guarantees that the honest parties always learn the function output (computed over the inputs of "active" parties) even if some parties prematurely abort the protocol. A relaxation of this notion, referred to as fairness, guarantees that either all the parties learn the output or no one does.

  It is well known that fairness and guaranteed output delivery are impossible to realize for general functions in the dishonest majority setting [10]. In the honest majority setting, however, these notions are indeed possible (see, e.g., [5,11]).

Our Questions. We now summarize the state of the art results for the aforementioned security notions and state our motivating questions. We refer the reader to Section 1.2 for a more comprehensive survey of prior work.

We first focus on security with abort. Ishai et al. [25] constructed two round protocols in a "super" honest majority model where a malicious adversary can corrupt up to $t \leqslant \frac{n}{3}$ parties (see also [23,28] for efficiency improvements when $n = 3$, $t = 1$). Their protocol achieves a weaker notion of *selective security with abort*, where the adversary can choose which honest parties learn the output. This is necessary since their protocol only uses private channels.

We ask whether it is possible to handle the optimal corruption threshold of $t < \frac{n}{2}$ in two rounds (which are known to be optimal [22]), while also achieving standard notion of security with abort (with the additional use of broadcast):

Q1: *Does there exist a two round MPC protocol in the plain model that achieves security with abort against any $t < \frac{n}{2}$ malicious corruptions?*

For the case of *semi-honest* adversaries, Ishai and Kushilevitz [24] constructed two-round MPC in the super honest majority model assuming only one-way functions for general computations, and with unconditional security for NC^1 computations. More recently, several new two-round MPC protocols have been constructed (see [16,6] and references therein); however, these protocols necessarily require at least semi-honest oblivious transfer since they can also handle a dishonest majority of corruptions. We ask whether it is possible to construct two-round (semi-honest) MPC with honest majority, from minimal assumptions:

Q2: *Does there exist a two round MPC protocol in the plain model against any $t < \frac{n}{2}$ corruptions, with unconditional security (or based only on one-way functions for general computations)?*

We next consider the stronger notion of guaranteed output delivery. In this setting, Gennaro et al. [17] established the impossibility of two round protocols against $t \geqslant 2$ malicious adversaries in the plain model. More recently, Gordon et al. [20] established the impossibility of two round protocols over broadcast channel (but no private channels) against fail-stop[3] adversaries in the common reference string (CRS) model. Put together, these works leave open the possibility of achieving guaranteed output delivery against fail-stop adversaries in two rounds using private channels in the plain model or just using broadcast channels in the bare public-key (BPK) model.[4]

Q3: *Does there exist a two round MPC protocol that achieves guaranteed output delivery against any $t < \frac{n}{2}$ fail-stop corruptions?*

In the broadcast-only model, Gordon et al. [20] also constructed a three round protocol with guaranteed output delivery tolerating $t < \frac{n}{2}$ fail-stop corruptions.[5] Their protocol, however, requires the use of a CRS and its security is based on the learning with errors assumption. To achieve security against malicious adversaries, they compile their protocol with non-interactive zero-knowledge (NIZK) proofs [7,14]. Their work leaves open the possibility of constructing three round protocols in the plain model, based on general assumptions:

Q3: *Does there exist a three round MPC protocol over broadcast channel that achieves guaranteed output delivery against any $t < \frac{n}{2}$ malicious corruptions based on general assumptions, in the plain model?*

1.1 Our Results

In this work, we resolve all of the aforementioned questions in the affirmative. Below, we elaborate upon our results in more detail. Unless mentioned otherwise, all of our results are in the *plain model*, and do not require any trusted setup.

Security with Abort. We construct two-round MPC for general computations that achieves security with abort against any minority of malicious corruptions,

[3] A fail-stop adversary behaves like a semi-honest adversary, except that it may choose to abort at any point (on all the communication channels) based on its view.

[4] The BPK model was proposed in [8] where, prior to the start of the protocol, every player is required to declare a public key and store it in a public file. Since no assumptions are made on whether or not the public keys deposited are unique or "bad", this is considered a weaker model than the standard PKI model.

[5] Assuming a special-purpose public-key infrastructure, their protocol can be collapsed to two rounds.

based on one-way functions. For the case of NC^1 computations, we can, in fact, achieve unconditional security.

Theorem 1 (Informal). *Assuming one-way functions, there exists a two round MPC protocol for general circuits that achieves security with abort against any $t < \frac{n}{2}$ malicious corruptions. For the case of NC^1 circuits, the protocol achieves information-theoretic security.*

We emphasize that our protocol in the above theorem only makes *black-box* use of one-way functions. In order to prove the above theorem, we devise a general compiler that "compresses" an arbitrary polynomial round MPC protocol (that may use both broadcast and private point-to-point channels) that achieves security with abort against any minority of malicious corruptions [9,5] into a two round MPC protocol. Our compiler builds upon the recent beautiful work of Garg and Srinivasan [16] who construct two-round UC secure MPC with dishonest majority from two-round UC secure oblivious transfer, in the CRS model. Indeed, our compiler can be viewed as an honest-majority analogue of their work (in the plain model).

Guaranteed Output Delivery. We next turn our attention to constructing protocols with guaranteed output delivery. We first consider security against *fail-stop* adversaries. In this case, we devise a round-preserving compiler that accepts any two-round semi-honest MPC protocol with a "delayed-function" property[6] and outputs a new protocol that achieves guaranteed output delivery against non-rushing fail-stop adversaries. If the underlying protocol tolerates semi-malicious[7] corruptions, then the resulting protocol achieves security against rushing, semi-malicious fail-stop adversaries. Our compiler only requires the use of one-way functions, and this assumption can be removed if the next-message functionality of each party in the input protocol can be represented by an NC^1 circuit.

Theorem 2 (Informal). *Assuming one-way functions, there exists a general compiler that transforms any two round semi-honest (resp., semi-malicious) MPC protocol with delayed-function property into a two-round protocol that achieves guaranteed output delivery against non-rushing fail-stop (resp., rushing, semi-malicious fail-stop) adversaries.*

Our compiler yields the following two kinds of protocols: (i) Protocols in the plain model that use only *private* channels, if the underlying protocol only uses broadcast channels. (ii) Protocols in the (bare) public-key model that only use *broadcast* channels, if the underlying protocol uses private channels. Note that in the latter case, the use of BPK model is necessary due to the impossibility result of [20].

[6] Roughly, a two-round MPC protocol satisfies the delayed-function property if the first round messages of the honest parties are computed independent of the function and the number of parties.

[7] A semi-malicious adversary is similar to a semi-honest adversary, except that it may choose its input and randomness arbitrarily [1].

By applying our compiler from theorem 2 on a variant of the protocol from Theorem 1 that achieves delayed-function[8] property, we obtain the following result in the BPK model:

Corollary 1 (Informal). *There exists a two round MPC protocol over broadcast channels in the BPK model that achieves guaranteed output delivery against any $t < \frac{n}{2}$ (semi-malicious) fail-stop corruptions.*

Furthermore, the above protocol can also be easily modified to obtain a three round protocol in the plain model based only on one-way functions (or with unconditional security, for NC^1 computations).

Next, by applying the compiler from Theorem 2 on a delayed-function variant of the semi-honest protocol from [16] (that only uses broadcast channels, unlike the protocol in Theorem 1), we get the following result:

Corollary 2 (Informal). *Assuming the existence of semi-honest (resp., semi-malicious), two-round oblivious transfer, there exists a two round MPC protocol over private channels that achieves guaranteed output delivery against any $t < \frac{n}{2}$ fail-stop (resp., semi-malicious fail-stop) corruptions in the plain model.*

We next consider security against *malicious* adversaries. We devise another compiler that accepts any two-round MPC protocol with guaranteed output delivery against semi-malicious fail-stop adversaries and outputs a three-round protocol that achieves guaranteed output delivery against malicious adversaries. The main tool used in our compiler is a new notion of *multi-verifier zero-knowledge* (MVZK) proofs, which may be of independent interest. Briefly, an MVZK protocol is a multiparty interactive protocol between a single prover and multiple verifiers where the soundness and zero knowledge properties only hold when a majority of the parties are honest. An MVZK must also achieve a strong completeness property such that the honest verifiers always accept the proof given by an honest prover even if some of the verifiers (who constitute a minority) are dishonest.

Theorem 3 (Informal). *Assuming public-key encryption and two round delayed-input[9] multi-verifier zero-knowledge arguments, there exists a general compiler that transforms any two round MPC protocol with guaranteed output delivery against semi-malicious fail-stop adversaries into a three round protocol with guaranteed output delivery against malicious adversaries.*

Our compiler only requires broadcast channels, and is therefore optimal in the number of rounds. We next give a simple construction of delayed-input MVZK arguments based on Zaps [13] (i.e., two round witness indistinguishable proofs), following the construction of NIZKs in the multi-CRS model [21]. Then, applying our compiler on the protocol from Corollary 1, we obtain the following result:

[8] In the technical sections, we describe a simply modification to achieve this property. The same idea also works for the protocols of [16].

[9] Delayed-input property for MVZK is defined in the same manner as two-party interactive proofs [27], namely, by allowing the prover to choose the instance after the first round of the protocol.

Corollary 3 (Informal). *Assuming Zaps and public-key encryption, there exists a three round MPC protocol over broadcast channels that achieves guaranteed output delivery against any* $t < \frac{n}{2}$ *malicious corruptions.*

1.2 Related Work

Over the years, the round complexity of MPC has been extensively studied both in the honest majority and dishonest majority settings. Here, we focus on the honest majority setting and refer the reader to [2] for a comprehensive survey of the literature in the dishonest majority setting.

The study of constant-round MPC was initiated by Beaver et al. [3]. They constructed such protocols against malicious adversaries in the honest majority setting using pseudorandom generators (PRGs). Damgård and Ishai [12] later achieved a similar result by making only black-box use of PRGs.

In a seminal work, Ishai and Kushilevitz [24] constructed two round and three round semi-honest MPC protocols that tolerate $t \leqslant \frac{n}{3}$ and $t < \frac{n}{2}$ corruptions, respectively. Subsequently, Ishai et al. [25] constructed two round protocols against $t \leqslant \frac{n}{3}$ malicious corruptions, achieving selective security with abort. More recently, Ishai et al. [23] and [28] constructed simpler two round protocols for $n = 3$ parties that tolerate any single, malicious corruption.

While the work of [5] already achieved fairness, several subsequent works also achieve guaranteed output delivery (see, e.g., [11] for references). We highlight a few results in this regime. Damgård and Ishai [12] constructed a three round MPC protocol with guaranteed output delivery for $t < \frac{n}{5}$. Ishai et al. [25] constructed a two round protocol with guaranteed output delivery using only private channels for the special case of $t = 1$, $n \geqslant 5$. Further, they also constructed a two-round protocol in the client server model, with n clients and $m > 2$ servers, that tolerates a single corrupted client and $t \leqslant \frac{n}{5}$ colluding servers. Subsequently, Asharov et al. [1] constructed five round protocols with guaranteed delivery for $t < N/2$, assuming learning with errors (LWE) and NIZKs. More recently, Ishai et al. [23] constructed several protocols with guaranteed output delivery for the case of $t = 1$ and $n = 4$: a two round statistically secure protocol for linear functionalities, a two round computationally secure protocol for general functionalities with guaranteed output delivery from injective one-way functions, and a two round unconditionally secure protocol for general functionalities with guaranteed output delivery in the preprocessing model. Gordon et al. [20] constructed a three round protocol with guaranteed output delivery in the CRS model with broadcast-only messages assuming LWE and NIZKs.

Gennaro et al. [17] established a lower bound for achieving guaranteed output delivery against malicious adversaries. They ruled out the existence of two round protocols in the plain model that achieve guaranteed output delivery against $t \geqslant 2$ malicious parties. More recently, Gordon et al. [20] established a stronger lower bound for protocols that only use broadcast channels. Specifically, they ruled out the existence of two round protocols over broadcast channels that achieve guaranteed output delivery against (non-rushing) fail-stop adversaries in the CRS model.

2 Preliminaries

In our constructions, we make use of some well studied primitives like *garbled circuits* [30] and *threshold secret sharing* [29]. While garbled circuits with selective security suffice for our application in section 4, we require *adaptive garbled circuits* in section 5. An adaptively secure garbled circuit is one where the adversary first gets to see a garbling for any circuit of his choice. After seeing this garbled circuit, he can adaptively choose an input and obtain labels corresponding to that input. For our application, the online complexity (i.e., size of input wire labels) is not important; as such it suffices to use one-time pads with Yao's garbled circuits as suggested in the work of Bellare et al. [4] to obtain adaptively secure garbled circuits from one-way functions. We refer the reader to [4] or the full version of our paper for a formal definition of this primitive.

We use various notions of security for secure multiparty computation (MPC) in our constructions. Apart from the standard notion of *security with abort* that guarantees correctness of output of the honest parties (when the adversary does not prematurely abort), we also consider a relaxed notion of security called *privacy with knowledge of output* [25]. The only difference from the definition for security with abort is that in the ideal world, on receiving outputs from the trusted party, the adversary can choose the output it wants to send to the honest parties. It is easy to see that this is a weaker notion since the correctness of output for the honest parties is no longer guaranteed. A formal definition of this can be found in the full version of our paper, or in [25]. We also consider security with *guaranteed output delivery* against both fail-stop and malicious adversaries.

For our application in section 5, we also consider MPC protocols with a *delayed function property*, i.e., protocols where the first round messages of all parties are independent of the function and the number of parties in the protocol.

3 Definitions

In this section, we define some new notions that we consider in this work.

3.1 Multiparty Oblivious Transfer Protocol

A multiparty oblivious transfer (OT) protocol consists of n-parties, where one of the parties P_n is the receiver and every other party $P_1, \ldots, P_{n-1}$ is a sender. Sender P_i has inputs $m_{i,0}, m_{i,1}$ and the receiver R has a private input σ_1. At the end of this protocol every party learns only $\{m_{i,\sigma_1}\}_{i \in [n-1]}$.

Before proceeding we note that every player gets the output. Therefore, on completion of the protocol there is no receiver security. For our applications this is completely fine. On the other hand, we will insist that if the second round of the protocol is not executed the receiver privacy is maintained. We should also point out that we have given a general definition, but this can be appropriately modified to let only the receiver obtain the output by setting every other party's output to be $\perp$.

We consider protocols that have both broadcast and private messages m_B and m_{priv}. But for convenience of notation we denote this as $m \coloneqq (m_B, m_{\mathsf{priv}})$. When we say such a message is sent, we indicate that m_B is sent by broadcast and m_{priv} is sent privately.

We consider a variant of the multiparty OT protocol, which we shall denote as *multiparty homomorphic oblivious transfer protocol.* In this variant there is a special designated sender $\widehat{S}$ $(=P_{n-1})$ with an additional input σ_2. At the end of the protocol, every party learns only $\{m_{i,\sigma_1 \oplus \sigma_2}\}_{i\in[n-1]}$. The regular multi-party OT can be thought of as a special mode of the homomorphic oblivious transfer where the additional input from this special designated sender is ignored (or set to 0). Hence, it is convenient to formally define the homomorphic notion of the multiparty OT, but we shall use both these notions:

Definition 1 (Multiparty Homomorphic OT Protocol). *A two round 1-out-of-2 multiparty homomorphic oblivious transfer protocol* $\mathsf{OT} = \left(\mathsf{OT}_1^R, \mathsf{OT}_1^{\widehat{S}}, \mathsf{OT}_1^S, \mathsf{OT}_2, \mathsf{OT}_3\right)$ *is an interactive protocol between n parties, where one of the parties is the receiver, one of the parties is a special designated sender and the others are senders. The sender parties* P_i *for* $i \in [n-1]$ *have inputs* $m_{i,0}, m_{i,1} \in \{0,1\}^\lambda$*, the receiver party* P_n *has an input bit* $\sigma_1 \in \{0,1\}$*, and the special designated sender* P_{n-1} *has an additional input* $\sigma_2 \in \{0,1\}$*.*

First round. *The parties compute their first round messages as follows:*

- ***Receiver:*** $\left\{\mathsf{ot}_n[j]^1_{\to n,n-1}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^R(\sigma_1)$ *where* $\mathsf{ot}_n[j]^1_{\to n,n-1}$ *refers to the message that the party j receives from the receiver (party* P_n*). The subscript* $\to n, n-1$ *denotes that the designated special sender is* P_{n-1} *while the receiver is* P_n*.*
- ***Special sender:*** $\left\{\mathsf{ot}_{n-1}[j]^1_{\to n,n-1}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^{\widehat{S}}\left(\left(m_{n-1,0}, m_{n-1,1}\right), \sigma_2\right)$*. The notation is almost identical to the previous case, but the notation here identifies this as a message from the special sender (party* P_{n-1}*).*
- ***Senders:*** *Each party* $i \in [n-2]$ *computes* $\left\{\mathsf{ot}_i[j]^1_{\to n,n-1}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^S(m_{i,0}, m_{i,1})$*. The notation is almost identical to the previous case, but the notation here identifies this as the message from the corresponding sender (party* P_i*).*

Each party sends P_j *its corresponding message. Thus, at the end of the first round each party* P_j *has* $\left\{\mathsf{ot}_i[j]^1_{\to n,n-1}\right\}_{i\in[n]}$

Second Round. *Each party* P_i *computes their second round message* $\mathsf{ot}_i[\perp]^2_{\to n,n-1} \leftarrow \mathsf{OT}_2\left(\left\{\mathsf{ot}_j[i]^1_{\to n,n-1}\right\}_{j\in[n]}\right)$*. Here by* $\perp$ *we denote that the message is broadcast to every party.*

Output Computation. *Every party computes the output as follows*

$$\left(\{\widetilde{m}_i\}_{i\in[n-1]}\right) \coloneqq \mathsf{OT}_3\left(\left\{\mathsf{ot}_i[\perp]^2_{\to n,n-1}\right\}_{i\in[n]}\right)$$

We require the following properties from the protocol:

1. ***Correctness:*** *For every $\sigma_1, \sigma_2 \in \{0,1\}$, and sender input messages $\forall i \in [n], b \in \{0,1\}$ $m_{i,b} \in \{0,1\}^\lambda$, $\Pr\left[\forall i \in [n-1] \quad \widetilde{m}_i = m_{i,\sigma_1 \oplus \sigma_2}\right] = 1$ where the randomness is over the coins used to compute the first and second round messages of the protocol.*
2. ***Security:*** *We consider two notions of security depending on whether or not the second round of the protocol is executed:*
   - ***Privacy.*** *If the protocol terminates at the end of the first round, then the notion of privacy is satisfied;*
   - ***Privacy with Knowledge of Outputs against Malicious Minority:*** *If the second round is executed (by the honest parties at least), then for any PPT adversary $\mathcal{A}$ controlling a minority set of the parties, there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_{OT}, \mathsf{Ext}_{OT})$ satisfying the security notion of privacy with knowledge of outputs (defined in 2).*
     The role of the extractor Ext_{OT} is to extract the adversary's input from its first round messages. On the other hand, the role of the Sim_{OT} is to generate the transcript for the protocol.

Instantiation. The multiparty homomorphic oblivious transfer protocol with inputs $(\{m_{i,b}[\ell]\}_{i \in [n-1], b \in \{0,1\}, \ell \in [\lambda]}, \sigma_1, \sigma_2)$, where $\sigma_1, \sigma_2, m_{i,b[\ell]} \in \{0,1\}$, can be thought of as a vector of degree 2 polynomials in $\mathbb{F}_2$: $\forall i \in [n-1], \ell \in [\lambda] \quad m_{i,0[\ell]} \cdot (1 + \sigma_1 + \sigma_2) + m_{i,1}[\ell] \cdot (\sigma_1 + \sigma_2)$. The work of Ishai, Kushilevitz and Paskin [25] gives us an explicit construction for such a degree 2 polynomial computation protocol:

Theorem 4 (**[25]**). *For $n = 2t + 1$, where t is the number of corrupted parties, there exists a 2 round protocol that computes a vector of polynomials of degree 2 and satisfies statistical t-privacy with knowledge of outputs.*

We note that the original stated lemma in [25] requires $|\mathbb{F}| > n$, but this condition can be relaxed to computing polynomials in $\mathbb{F}_2$ if we can construct a 2-multiplicative $(2t+1, t)$ linear secret sharing scheme that is pairwise verifiable (see [25] for details). In fact, [25] discusses how to construct such a scheme, which in turn suffices for our notion of the multiparty homomorphic OT.

3.2 Multi-Verifier Zero Knowledge Proof System

A multi-verifier zero-knowledge proof system consists of a prover P and n verifiers $V_1, \ldots, V_n$. The prover and the verifiers share a statement x that belongs to an NP-language. The prover additionally holds a private input w. If w is a valid witness for the statement x, all honest verifiers must be able to output 1. If x does not belong to the NP-language, honest verifiers should not output 1 with a very high probability. The verifiers should not learn anything about w in either case.
Consider n verifiers, where t can be corrupted. For completeness, $t \leqslant n$, for soundness, $t \leqslant n-1$, and for ZK, $t \leqslant n$. Note that in the extreme case, the definition subsumes the standard ZK definition since all the verifiers can be

combined into one. In our constructions, we will focus on the honest-majority case, where for soundness, $t \leqslant \frac{n-1}{2}$ and for ZK, $t \leqslant \frac{n}{2}$. For our constructions, we require a two round multi-verifier zero knowledge protocol that satisfies delayed input property, i.e., first round messages of both the prover and verifier and second round messages of verifier are independent of the statement.
A formal definition of this primitive is as follows:

Definition 2 (Two Round Multi-Verifier Zero Knowledge). *A two round multi-verifier zero-knowledge proof system associated with an NP relation $\mathcal{R}$ is an interactive zero-knowledge protocol with a prover P and n verifiers $V_1, \ldots, V_n$. The prover and the verifiers hold an instance x of the language $\mathcal{L}(\mathcal{R})$ defined by the relation $\mathcal{R}$. The prover also holds a string $w \in \{0,1\}^\lambda$. It can be defined as a tuple of PPT algorithms* $\mathsf{mvzk} := (\mathsf{P}^1_{\mathsf{mvzk}}, \mathsf{V}^1_{\mathsf{mvzk}}, \mathsf{P}^2_{\mathsf{mvzk}}, \mathsf{V}^2_{\mathsf{mvzk}}, \mathsf{Verify}_{\mathsf{mvzk}})$.

– $\mathsf{pMsg}^1 \leftarrow \mathsf{P}^1_{\mathsf{mvzk}}(1^\lambda)$*:* $\mathsf{P}^1_{\mathsf{mvzk}}$ *takes the security parameter λ as input and outputs first round messages of the prover.*
– $\mathsf{vMsg}^1_i \leftarrow \mathsf{V}^1_{\mathsf{mvzk}}(1^\lambda, i)$*:* $\mathsf{V}^1_{\mathsf{mvzk}}$ *takes the security parameter λ and index i of the verifier as input and outputs first round messages of the verifier.*
– $\mathsf{pMsg}^2 \leftarrow \mathsf{P}^2_{\mathsf{mvzk}}(\mathsf{trans}^1_{\mathsf{mvzk}}, x, w)$*:* $\mathsf{P}^2_{\mathsf{mvzk}}$ *takes the first round transcript of* mvzk*,* $\mathsf{trans}^1_{\mathsf{mvzk}} := (\mathsf{pMsg}^1, \{\mathsf{vMsg}^1_i\}_{i\in[n]})$*, the statement x and the witness w as input and outputs second round messages of the prover.*
– $\mathsf{vMsg}^2_i \leftarrow \mathsf{V}^2_{\mathsf{mvzk}}(i, \mathsf{trans}^1_{\mathsf{mvzk}})$*:* $\mathsf{V}^2_{\mathsf{mvzk}}$ *takes index i of the verifier and the first round transcript of* mvzk*,* $\mathsf{trans}^1_{\mathsf{mvzk}} := (\mathsf{pMsg}^1, \{\mathsf{vMsg}^1_i\}_{i\in[n]})$ *as input and outputs second round messages of the verifier.*
– $b := \mathsf{Verify}_{\mathsf{mvzk}}(i, \{\mathsf{trans}^r_{\mathsf{mvzk}}\}_{r\in[2]}, x)$*:* $\mathsf{Verify}_{\mathsf{mvzk}}$ *takes index i of the verifier, the entire transcript of the protocol and the statement x as input and outputs a bit b.*

We want the multi-verifier zero-knowledge proof system to satisfy the following properties:

1. ***Completeness:*** *For every n.u. PPT adversary $\mathcal{A}$ that corrupts up to t verifiers, let $H \subset [n]$ be the set of honest verifiers, then for every $x \in \mathcal{L}(\mathcal{R})$ and for all honest verifiers V_i, where $i \in H$,*

$$\Pr[\mathsf{Verify}_{\mathsf{mvzk}}(i, \{\mathsf{trans}^r_{\mathsf{mvzk}}\}_{r\in[2]}, x) = 1] = 1$$

2. ***Soundness:*** *For every adversary $\mathcal{A}$ controlling the prover (P^*) and upto t verifiers, let $H \subseteq [n]$ be the set of honest verifiers, then for every $x \notin \mathcal{L}(\mathcal{R})$ and for all honest verifiers V_i where $i \in H$,*

$$\Pr[\mathsf{Verify}_{\mathsf{mvzk}}(i, \{\mathsf{trans}^r_{\mathsf{mvzk}}\}_{r\in[2]}, x) = 1] \leqslant \mu(\lambda)$$

for some negligible function μ.
We require a slightly stronger notion of soundness, where soundness holds, even if an adversarial prover is allowed to choose the statement after looking at the first round messages of honest verifiers.

3. ***Zero-Knowledge:*** *For every n.u. PPT adversary $\mathcal{A}$, that corrupts upto t verifiers, let $H \subseteq [n]$ be the set of honest verifiers, then there exists a PPT Simulator* $\mathsf{Sim}_{\mathsf{mvzk}} := (\mathsf{Sim}^1_{\mathsf{mvzk}}, \mathsf{Sim}^2_{\mathsf{mvzk}})$*, s.t., for every* $x \in \mathcal{L}(\mathcal{R}), w \in \mathcal{R}(x), \mathbf{z} \in \{0,1\}^*$ *and a negligible function* $\mu(.)$*,*

$$|\Pr[\mathsf{Exp}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{Sim}_{\mathsf{mvzk}}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{Sim}_{\mathsf{mvzk}}}(1^\lambda, 1) = 1]| \leqslant \mu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{ZK}}_{\mathcal{A},\mathsf{Sim}_{\mathsf{mvzk}}}(1^\lambda, \mathsf{b})$ *is defined as follows:*

   (a) *The adversary $\mathcal{A}$ gets* $(\mathsf{pMsg}^1, \{\mathsf{vMsg}^1_i\}_{i \in H})$*, which are computed as follows:*
      – ***If*** $b = 0$*:* $\mathsf{pMsg}^1 \leftarrow \mathsf{P}^1_{\mathsf{mvzk}}(1^\lambda)$*,* $\{\mathsf{vMsg}^1_i\}_{i \in H} \leftarrow \{\mathsf{V}^1_{\mathsf{mvzk}}(1^\lambda, i)\}_{i \in H}$
      – ***If*** $b = 1$*:* $(\mathsf{pMsg}^1, \{\mathsf{vMsg}^1_i\}_{i \in H}) \leftarrow \mathsf{Sim}^1_{\mathsf{mvzk}}(1^\lambda, H)$*.*

   (b) *The adversary $\mathcal{A}$ sends* $\{\mathsf{vMsg}^1_i\}_{i \notin H}$ *and specifies x, w and gets* $(\mathsf{pMsg}^2, \{\mathsf{vMsg}^2_i\}_{i \in H})$*, which are computed as follows:*
      – ***If*** $b = 0$*:* $\mathsf{pMsg}^2 \leftarrow \mathsf{P}^2_{\mathsf{mvzk}}(\mathsf{trans}^1_{\mathsf{mvzk}}, x, w)$*,* $\{\mathsf{vMsg}^2_i\}_{i \in H} \leftarrow \{\mathsf{V}^2_{\mathsf{mvzk}}(i, \mathsf{trans}^1_{\mathsf{mvzk}})\}_{i \in H}$*.*
      – ***If*** $b = 1$*:* $(\mathsf{pMsg}^2, \{\mathsf{vMsg}^2_i\}_{i \in H}) \leftarrow \mathsf{Sim}^2_{\mathsf{mvzk}}(H, \mathsf{trans}^1_{\mathsf{mvzk}}, x)$*.*

   (c) *The adversary outputs a bit b', which is the output of the experiment.*

If the soundness property only holds against polynomial-time adversaries, then we refer to the above system as an *argument* system.

In the full version, we provide a construction of multi-verifier ZK arguments based on Zaps. Our protocol is based on the multi-CRS NIZK construction of [21], with some changes to achieve the strong completeness property.

4 Security with Abort against Malicious Adversaries

Overview. We start by providing an overview of our construction. Our starting point is the recent beautiful work of Garg and Srinivasan [16].

Recap of [16]. Garg and Srinivasan [16] constructed two-round maliciously secure MPC against dishonest majority based on any two-round OT in the CRS model with some specific security properties (that we discuss below). At a high level, their protocol works by compiling a multi-round maliciously secure protocol of a very specific syntactic structure (where each round only consists of a single bit broadcast by one party to all the other parties), which they refer to as a *conforming protocol*, into a two round protocol using OT.

The compiler of Garg and Srinivasan uses a two-round OT protocol in the CRS model with the following properties: (1) simulation-based security against malicious receivers (which implies that the simulator can extract the input bit from a malicious receiver); and (2) equivocation of the honest receiver bit. Unfortunately, in two rounds, these properties can only be achieved in the common random string (CRS) model in the dishonest majority setting.

At a high level, OT is used to transmit garbled circuit labels for a single input (that corresponds to a message in the underlying conforming protocol) to an evaluator. Loosely speaking, a "speaker" party in any round of the underlying conforming protocol sends a receiver's OT message in the first round of the two-round protocol. The receiver's message is computed using as input the bit b which is supposed to be broadcast in the underlying protocol. Note that these messages are not actually known in the first round, so the "speaker" party actually prepares multiple OT messages. Every other party (unaware of this bit ahead of time) computes the OT protocol message with the two labels for its own garbled circuit as its sender input. At a later point, when the message bit is broadcast, the OT receiver also reveals the randomness used to compute the appropriate first OT message. This enables an evaluator, different from the receiver, to obtain the appropriate labels for each garbled circuit and then evaluate them correctly.

However, this release of the randomness used to compute the OT receiver messages creates a problem during simulation against a rushing adversary since a simulator, who computes an OT receiver message on behalf of an honest party, does not know what inputs to use. For this reason, the compiler in [16] requires the ability to equivocate receiver's randomness.

Challenges. We face some challenges in adopting the template of [16] to achieve our goal of constructing a maliciously secure two-round MPC protocol in the honest majority setting from one-way functions. We highlight a couple of them below.

– *Issue #1. Replacing Oblivious Transfer:* If we have any hope of basing our construction on one-way functions, we first need to figure out how to replace the oblivious transfer protocols in [16] for the honest majority setting. Note that the oblivious transfer protocols are used in two places in [16]: (i) in the interactive secure MPC protocol and, (ii) in the transformation of conforming protocols into two-round secure MPC protocols. We handle both (i) and (ii) separately.
– *Issue #2. Private Channels:* We first handle (i) by starting with a interactive secure MPC protocol in the honest majority setting. The existence of such a protocol achieving perfect security is known in literature [9,5]. However such protocols, in addition to broadcast channels, inherently use private channels – every pair of parties has a channel designated to them such that any communication on this channel cannot be observed by an external entity. However, the approach of [16] starts with an interactive secure MPC protocol that uses only broadcast channels. Hence, we need to modify their approach that will enable us to handle private channels in the underlying interactive secure MPC protocol.

Multiparty Homomorphic Oblivious Transfer. Towards solving both the above issues, we introduce the notion of multiparty homomorphic oblivious transfer. For simplicity, we first focus on achieving the weaker goal of semi-honest secure two-round MPC in the honest majority setting.

As the name suggests, this notion is a multiparty protocol where only three of the parties have inputs and the rest of the parties have no inputs. These three parties are termed as sender, receiver and designated sender. The sender[10], has inputs (m_0, m_1), receiver has input a bit σ_1 and the designated sender has input a mask σ_2. At the end of the protocol, every party receives the output $m_{\sigma_1 \oplus \sigma_2}$. We can also consider a weaker notion where the designated sender does not supply any input and we term such a notion as multiparty OT (in particular, not homomorphic). In this case, every party receives m_{σ_1}.

We can use this protocol to replace the oblivious transfer protocols in the transformation from conforming protocols to two-round secure MPC protocols. Moreover this protocol can be instantiated from two-round perfectly secure MPC protocols for quadratic polynomials [9,5][11]. To see how this can be used to solve the issue of private channels, we make the following modifications to the framework of [16].

- We start with an interactive perfectly secure MPC protocol that uses only broadcast channels in the pre-processing setting. By pre-processing, we mean that the parties can exchange information with each other over private channels before seeing any input. Once pre-processing phase is over, the parties receive the inputs in the online phase and during this phase, they perform secure computation only using broadcast channels. Such a protocol can be achieved by starting with an perfectly secure protocol without pre-processing but using private channels: the parties can exchange one-time pads (of suitable length) in the pre-processing phase to emulate the private channels in the online phase. In particular, whenever a party P_i has to send a message to another party P_j, it encrypts its message using the one-time pad P_j sent to P_i during the online phase. We transform such a interactive MPC protocol into a conforming protocol in the pre-processing setting.
- To transform a conforming protocol in the pre-processing setting into a two-round protocol, the main challenge we encounter is to get rid of the pre-processing phase. Specifically, every party in the two-round protocol is required to commit to all its actions (corresponding to the conforming protocol) in the first round. This is not possible if we start with a conforming protocol in the pre-processing setting since the actions of the parties depend on the output of the pre-processing phase which cannot be computed before the first round in the two-round protocol. This is where we crucially use the homomorphism property of the multiparty homomorphic OT protocol.

Malicious Security. While the use of multiparty homomorphic OT protocol can be used to achieve a semi-honest secure two-round MPC protocol in the honest majority setting, we need additional mechanisms to prove security against malicious adversaries. We start by incorporating the equivocation mechanism inside our multiparty homomorphic protocol.

[10] For simplicity, we explain the main ideas using just a single sender. We use a generalized version with multiple senders.

[11] Note that [9,5] dealt with computations over large fields while we need to securely compute quadratic polynomials over boolean fields. By suitably using extension fields in [5] we can solve this issue.

Equivocation Instead of using an OT protocol that explicitly allows for randomness equivocation, we achieve a similar effect from the fact that an honest receiver's input in the multiparty OT protocol is not fixed in the adversary's view when it can corrupt only a minority of parties.

Given a maliciously secure multiparty homomorphic OT protocol satisfying these properties, one could obtain the required compiler following the above strategy. However, we do not know of such a protocol in only two rounds.

Towards this, we note that the work of Ishai, Kushilevitz and Paskin [25] construct a two round protocol for degree 2 polynomial computation, in the honest majority setting. While their protocol does not achieve full malicious security, it achieves a weaker notion they refer to as *privacy with knowledge of outputs*. Roughly, this notion is similar to standard malicious security, except that it does not guarantee correctness of outputs received by the honest parties. In particular, the adversary can explicitly set the output of the honest parties to any value of its choice (in this sense, it "knows" the honest party outputs). Since a multi-party OT can be represented as a degree 2 polynomial computation, a two-round multi-party OT protocol achieving this weaker security notion can be obtained from [25].

Our main insight is that this weaker notion of multiparty homomorphic OT can still be used to obtain our desired compiler. In the protocol by Garg and Srinivasan [16] , it is essential that OT security holds against malicious receivers that attempt to equivocate their receiver bit. It would seem that in our weaker model, since the adversary can set the output to be a value of its choice, it could potentially change the output from say m_b to m_{1-b}, where b was its input to the OT protocol. This would completely break simulation since the adversary could essentially equivocate its input, and thus the guarantees of the protocol in [16] would no longer apply. This is where we use the knowledge of output property of the protocol, i.e. the output that the honest parties receive is known to the adversary. In the case of the OT protocol, when the sender is honest, an ideal world adversary receives only m_b and m_{1-b} remains hidden. Thus the output of honest parties forced by the adversary are independent of m_{1-b}. This does not stop the adversary to from setting it to a random value. However, since messages m_b and m_{1-b} correspond to wire keys of a garbled circuit, we can rely on the security of the garbling scheme which ensures that a garbled circuit cannot be evaluated unless the evaluator has one of the keys.

4.1 Conforming Protocols

Let Φ be an n-party deterministic MPC protocol with honest majority. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties in the protocol with inputs $x_1, \ldots, x_n$ respectively. A conforming protocol can be defined by a tuple of 3 functions $(\mathsf{pre}, \mathsf{comp}, \mathsf{post})$.

Pre-processing Phase: For each $(i \in [n])$,party P_i computes the following: $(z_i, \widehat{v}_i) \leftarrow \mathsf{pre}(1^\lambda, i, x_i)$. The randomized algorithm pre takes as input, the index i of the party, its input x_i and outputs $z_i \in \{0,1\}^{\ell/n}$ and $\widehat{v}_i \in \{0,1\}^{\ell}$. $\widehat{v}_i$ is private information, that it retains with itself. We require that $\widehat{v}_i[k] = 0$ for all $k \in [\ell] \setminus \{(i-1)\ell/n, \cdots, i\ell/n\}$. z_i is a public value that is broadcast to every other party in the protocol.

Each party P_i additionally samples masks $r_{k \to i}$ for all $k \in [n] \setminus \{i\}$ of appropriate length (to be discussed shortly). The mask $r_{k \to i}$ is sent privately to P_k.

Computation Phase: The computation phase can be viewed as a series of T actions $\Phi_1, \dots, \Phi_T$. Each action Φ_r, for $t \in [T]$, can be parsed as tuple of 5 indices, $\Phi_t = (i^*, j^*, f, g, h)$, where $i^* \in [n]$, $j^* \in [n] \cup \{\perp\}$, and $f, g, h \in [\ell]$. Since Φ is a deterministic protocol, $\Phi_1, \dots, \Phi_T$, are known to each party in advance.

– For all $j \in [n] \setminus \{i\}$, $\mathbb{I}_{j \to i} = \{h \mid \Phi.(j, i, \cdot, \cdot, h)\}$, and $\mathbb{I}_i := \cup_{j \in [n] \setminus \{i\}} \mathbb{I}_{j \to i}$[12]. Hence, for each $k \in [n] \setminus \{i\}$, $r_{k \to i} \in \{0,1\}^{|\mathbb{I}_{k \to i}|}$. From each $r_{k \to i}$, we want to refer to the bit in $r_{k \to i}$ that is associated with the index h. This is achieved by defining the following function $r_{k \to i}(h) := r_{k \to i}[\rho(h)]$, where $\rho(h)$ is the index of h in $\mathbb{I}_{k \to i}$. We are able to do so because we are treating as an ordered set.
– We now create the vector $v \in \{0,1\}^\ell$ from $\widehat{v}$ and masks $r_{k \to i}$.

$$v_i[k] := \begin{cases} \widehat{v}[k] & \text{if } k \in \{(i-1)\ell/n, \cdots, i\ell/n\} \\ r_{\pi(k) \to i}[k] & \text{if } k \in \mathbb{I}_i \\ 0 & \text{otherwise} \end{cases}$$

where $\pi(k)$ is j such that $k \in \mathbb{I}_{j \to i}$. We simply update $\widehat{v}$ to include the mask bits at the appropriate position. It is important to note that these updates make sense only if for every i the sets $\mathbb{I}_{j \to i}$ are disjoint. This will indeed be enforced in the conforming protocol (see below).

Let For each $i \in [n]$, party P_i does the following:
Sets, $\mathsf{st}_i = (z_1 || \dots || z_n) \oplus v_i$
For each $t \in \{1, \dots, T\}$,

1. Parse Φ_t as (i^*, j, f, g, h)
2. **If** $i = i^*$, compute $\mathsf{st}_i[h] = \mathsf{NAND}(\mathsf{st}_i[f], \mathsf{st}_i[g]) \oplus r_{i \to j}(h)$ (where $r_{i \to j}(h) = 0$ if $j = \perp$) and broadcast $\mathsf{st}_i[h] \oplus v_i[h]$ to all other parties.
3. **Else**, updates $\mathsf{st}_i[h]$ to the bit value received from P_{i^*}.

We require each action Φ_t, for $t \in [T]$, to update a unique position h in the state. More specifically, $\forall t, t' \in [T]$ such that $t \neq t'$, if $\Phi_t = (., ., ., ., h)$ and $\Phi_{t'} = (., ., ., ., h')$, then $h \neq h'$. Additionally, for every party P_i we require that a bit at index h sent privately to a party P_j is not used as a input to a NAND computation by P_i. Formally, $\forall t \in [T]$ if $\Phi_t = (i, j, \cdot, \cdot, h)$ where $j \neq \perp$ then $\nexists t' \in [t, T]$ such that $\Phi_{t'} = (i, \cdot, h, \cdot, \cdot)$ or $\Phi_{t'} = (i, \cdot, \cdot, h, \cdot)$. We denote $A_i \subset [T]$ be set of rounds in which party P_i sends a bit.

[12] We abuse notation slightly and consider each $\mathbb{I}_{j \to i}$ to be an ordered set, ordered increasingly by h.

We note that the non-repetition of h ensures that for every i, k the sets $\mathbb{I}_{i\to k}$ are disjoint.

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

Transformation to a Conforming Protocol. Let Π be an n-party deterministic MPC protocol with honest majority. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the parties in the protocol Π. Let each party P_i have input $x_i \in \{0,1\}^m$. We want to transform this protocol Π to a conforming protocol Φ, while preserving its security and correctness. We allow the protocol Π to use both broadcast and private channels.

We can assume w.l.o.g. that only a single bit is communicated by one party in each round of Π. This can trivially be achieved by increasing the round complexity of the protocol. As discussed, this bit can be broadcast or sent to a specific party. Since only a single bit is communicated in each round by one party, the message complexity in this case is equivalent to the round complexity. Let the message/round complexity of Π after increasing the round complexity be p. Let C_r be the circuit computed in round $r \in [p]$. Again we can assume without loss of generality that this circuit is only composed of NAND gates with fan-in two and each C_r is composed of q NAND gates.

We now describe how to transform Π into a conforming protocol Φ. There are $T = pq$ rounds in Φ. Let $\ell = mn + pq$ and $\ell' = pq/n$

- $\mathsf{pre}(1, x_i)$:
  1. Samples $r_i \leftarrow \{0,1\}^m$ and $s_i \leftarrow (\{0,1\}^{g-1}||0)^{p/n}$.
  2. Output $z_i := x_i \oplus r_i||0^{\ell'}$ and $v_i := 0^{\ell/n}|| \ldots ||r_i||s_i|| \ldots ||0^{\ell/n}$
- $\mathsf{comp} := \{\Phi_1, \ldots, \Phi_T\}$: As specified in the transformation above, each round $r \in [p]$ in Π is expanded into q actions in Φ. Each of these actions $\{\Phi_t\}_t$, where $t \in \{(r-1)q+1, \ldots, rq\}$ is a single NAND computation. For each t, Φ_t is set as (i^*, j^*, f, g, h). f, g are the locations in st_{i^*} that the t^{th} NAND gate in C_r is computed on. h is the first location in st_{i^*} amongst the locations $(i^* - 1)\ell/n + m + 1$ to $i\ell/n$ that has not been updated before. For $t \in \{(r-1)q+1, \ldots, rq-1\}$, $j^* :=\perp$, and for $t = rq$, j^* is set to be the recipient of the bit in the round r of Π. If the bit is to be broadcast in round r of Π, j^* is set to $\perp$.
- $\mathsf{post}(i, \mathsf{st}_i)$ Party P_i gathers messages sent by other parties in Π from the final st_i and runs the output phase of Π to output the output.

To ensure the global invariant property (defined shortly) when there are private channels involved, we require the second property described in the conforming protocol. Namely, if a player P_i sends a bit in index h over a private channel, P_i cannot subsequently use the index h as an input to a NAND gate. This is easily fixed by "copying" the bit at index h by recomputing the bit to a new position h' in the subsequent round of P_i. This increases the number of NAND gate in each round by 1, and does not affect the transformation above.

The changes in the conforming protocol, and the transformation is to accommodate underlying protocols that use both broadcast and private channels.

The conforming protocol in [16] relies on the underlying protocol to use only broadcast channels.

4.2 Our Compiler

Building Blocks. The main primitives required in this construction are:

1. A maliciously secure conforming protocol Φ with honest majority.
2. A garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ for circuits.
3. A 2 round Multiparty Homomorphic Oblivious Transfer Protocol that works in the honest majority setting.

Theorem 5. *Assuming maliciously secure conforming protocol Φ, secure garbling scheme* $(\mathsf{Garble}, \mathsf{Eval})$ *and a 2 round multiparty homomorphic OT protocol the two round protocol Π described below achieves security with abort against any $t < \frac{n}{2}$ malicious corruptions.*

A few remarks are in order for our protocol. For the underlying protocol, we instantiate them using information theoretic protocols secure with honest majority [9,5]. For general functions, this gives us *next message functions* that are in P, and we use garbling schemes based on one-way functions. But for functions in NC^1, the corresponding *next message functions* are in NC^1 and we can use information theoretic garbling to achieve unconditional security. It should be noted that our compiler makes only black-box use of one-way functions.

While we describe our complier for malicious adversaries using both broadcast and private channels, it is easy to see that our protocol is secure against semi-honest adversaries that use only private channels.

Protocol. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties in the protocol and let $\{x_1, \ldots, x_n\}$ and $\{\widetilde{r}_1, \ldots, \widetilde{r}_n\}$ be their respective inputs and randomness. Next, we describe the protocol Π in detail:

Round 1. Each party P_i does the following:

1. Run the pre-computation phase to compute (z_i, v_i): $(z_i, \widehat{v}_i,) \leftarrow \mathsf{pre}(1^\lambda, i, (x_i, \widetilde{r}_i))$. Sample masks $\{r_{j\to i}\}_{j\in[n]\setminus\{i\}}$ of appropriate length and construct v_i as in the conforming protocol (see subsection 4.1). Broadcast z_i and send each $r_{j\to i}$ to P_j.
2. **For** each round $t \in [T]$:
   - Parse Φ_t as (i^*, j^*, f, g, h)
   - **If** P_i is the speaker, i.e., $i = i^*$, we compute the first round OT receiver messages. Specifically, for each $\alpha, \beta \in \{0, 1\}$:

     $$\left\{\mathsf{ot}_i[j]^{1,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^R\left(v_{i,h} \oplus \mathsf{NAND}\left(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta\right)\right).$$

     In the case that $j^* = \perp$, this is the regular OT (without the special designated sender). Send P_j its corresponding message.

- **Else** (if $i \neq i^*$),
  - it computes the sender OT messages. First, it generates labels for the t-th round: $\left\{\mathsf{lab}^{i,t}_{k,0}, \mathsf{lab}^{i,t}_{k,1}\right\}_{k\in[\ell]} \leftarrow \mathsf{Gen}(1^\lambda)$. Next, it computes the OT messages: $\forall \alpha, \beta \in \{0,1\}$
    - if $i = j^*$,

$$\left\{\mathsf{ot}_i[j]^{1,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^{\widehat{S}}\left(\mathsf{lab}^{i,t}_{h,r_{i^*\to i}(h)}, \mathsf{lab}^{i,t}_{h,1\oplus r_{i^*\to i}(h)}, r_{i^*\to i}(h)\right)$$ [13]

    - else, $\left\{\mathsf{ot}_i[j]^{1,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{j\in[n]} \leftarrow \mathsf{OT}_1^{S}\left(\mathsf{lab}^{i,t}_{h,0}, \mathsf{lab}^{i,t}_{h,1}\right)$

    where h is the index specified by Φ_t.

  Send $\left\{\mathsf{ot}_i[j]^{1,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{\alpha,\beta\in\{0,1\}}$ to party P_j.

Round 2. Each party P_i does the following:

1. **Set state.** The local state is defined as $\mathsf{st}_i := (z_1||\dots||z_i||\dots||z_n) \oplus v_i$
2. **For** each t from T to 1,
   (a) Parse Φ_t as (i^*, j^*, f, g, h)
   (b) Compute the second round OT messages as follows:
   $\forall \alpha, \beta \in \{0,1\}, \quad \mathsf{ot}_i[\perp]^{2,t,\alpha,\beta}_{\to i^*,j^*} \leftarrow \mathsf{OT}_2\left(\left\{\mathsf{ot}_j[i]^{1,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{j\in[n]}\right)$
   (c) Compute the garbled circuit as

$$\tilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Garble}\left(\mathsf{P}\left[i, \Phi_t, v_i, \left\{\mathsf{ot}_i[\perp]^{2,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{\alpha,\beta\in\{0,1\}}, \overline{\mathsf{lab}}^{i,t+1}, \{r_{i\to j}\}_{j\in[n]\setminus\{i\}}\right], \left\{\mathsf{lab}^{i,t}_{k,b}\right\}_{k\in[\ell],b\in\{0,1\}}\right).$$

   where the program P is defined in Figure 1.
3. Broadcast the garbled program, and the keys to the first circuit: $\left(\{\tilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \left\{\mathsf{lab}^{i,1}_{k,\mathsf{st}_{i,k}}\right\}_{k\in[\ell]}\right)$ to every other party.

Evaluation. To compute the output of the protocol, each party P_i does the following:

1. For each $j \in [n]$, let $\widetilde{\mathsf{lab}}^{j,1} := \left\{\mathsf{lab}^{j,1}_k\right\}_{k\in[\ell]}$ be the labels received from party P_j at the end of Round 2.
2. **For** each t from 1 to T do:
   (a) Parse Φ_t as (i^*, j^*, f, g, h)
   (b) Evaluate the t-th garbled circuit received from party i^*

$$\left((\alpha,\beta,\gamma), \widetilde{\mathsf{lab}}^{i^*,t+1}, \mathsf{ot}_{i^*}[\perp]^{2,t}_{\to i^*,j^*}\right) := \mathsf{Eval}\left(\tilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t}\right)$$

[13] The labels are ordered such that when $\sigma_1 \oplus \sigma_2 = r_{i^*\to i}(h) \oplus \gamma$, the selected label would be $\mathsf{lab}^{i,t}_{h,\gamma}$

(c) Update the h-th bit in the local state: $\mathsf{st}_{i,h} := \gamma \oplus v_{i,h}$.

(d) Evaluate the t-th garbled circuits for each other party.
For each $j \neq i^*$ compute:

$$\left(\left\{\mathsf{lab}_k^{j,t+1}\right\}_{k\in[\ell]\setminus\{h\}}, \mathsf{ot}_j[\perp]^{2,t}_{\to i^*,j^*}\right) := \mathsf{Eval}\left(\tilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t}\right)$$

(e) To compute the label of the h-th input wire, of the $(t+1)$-th garbled circuit, for each party other than i^*, we apply the OT output function OT_3. Recover

$$\left(\left\{\mathsf{lab}_h^{j,t+1}\right\}_{j\in[n]\setminus\{i^*\}}\right) := \mathsf{OT}_3\left(\left\{\mathsf{ot}_j[\perp]^{2,t}_{\to i^*,j^*}\right\}_{j\in[n]}\right)$$

For each $j \neq i^*$ set $\widetilde{\mathsf{lab}}^{j,t+1} := \left\{\mathsf{lab}_k^{j,t+1}\right\}_{k\in[\ell]}$.

3. Compute the output as $\mathsf{post}(i, \mathsf{st}_i)$.

Program P .

Input. st_i
Hardcoded. The index i of the party, the action $\Phi_t = (i^*, j^*, f, g, h)$, the secret value v_i, the OT messages $\left\{\mathsf{ot}_i[\perp]^{2,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{\alpha,\beta\in\{0,1\}}$, a set of labels $\overline{\mathsf{lab}}^{i,t+1} = \{\mathsf{lab}^{i,t+1}_{k,0}, \mathsf{lab}^{i,t+1}_{k,1}\}_{k\in[\ell]}$ and masks $\{r_{i\to j}\}_{j\in[n]\setminus\{i\}}$.

```
1. If $i = i^*$ then:
   (a) Compute $\mathsf{st}_i[h] := r_{i^*\to j^*}(h) \oplus \mathsf{NAND}(\mathsf{st}_i[f], \mathsf{st}_i[g])$, $\alpha := \mathsf{st}_i[f] \oplus v_i[f]$, $\beta := \mathsf{st}_i[g] \oplus v_i[g]$ and $\gamma := \mathsf{st}_i[h] \oplus v_i[h]$.
   (b) Output $\left((\alpha,\beta,\gamma), \left\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_i[k]}\right\}_{k\in[\ell]}, \left\{\mathsf{ot}_i[\perp]^{2,t,\alpha,\beta}_{\to i^*,j^*}\right\}_{j\in[n]\setminus\{i\}}\right)$
2. Else:
   (a) Output $\left(\left\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_i[k]}\right\}_{k\in[\ell]\setminus\{h\}}, \mathsf{ot}_i[\perp]^{2,t,\mathsf{st}_i[f],\mathsf{st}_i[g]}_{\to i^*,j^*}\right)$
```

Fig. 1: Program P

Correctness. An important property of the protocol is that $\forall i, j \in [n]$ and $k \in \ell$, we have $\mathsf{st}_i[k] \oplus v_i[k] = \mathsf{st}_j[k] \oplus v_j[k]$. This is denoted by a value st^*, which we shall refer to as the *global invariant*. In addition, the transcript of the execution in the computation phase is denoted by $\mathsf{Z} \in \{0,1\}^T$. Correctness of the protocol in [16] follows from this global invariant property and the structure of v_i.

From the correctness of the multiparty homomorphic OT, the difference from the protocol in [16] arises when there exists $t \in [T]$ such that $\Phi_t = (\cdot, j, \cdot, \cdot, h)$ such that $j \neq \perp$. Or in other words, when there is a private message to be sent.

In this case, every P_i for $i \in [n] \setminus \{j\}$ sets their respective state $\mathsf{st}_i[h]$ to be $r_{i \to j} \oplus \delta$ where δ is the computation of the NAND functionality, and $r_{i \to j}$ is the mask selected by P_j. From the structure of v_i, for every $i \in [n] \setminus \{j\}, v_i[h] = 0$. On the other hand, P_j updates its state to be $\mathsf{st}_j[h] = \delta$, but from the structure of v_j, we have $v_j[h] = r_{i \to j}$. Thus this maintains the global invariant, $\forall i, j \in [n]$ and $k \in \ell$, we have $\mathsf{st}_i[k] \oplus v_i[k] = \mathsf{st}_j[k] \oplus v_j[k]$.

In addition, since P_j knows $v_j[h]$ in the first round, it can compute the OT receiver message in the first round to subsequently use position h in the protocol. But this is not true for P_i, which is why we incorporate the process of "copying" the bit sent to get around this issue (see subsection 4.1).
The proof of our construction can be found in the full version of our paper.

4.3 Achieving Function-Delayed Property

A conforming protocol Φ is defined by computation steps or *actions* $\Phi_i, \ldots, \Phi_T$ where T is the total number of rounds of this conforming protocol. The pre-processing phase in [16] depends only on T, and is otherwise independent of Φ. We shall leverage this fact to construct protocols for functions that require at most T rounds in the conforming protocol. The function itself can be decided after the pre-processing phase, but must be fixed prior to the computation phase.

An action for a given round t is denoted by a five-tuple (i, f, g, h), where $i \in [n], j \in [n] \cup \{\bot\} f, g, h \in [\ell]$. Given that the state is of length ℓ[14], there can be at most $n \cdot (n+1) \cdot \ell^3$ actions. While there are further restrictions on the choices of (f, g, h), we are satisfied with a loose upper bound. When we compress the protocol, as in [16] , we seem to run into a problem since we send messages for the computation phase in the first round of the compressed protocol, prior to the function being decided.

To account for this, we compute first round OT messages for all possible actions in each round.

For instance, party P_i computes receiver OT messages as follows: $\forall j \in [n] \cup \{\bot\}, f, g, h \in [\ell], \forall \alpha, \beta \in \{0, 1\}$ $\mathsf{ot}^{i,j,f,g,h}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}^R_1\left(v_{i,h} \oplus \mathsf{NAND}\left(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta\right)\right)$ Similarly P_i computes the first round OT messages when it takes the roles of the special designate sender, and the sender. These OT messages are indexed by the tuple (i, j, f, g, h). Thus for each round t, there are $4 \cdot n \cdot (n+1) \cdot \ell^3$ (polynomially many) first round OT messages that are computed. These are sent to the respective parties in the first round.

By the second round, when the parties are creating the garbled circuit they are aware of the function Φ being computed. Let the action in the t-th round be $(\widehat{i}, \widehat{j}, \widehat{f}, \widehat{g}, \widehat{h})$. Thus, when party P_i is preparing its garbled circuit, it will compute its second round OT message accordingly.

While we have described how to achieve the function delayed property in our protocol, the same ideas hold for the protocol in [16]. In fact, we will use the function delayed property of both our protocol, and that of [16] to achieve

[14] It is typically polynomial in the security parameter.

subsequent results. Further discussion, and the security sketch can be found in the full version.

5 Guaranteed Output Delivery: Fail-Stop Adversaries

In this section we describe a general compiler to get a two-round MPC protocol with guaranteed output delivery against semi-malicious fail stop adversaries, from any 2 round semi-malicious MPC protocol that satisfies the delayed function property and only uses broadcast channels.

Overview. A semi-malicious fail stop adversary may choose to abort at any point in the protocol. To achieve security with guaranteed output delivery, we want to implement a mechanism that enables the honest parties to continue the execution, even if some parties abort prematurely. In a two-round protocol, a corrupted party might choose to abort either in the first round or in the second round. If a party aborts in the first round, the honest parties should be able to alter the functionality and continue execution while ignoring its input. However, if a party only aborts in the second round, we cannot ignore its input because such a protocol would clearly not be secure.[15] Let us say that a party is "active", if it does not abort in the first round. In order to achieve guaranteed output delivery, we need to make sure that the honest parties have sufficient information about the input of all the active parties (in some encoded manner) by the end of the first round, so that even if an active party aborts in the second round, the honest parties can still include its input in the computation of the output.

Let us first focus on adversaries who only abort in the first round. In order to give the honest parties enough liberty to modify the functionality in case some parties abort, a secure protocol with guarantee of output must have a *delayed function* property, namely, where the first round message of an honest party is independent of the function and the number of parties. Indeed, for this reason, our starting point is a two-round semi-malicious protocol with delayed function property.

In order to handle adversaries who abort in the second round, our main idea is to require each party to send, in the first round itself, a garbled circuit of an *augmented* second-round next-message function. This augmented next-message function takes a list of active and inactive parties as input and computes second round messages for the appropriate functionality (namely, where the inputs of the inactive parties are set to some default values). To enable the honest parties to continue execution in the second round, we also require each party to send $(t+1, n)$ secret shares of all the labels for its garbled circuit over private channels (in particular, each party only receives one of the shares for each label). At the end of the first round, each party prepares of list of active and inactive parties based on who aborted the protocol. In the second round, each party simply

[15] Indeed, such a protocol would allow an adversary to "ignore" the input of one or more honest parties and learn multiple outputs, which would clearly break security.

broadcasts the appropriate shares for each garbled circuit, based on its list of active and inactive parties. Since we use a $(t+1,n)$ secret sharing scheme, even if some parties abort in the second round, the honest parties have sufficient information to compute the output.

Finally, we remark that our techniques can be seen as a generalization of the techniques used by Gordon et. al. in [20], who constructed a three round protocol with guaranteed output delivery using threshold fully homomorphic encryption with special properties. In contrast, we develop a general compiler using only one-way functions (or without any assumptions if the next message function of the underlying protocol is in NC^1).

5.1 Our Construction

Building Blocks. The main primitives required in this construction are:

1. A two-round semi-malicious MPC protocol Φ, with delayed function property that only uses broadcast channels.
2. An adaptive garbling scheme $(\mathsf{AdapGarble}, \mathsf{AdapEval})$ for circuits.
3. A threshold secret sharing scheme. We denote this by $\mathsf{SS}(\mathsf{Share}, \mathsf{Reconstruct})$.

Next, we establish some notations that are used in our construction.

Active Parties. For any two-round semi-malicious protocol Φ, we say that a party is 'active' in an execution of Φ, if it does not abort in the first round. Let $\mathsf{active} \in \{0,1\}^n$ be an $n-$bit binary string that denotes which parties are active in the last round of the protocol. For each $i \in [n]$, we set $\mathsf{active}_i := 1$, if party P_i is active and $\mathsf{active}_i := 0$ otherwise.

Augmented Next Message Function. Let Φ be a 2 round MPC protocol that supports delayed function property (i.e, where the first round messages of each honest party is independent of the function). Let $\mathsf{Msg}^j_\Phi(i, x_i, \mathsf{trans}^{j-1}_\Phi; r_i)$ be the next message function for round j. It takes as input, party index i, it's input x_i, previous round transcripts $\mathsf{trans}^{j-1}_\Phi$ and randomness r_i. Delayed function property ensures that $\mathsf{Msg}^1_\Phi(\cdot,\cdot,\cdot;\cdot)$ is independent of the function $\mathcal{F}$ that the MPC computes and only $\mathsf{Msg}^2_\Phi(\cdot,\cdot,\cdot;\cdot)$ depends on it.
We define an 'augmented' second round next message function, that additionally takes a list of active parties (active) in the protocol as input, and computes the second round messages. More specifically, this augmented next message function has the function $\mathcal{F}$ and default inputs for all parties hard coded inside it. Given a list active, it substitutes the actual input of an inactive party with this default input in $\mathcal{F}$ and computes the second round messages. We denote this augmented second round next message function by $\mathsf{AugMsg}^2_\Phi(i, x_i, \mathsf{trans}^1_\Phi, \mathsf{active}; r_i)$.

Theorem 6. *Let Φ be any two-round semi-honest (resp., semi-malicious) broadcast channel MPC protocol with delayed function property,* $(\mathsf{AdapGarble}, \mathsf{AdapEval})$ *be an adaptively secure garbling scheme for circuits and* $\mathsf{SS}(\mathsf{Share}, \mathsf{Reconstruct})$ *be a threshold secret sharing scheme. There exists a general compiler that transforms Φ into a two-round protocol that achieves guaranteed output delivery against non-rushing fail-stop (resp., rushing, semi-malicious fail-stop) adversaries.*

A few corollaries of the above theorem are in order:

- The protocol from theorem 5 (with the function-delayed property) can be easily transformed into a protocol that only uses broadcast channels in the BPK model [8]. Applying the compiler from theorem 6 to this protocol, we obtain a two-round broadcast-channel MPC protocol in the BPK model that achieves guaranteed output delivery against any $t < \frac{n}{2}$ (semi-malicious) fail-stop corruptions.
- The semi-honest construction from [16] can be modified to support the function-delayed property as discussed in section 4.3. Applying the compiler from theorem 6 to this modified construction, we obtain a two round MPC protocol over private point to point channels, that achieves guaranteed output delivery against any $t < \frac{n}{2}$ non-rushing fail-stop (resp., rushing, semi-malicious fail-stop) corruptions in the plain model.

We now describe our protocol in detail. For simplicity, we describe a compiler that uses both broadcast and private channels. But since this protocol is only secure against fail-stop adversaries, it can be easily modified to work only using private channels in the plain model. If the underlying protocol works in the (bare) public key model, then the compiler can be modified to work only using broadcast channels. We specify these modifications in the protocol description.

Protocol. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties in the protocol. Let $\{x_1, \ldots, x_n\}$ be their respective inputs and $\{r_1, \ldots, r_n\}$ be their respective randomness used in the underlying protocol Φ. If the underlying program assumes existence of the BPK model, then let $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ and $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$ be the respective public and secret keys of the parties. Let λ be the security parameter.

Round 1. Each party P_i does the following in the first round:

1. Computes the first round message Φ_i^1 using its input x_i and randomness r_i, i.e., $\Phi_i^1 := \mathsf{Msg}_\Phi^1(i, x_i, \perp; r_i)$
2. Computes an adaptive garbling of the augmented second round next message function $\mathsf{AugMsg}_\Phi^2[i, x_i; r_i](\cdot, \cdot)$ with it's index i, input x_i and randomness r_i hardcoded inside it. This function only takes the first round transcript (trans_Φ^1) and the list active as input, i.e.,
$(\widetilde{\mathsf{NMF}}_i, \{\mathsf{lab}_i^{w,b}\}_{w\in[\mathsf{inp}],b\in\{0,1\}}) \leftarrow \mathsf{AdapGarble}(1^\lambda, \mathsf{AugMsg}_\Phi^2[i, x_i; r_i])$,
where inp is the length of input to $\mathsf{AugMsg}_\Phi^2[i, x_i; r_i]$.
3. Uses a threshold secret sharing scheme to compute $(t+1, n)$ shares of the input labels, i.e., $\{\mathsf{lab}_{i,1}^{w,b}, \ldots, \mathsf{lab}_{i,n}^{w,b}\}_{w\in\mathsf{inp},b\in\{0,1\}} \leftarrow \mathsf{Share}(1^\lambda, \{\mathsf{lab}_i^{w,b}\}_{w\in\mathsf{inp},b\in\{0,1\}})$
4. Broadcasts $M_i^1 := (\Phi_i^1, \widetilde{\mathsf{NMF}}_i)$ to all other parties.
5. Sends $\{\mathsf{lab}_{i,j}^{w,b}\}_{w\in\mathsf{inp},b\in\{0,1\}}$ to party P_j (for $j \in [n]\backslash\{i\}$) over private channels. (In the BPK model, the message for party P_j is encrypted under pk_j and then sent over the broadcast channel.)

At the end of Round 1. Each party P_i does the following:

1. **For** j from 1 to n:

   (a) **If** party P_j sent its first round messages, parse M_j^1 as $(\Phi_j^1, \widetilde{\mathsf{NMF}}^j)$ and set $\mathsf{active}_j := 1$
   (b) **If** party P_j aborts in the first round, set $\Phi_j^1 := 0^\ell$, where ℓ is the length of each party's first round message in Φ and set $\mathsf{active}_j := 0$
2. Sets $\mathsf{trans}_\Phi^1 := \{\Phi_j^1\}_{j\in[n]}$.
3. In the BPK model, it decrypts the encrypted labels sent by other parties using its secret key sk_i.

Round 2. Each party P_i does the following in the second round:

1. It sets $z = \mathsf{trans}_\Phi^1 || \mathsf{active}$.
2. For each garbled circuit $\{\widetilde{\mathsf{NMF}}_j\}_{j\in[n]}$, it sends shares for the key and the labels corresponding to active and trans_Φ^1 i.e., $M_i^2 := \{\mathsf{lab}_{j,i}^{w,z[w]}\}_{w\in[\mathsf{inp}],j\in[n]}$. We assume that $\mathsf{lab}_{j,i}^{\mathsf{trans}_\Phi^1||\mathsf{active}} = \bot$ for a party P_j that aborts in the second round.

Output Phase. Let $\mathbf{Y}$ be the set of any $t+1$ parties that send first and second round. messages. Each party P_i does the following:

1. **For** $j \in \mathbf{Y}$:
   (a) Parse M_j^2 as $\{\mathsf{lab}_{k,j}^{w,z[w]}\}_{w\in[\mathsf{inp}],k\in[n]}$
   (b) **If** $\mathsf{active}_j = 1$, reconstruct the input labels and evaluate the garbled circuit, i.e., $\{\mathsf{lab}_j^{w,z[w]}\}_{w\in[\mathsf{inp}]} := \{\mathsf{Reconstruct}(\{\mathsf{lab}_{j,k}^{w,z[w]}\}_{k\in\mathbf{Y}})\}_{w\in[\mathsf{inp}]}$ and $\Phi_j^2 := \mathsf{AdapEval}(\widetilde{\mathsf{NMF}}_j, \{\mathsf{lab}_j^{w,z[w]}\}_{w\in[\mathsf{inp}]})$
2. Let $\mathbf{A}$ be the set of 'active' parties in the protocol.
3. Runs the output phase of Φ, $\mathsf{Out}_\Phi(\{\Phi_j^2\}_{j\in\mathbf{A}})$ to learn the output.

Remark. The above compiler can also be modified to get a three-round protocol in the plain model only assuming one-way functions (or unconditionally, for NC^1 computations). This main idea is to divide the first round messages of the above compiler into two. More specifically, the parties exchange their first round messages of Φ (which may include private channel messages) in the first round. In the second round, each party P_i computes an adaptive garbled circuit on the augmented second round next message function $\mathsf{AugMsg}_\Phi^2[i, x_i, \mathsf{trans}_{\Phi,i}^1; r_i](.)$ of Φ, that has it's index i, input x_i, it's first round transcript $\mathsf{trans}_{\Phi,i}^1$ and randomness r_i hard wired inside it. Since the first round messages are already hard-wired, this garbled circuit only takes the list of 'active' parties as input. Each party also secret shares all the input labels to this garbled circuit. The third round proceeds similar to the second round in the above compiler, with the only difference that all the parties who participate in the first two rounds constitute the list of 'active' parties. Instantiating this modified compiler with the protocol from theorem 5 (with the function-delayed property), we get the following corollary:

Corollary 4. *Assuming one-way functions (or unconditionally, for NC^1 computations), there exists a three round MPC protocol that achieves guaranteed output delivery against any $t < \frac{n}{2}$ (semi-malicious) fail-stop corruptions.*

6 Guaranteed Output Delivery: Malicious Adversaries

In this section we describe a general compiler to get a three-round malicious MPC protocol with guaranteed output delivery in the plain model from our two round semi-malicious MPC protocol with guaranteed output delivery.

Overview In order to compile our semi-malicious protocol from the previous section into a maliciously secure one, we use the standard "commit-and-prove" methodology of [19], where the adversary initially commits to his input and randomness and then gives a zero-knowledge proof of "honest behavior" together with each round of the underlying semi-malicious protocol. We note, however, that implementing this methodology in the setting of guaranteed output delivery requires extra care. In particular, we need to ensure that all the honest parties have a *consistent view of which parties aborted in a given round* since the behavior of an honest party in the next round depends upon this view.

Note that if the underlying semi-malicious protocol uses private channels, then a party may need to prove different statements to different parties in order to establish honest behavior, and in particular, the statement being proven by party i to party j may not be known to another party k. This presents a problem in ensuring that the honest parties have consistent views (of the form as discussed above). Therefore, as a first step, we transform the two-round semi-malicious protocol into a three-round protocol that only uses broadcast channels, using public-key encryption. However, if the underlying semi-malicious protocol works in the (bare) public key model and only uses broadcast channels, we can transform this two-round semi-malicious protocol into a three-round semi-malicious protocol in the plain model by exchanging public keys in the first round.

Next we note that zero-knowledge proofs with black-box simulation are known to require at least four rounds [18]. To overcome this lower bound, and in order to obtain a three round maliciously secure protocol, we leverage the fact that we are in the honest majority setting. Towards this, we define a new notion of *multi-verifier zero-knowledge* (MVZK) proofs. Briefly, an MVZK proof system is an interactive multiparty protocol between a prover and multiple verifiers. Similar to standard ZK, we require MVZK to achieve soundness and zero knowledge properties. In particular, we require the soundness property to hold as long as the honest verifiers constitute a majority. Similarly, we require ZK property to hold as long as the honest prover and the honest verifiers, together constitute a majority. In order to use MVZK in our setting, we also require a "strong completeness" property which guarantees that any set of dishonest verifiers (who constitute a minority) cannot prevent the honest verifiers from accepting a proof from an honest prover.

We implement our compiler using two-round MVZK arguments with a delayed input property, namely, where the first round messages of the honest parties are independent of the statement. We note that while our two-round MPC protocol from Section 4 can be used to construct a two-round MVZK without the aforementioned strong completeness property; therefore it does not suffice here. Instead, in the full version, we give a construction of two round delayed-input

MVZK (that achieves strong completeness) from Zaps, following the construction of multi-CRS NIZKs by Groth and Ostrovsky [21]. We then use this MVZK to implement our compiler.

6.1 Our construction

Building Blocks. The main primitives required in this construction are:

1. A two-round MPC protocol Π that achieves guaranteed output delivery against semi-malicious fail-stop adversaries. Let $\mathsf{Msg}^j_{\Pi}(i, x_i, \{\mathsf{trans}^k_{\Pi}\}_{k\in[j-1]}; r_i)$ be the next message function for round j. It takes as input, index i of the party, it's input x_i, previous round transcripts $\{\mathsf{trans}^k_{\Pi}\}_{k\in[j-1]}$ and randomness r_i.
2. A threshold secret sharing scheme. We denote this by $\mathsf{SS} := (\mathsf{Share}, \mathsf{Reconstruct})$.
3. Two-round delayed-input multi-verifier zero-knowledge arguments $\mathsf{mvzk} := (\mathsf{P}^1_{\mathsf{mvzk}}, \mathsf{V}^1_{\mathsf{mvzk}}, \mathsf{P}^2_{\mathsf{mvzk}}, \mathsf{V}^2_{\mathsf{mvzk}}, \mathsf{Verify}_{\mathsf{mvzk}})$.
4. A public-key encryption scheme $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

Theorem 7. *Let Π be a two-round MPC protocol with guaranteed output delivery against semi-malicious fail-stop adversaries,* $(\mathsf{AdapGarble}, \mathsf{AdapEval})$ *be an adaptively secure garbling scheme,* $\mathsf{mvzk} := (\mathsf{P}^1_{\mathsf{mvzk}}, \mathsf{V}^1_{\mathsf{mvzk}}, \mathsf{P}^2_{\mathsf{mvzk}}, \mathsf{V}^2_{\mathsf{mvzk}}, \mathsf{Verify}_{\mathsf{mvzk}})$ *be a delayed-input MVZK argument system and* $\mathcal{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme. Then there exists a general compiler that transforms Π into a three round protocol with guaranteed output delivery against malicious adversaries.*

Applying the compiler from theorem 7 to the two-round BPK model protocol from Section 5, we get a three round protocol based on Zaps and public-key encryption. Next, we describe the protocol in detail:

Protocol. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties in the protocol and let $\{x_1, \ldots, x_n\}$ be their respective inputs. Let λ be the security parameter.

Round 1. Each party P_i does the following in the first round:

1. Generates a key pair for the public key encryption scheme, i.e., $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Gen}(1^\lambda; q_i)$
2. Computes the first round prover message of MVZK and verifier messages for all other parties, i.e., $\mathsf{pMsg}^{1,i} \leftarrow \mathsf{P}_{\mathsf{mvzk}}(1^\lambda)$ and $\{\mathsf{vMsg}^{1,j}_i\}_{j\in[n]\setminus i} \leftarrow \{\mathsf{V}_{\mathsf{mvzk}}(1^\lambda, i)\}_{j\in[n]\setminus i}$
3. Broadcasts $M^1_i := (\mathsf{pk}_i, \mathsf{pMsg}^{1,i}, \{\mathsf{vMsg}^{1,j}_i\}_{j\in[n]\setminus i})$ to all other parties.

At the end of Round 1. Each Party P_i for $i \in [n]$ does the following:

1. **For** j from 1 to n:
   (a) **If** Party P_j sends its first round messages, parse M^1_j as $(\mathsf{pk}_j, \mathsf{pMsg}^{1,j}, \{\mathsf{vMsg}^{1,k}_j\}_{k\in[n]\setminus j})$
   (b) **Else**, set $\mathsf{pMsg}^{1,j} := \perp$ and $\{\mathsf{vMsg}^{1,k}_j\}_{k\in[n]\setminus j} := \perp$

2. **For** j from 1 to n, set $\mathsf{trans}^{1,j}_{\mathsf{mvzk}} := (\mathsf{pMsg}^{1,j}, \{\mathsf{vMsg}^{1,j}_k\}_{k\in[n]\setminus j})$.

Round 2. Each party P_i does the following in the first round:

1. Computes the first round message Π^1_i using its input x_i and randomness r_i, i.e., $\Pi^1_i := \mathsf{Msg}^1_\Pi(i, x_i, \perp; r_i)$
2. Uses a threshold secret sharing scheme to compute $(t+1, n)$ shares of $X_i = (x_i, r_i)$, i.e., $\{X_{i,1}, \ldots, X_{i,n}\} := \mathsf{Share}(1^\lambda, X_i; s_i)$
3. For each $j \in [n]$, it encrypts the share $X_{i,j}$ under public key pk_j, i.e., $c_{i,j} := \mathsf{Enc}(\mathsf{pk}_j, X_{i,j}; t_{i,j})$
4. Proves the following:
   (a) The public key pk_i was honestly generated **AND**
   (b) Each ciphertext $c_{i,j}$ is an honestly computed encryption **AND**
   (c) The first round messages of Π are computed honestly using the input x_i and randomness r_i that were honestly secret shared and each of these shares were honestly encrypted.

   Using the language:

$$\begin{aligned} L =&\{(\Pi^1_i, \{\mathsf{pk}_j\}_{j\in[n]}, \{c_{i,j}\}_{j\in[n]}) \mid \exists (x_i, r_i, s_i, q_i, \{t_{i,j}\}_{j\in[n]}) \\ &\text{s.t. } ((\mathsf{pk}_i, \cdot) = \mathsf{Gen}(1^\lambda; q_i)) \text{ AND } (\Pi^1_i = \mathsf{Msg}^1_\Pi(i, x_i, \perp; r_i)) \\ &\text{AND } (\{X_{i,j}, \ldots, X_{i,n}\} := \mathsf{Share}(1^\lambda, X_i; s_i)) \\ &\text{AND } (\{c_{i,j}\}_{j\in[n]} := \{\mathsf{Enc}(\mathsf{pk}_j, X_{i,j}; t_{i,j})\}_{j\in[n]}))\} \end{aligned}$$

   It computes second round prover messages of mvzk as follows:
   Let $Y_i = (\Pi^1_i, \{\mathsf{pk}_j\}_{j\in[n]}, \{c_{i,j}\}_{j\in[n]})$ and $W_i = (x_i, r_i, s_i, q_i, \{t_{i,j}\}_{j\in[n]})$, i.e., $\mathsf{pMsg}^{2,i} \leftarrow \mathsf{P}_{\mathsf{mvzk}}(\mathsf{trans}^{1,i}_{\mathsf{mvzk}}, Y_i, W_i)$
5. Computes second round verifier messages of mvzk for all other parties for the same language, i.e., $\{\mathsf{vMsg}^{2,j}_i\}_{j\in[n]\setminus i} \leftarrow \{\mathsf{V}_{\mathsf{mvzk}}(i, \mathsf{trans}^{1,j}_{\mathsf{mvzk}})\}_{j\in[n]\setminus i}$
6. Computes another set of first round prover message of MVZK and verifier messages for all other parties, i.e., $\widetilde{\mathsf{pMsg}}^{1,i} \leftarrow \mathsf{P}_{\mathsf{mvzk}}(1^\lambda)$ and $\{\widetilde{\mathsf{vMsg}_i}^{1,j}\}_{j\in[n]\setminus i} \leftarrow \{\mathsf{V}_{\mathsf{mvzk}}(1^\lambda, i)\}_{j\in[n]\setminus i}$
7. Broadcasts $M^2_i := (\Pi^1_i, \{c_{i,j}\}_{j\in[n]}, \mathsf{pMsg}^{2,i}, \{\mathsf{vMsg}^{2,j}_i\}_{j\in[n]\setminus i}, \widetilde{\mathsf{pMsg}}^{1,i}, \{\widetilde{\mathsf{vMsg}_i}^{1,j}\}_{j\in[n]\setminus i})$ to all other parties.

At the end of Round 2. Each party does the following:

1. **For** j from 1 to n:
   (a) **If** Party P_j sent its first and second round messages, parse M^2_j as $(\Pi^2_i, \{c_{j,k}\}_{j\in[n]}, \mathsf{pMsg}^{2,j}, \{\mathsf{vMsg}^{2,k}_i\}_{k\in[n]\setminus j}, \widetilde{\mathsf{pMsg}}^{1,j}, \{\widetilde{\mathsf{vMsg}_i}^{1,k}\}_{k\in[n]\setminus j})$
   (b) **Else** set $\mathsf{pMsg}^{2,j}, \widetilde{\mathsf{pMsg}}^{1,j} := \perp$ and $\{\mathsf{vMsg}^{2,k}_j, \widetilde{\mathsf{vMsg}_j}^{1,k}\}_{k\in[n]\setminus j} := \perp$
2. **For** j from 1 to n:
   (a) Set $Y_j := (\Pi^1_j, \{\mathsf{pk}_k\}_{k\in[n]}, \{c_{j,k}\}_{k\in[n]})$
   (b) **If** $\mathsf{Verify}_{\mathsf{mvzk}}(i, \{\mathsf{trans}^{r,j}_{\mathsf{mvzk}}\}_{r\in[2]}, Y_j)$=1, decrypt $c_{j,i}$, i.e., $m_{j,i} := \mathsf{Dec}(\mathsf{sk}_i, c_{j,i})$ and parse $m_{j,i}$ as $X_{j,i}$

(c) **Else:**

i. Set $\mathit{\Pi}_j^1 := 0^\ell$, where ℓ is the length first round messages in $\mathit{\Pi}$.

ii. Set $\widetilde{\mathsf{pMsg}}^{1,j} := \perp$ and $\{\widetilde{\mathsf{vMsg}_j}^{1,k}\}_{k\in[n]\setminus j} := \perp$

3. **For** j from 1 to n, set $\mathsf{trans}^{1,j}_{\widetilde{\mathsf{mvzk}}} := (\widetilde{\mathsf{pMsg}}^{1,j}, \{\widetilde{\mathsf{vMsg}_k}^{1,j}\}_{k\in[n]\setminus j})$.
4. Set $\mathsf{trans}^1_{\mathit{\Pi}} := \{\mathit{\Pi}^1_j\}_{j\in[n]}$.

Round 3. Each party P_i does the following in the third round:

1. Computes second round messages of $\mathit{\Pi}$, i.e., $\mathit{\Pi}_i^2 := \mathsf{Msg}^2_{\mathit{\Pi}}(i, x_i, \mathsf{trans}^1_{\mathit{\Pi}}; r_i)$
2. Proves that the second round message $\mathit{\Pi}_i^2$ was computed honestly using the language $L = \{(\mathit{\Pi}_i^2, \mathsf{trans}^1_{\mathit{\Pi}}) \mid \exists(x_i, r_i) \text{ s.t. } \mathit{\Pi}_i^2 := \mathsf{Msg}^2_{\mathit{\Pi}}(i, x_i, r_i, \mathsf{trans}^1_{\mathit{\Pi}})\}$. It computes second round prover messages of mvzk as follows; Let $Z_i = (\mathit{\Pi}_i^2, \mathsf{trans}^1_{\mathit{\Pi}})$ and $W_i = (x_i, r_i)$, i.e., $\widetilde{\mathsf{pMsg}}^{2,i} \leftarrow \mathsf{P}_{\mathsf{mvzk}}(\mathsf{trans}^{1,i}_{\widetilde{\mathsf{mvzk}}}, Z_i, W_i)$
3. Computes second round verifier messages of mvzk for all other parties for the same language, i.e., $\{\widetilde{\mathsf{vMsg}_i}^{2,j}\}_{j\in[n]\setminus i} \leftarrow \{\mathsf{V}_{\mathsf{mvzk}}(i, \mathsf{trans}^{1,j}_{\widetilde{\mathsf{mvzk}}})\}_{j\in[n]\setminus i}$
4. Broadcasts $M_i^3 := (\mathit{\Pi}_i^2, \widetilde{\mathsf{pMsg}}^{2,i}, \{\widetilde{\mathsf{vMsg}_i}^{2,j}\}_{j\in[n]\setminus i})$ to all other parties.

Output Phase. Each party P_i does the following:

1. **For** j from 1 to $[n]$
   (a) **If** party P_j sent a message in the third round, parse M_i^3 as $(\mathit{\Pi}_j^2, \widetilde{\mathsf{pMsg}}^{2,j}, \{\widetilde{\mathsf{vMsg}_j}^{2,k}\}_{k\in[n]\setminus j})$
   (b) **Else** set $\mathit{\Pi}_j^2 := \perp$, $\widetilde{\mathsf{pMsg}}^{2,j} := \perp$ and $\{\widetilde{\mathsf{vMsg}_j}^{2,k}\}_{k\in[n]\setminus j} := \perp$
2. **For** j from 1 to n:
   (a) Set $Z_j := (\mathit{\Pi}_j^2, \mathsf{trans}^1_{\mathit{\Pi}})$
   (b) **If** $\mathsf{Verify}_{\mathsf{mvzk}}(i, \{\mathsf{trans}^{r,j}_{\widetilde{\mathsf{mvzk}}}\}_{r\in[2]}, Z_j) = 0$, set $\mathit{\Pi}_j^2 := \perp$
3. Set $\mathsf{trans}^2_{\mathit{\Pi}} = \{\mathit{\Pi}_j^2\}_{j\in[n]}$ and run the output phase of $\mathit{\Pi}$, $\mathsf{Out}_{\mathit{\Pi}}(\mathsf{trans}^1_{\mathit{\Pi}}, \mathsf{trans}^2_{\mathit{\Pi}})$ to learn the output.

Acknowledgments. The second author would like to thank Ignacio Cascudo for helpful discussions. This research was supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213, and a subaward from NSF CNS-1414023.

References

1. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (Apr 2012)
2. Badrinarayanan, S., Goyal, V., Jain, A., Kalai, Y.T., Khurana, D., Sahai, A.: Promise zero knowledge and its applications to round optimal mpc. In: CRYPTO (2018), https://eprint.iacr.org/2017/1088

3. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC. pp. 503–513. ACM Press (May 1990)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (Dec 2012)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988)
6. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 500–532. Springer, Heidelberg (Apr / May 2018)
7. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC. pp. 103–112. ACM Press (May 1988)
8. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: 32nd ACM STOC. pp. 235–244. ACM Press (May 2000)
9. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC. pp. 11–19. ACM Press (May 1988)
10. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th ACM STOC. pp. 364–369. ACM Press (May 1986)
11. Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 466–485. Springer, Heidelberg (Dec 2014)
12. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (Aug 2005)
13. Dwork, C., Naor, M.: Zaps and their applications. In: 41st FOCS. pp. 283–293. IEEE Computer Society Press (Nov 2000)
14. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS. pp. 308–317. IEEE Computer Society Press (Oct 1990)
15. Garg, S., Mukherjee, P., Pandey, O., Polychroniadou, A.: The exact round complexity of secure computation. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 448–476. Springer, Heidelberg (May 2016)
16. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 468–499. Springer, Heidelberg (Apr / May 2018)
17. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 178–193. Springer, Heidelberg (Aug 2002)
18. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM J. Comput. 25(1), 169–192 (1996), https://doi.org/10.1137/S0097539791220688
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
20. Gordon, S.D., Liu, F.H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (Aug 2015)

21. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 323–341. Springer, Heidelberg (Aug 2007)
22. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 132–150. Springer, Heidelberg (Aug 2011)
23. Ishai, Y., Kumaresan, R., Kushilevitz, E., Paskin-Cherniavsky, A.: Secure computation with minimal interaction, revisited. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 359–378. Springer, Heidelberg (Aug 2015)
24. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st FOCS. pp. 294–304. IEEE Computer Society Press (Nov 2000)
25. Ishai, Y., Kushilevitz, E., Paskin, A.: Secure multiparty computation with minimal interaction. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 577–594. Springer, Heidelberg (Aug 2010)
26. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (Aug 2004)
27. Lapidot, D., Shamir, A.: Publicly verifiable non-interactive zero-knowledge proofs. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 353–365. Springer, Heidelberg (Aug 1991)
28. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 591–602. ACM Press (Oct 2015)
29. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery 22(11), 612–613 (Nov 1979)
30. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)