The Round Complexity of Secure Computation Against Covert Adversaries

Arka Rai Choudhuri* Vipul Goyal† Abhishek Jain‡

Abstract

We investigate the exact round complexity of secure multiparty computation (MPC) against *covert* adversaries who may attempt to cheat, but do not wish to be caught doing so. Covert adversaries lie in between semi-honest adversaries who follow protocol specification and malicious adversaries who may deviate arbitrarily.

Recently, two round protocols for semi-honest MPC and four round protocols for malicious-secure MPC were constructed, both of which are optimal. While these results can be viewed as constituting two end points of a security spectrum, we investigate the design of protocols that potentially span the spectrum.

Our main result is an MPC protocol against covert adversaries with variable round complexity: when the detection probability is set to the lowest setting, our protocol requires two rounds and offers same security as semi-honest MPC. By increasing the detecting probability, we can increase the security guarantees, with round complexity five in the extreme case. The security of our protocol is based on standard cryptographic assumptions.

We supplement our positive result with a negative result, ruling out *strict* three round protocols with respect to black-box simulation.

*Johns Hopkins University. achoud@cs.jhu.edu

†Carnegie Mellon University and NTT Research. goyal@cs.cmu.edu

‡Johns Hopkins University. abhishek@cs.jhu.edu

Contents

1 Introduction

The ability to securely compute on private datasets of individuals has wide applications of tremendous benefits to society. Secure multiparty computation (MPC) [Yao86, GMW87] provides a solution to the problem of computing on private data by allowing a group of parties to jointly evaluate any function over their private inputs in such a manner that no one learns anything beyond the output of the function.

Since its introduction nearly three decades ago, MPC has been extensively studied with respect to various complexity measures. In this work, we focus on the *round complexity* of MPC. This topic has recently seen a burst of activity along the following two lines:

- *Semi-honest adversaries*: Recently, several works have constructed *two* round MPC protocols [GGHR14, MW16, GS17, GS18, BL18] that achieve security against semi-honest adversaries who follow protocol specifications but may try to learn additional information from the protocol transcript. The round complexity of these protocols is optimal [HLP11].
- *Malicious adversaries*: A separate sequence of works have constructed *four* round MPC protocols [ACJ17, BGJ+18, HHPV18, CCG+19] that achieve security against malicious adversaries who may arbitrarily deviate from the protocol specification. The round complexity of these protocols is optimal w.r.t. black-box simulation and crossing this barrier via non-black-box techniques remains a significant challenge.

These two lines of work can be viewed as constituting two ends of a security spectrum: on the one hand, the two round protocols do not offer any security whatsoever against malicious behavior, and may therefore not be suitable in many settings. On the other hand, the four round protocols provide very strong security guarantees against all malicious strategies.

Our Question. The two end points suggest there might be interesting intermediate points along the spectrum that provide a trade-off between security and round complexity. In this work, we ask whether it is possible to devise protocols that span the security spectrum (as opposed to just the two end points), but do not always require the cost of four rounds. More specifically, is it possible to devise MPC protocols with a "tunable parameter" ϵ such that the security, and consequently, the round complexity of the protocol can be tuned to a desired "level" by setting ϵ appropriately?

Background: MPC against Covert Adversaries. Towards answering this question, we look to the notion of MPC against *covert adversaries*, first studied by Aumann and Lindell [AL07]. Intuitively, covert adversaries may deviate arbitrarily from the protocol specification in an attempt to cheat, but do not wish to be "caught" doing so. As Aumann and Lindell argue, this notion closely resembles real-world adversaries in many commercial, political and social settings, where individuals and institutions are potentially willing to cheat to gain advantage, but do not wish to suffer the loss of reputation or potential punishment associated with being caught cheating. Such adversaries may weigh the risk of being caught against the benefits of cheating, and may act accordingly.

Aumann and Lindell model security against covert adversaries by extending the real/ideal paradigm for secure computation. Roughly, the ideal world adversary (a.k.a. simulator) is allowed to send a special cheat instruction to the trusted party. Upon receiving such an instruction, the trusted party hands all the honest parties' inputs to the adversary. Then, it tosses coins and with probability ϵ, announces to the honest parties that cheating has taken place. This refers to the case where the adversary's cheating has been *detected*. However, with probability $1 - \epsilon$, the trusted party does not announce that cheating has taken place. This refers to the case where the adversary's cheating has not been detected.[1]

[1] The work of [AL07] also provides two other formulations of security. In this work, we focus on the above formulation.

Our Results. Towards answering the aforementioned question, we investigate the round complexity of MPC against covert adversaries. We provide both positive and negative results on this topic.

Positive Result. Our main result is an MPC protocol against covert adversaries with round complexity $2 + 5 \cdot q$, where q is a protocol parameter that is a function of the cheating detection parameter ϵ. When $q = 0$, our protocol requires only two rounds and provides security against semi-honest adversaries. By increasing q, we can increase the security guarantees of our protocol, all the way up to $q = 1$, when our protocol requires seven rounds. We note, however, that by appropriately parallelizing rounds, the round complexity in this case can in fact be decreased to five (see the technical sections for more details). Our protocol relies on injective one-way functions, two-round oblivious transfer and Zaps (two-message public coin witness indistinguishable proofs introduced in [DN00]).

As mentioned earlier, in this work, we consider the aforementioned formulation of security against covert adversaries where the adversary always learns the honest party inputs whenever it decides to cheat. An interesting open question is whether our positive result can be extended to the stronger model where the adversary only learns the honest party inputs when its cheating goes undetected.

Negative Result. We supplement our positive result with a negative result. Namely, we show that security against covert adversaries is impossible to achieve in *strict* three rounds with respect to black-box simulation. We prove this result for the zero-knowledge proofs functionality in the simultaneous-broadcast model for MPC.

1.1 Technical Overview

Negative Result. We start by briefly summarizing the main ideas underlying our negative result.

Recall that a black-box simulator works by rewinding the adversary in order to simulate its view. In a three round protocol over simultaneous-broadcast model, the simulator has three potential opportunities to rewind – one for each round. In order to rule out three round ZK against covert adversaries, we devise a covert verifier strategy that foils all rewinding strategies in every round.

As a starting point, let us consider the first round of the protocol. In this case, a rushing covert verifier can choose a fresh new first round message upon being rewound, which effectively leads to protocol restart. Next, let's consider the second round. Here, the covert verifier can choose to not be rushing, and instead compute its second round message independent of the prover's message in the second round. That is, upon being rewound, the verifier simply re-sends the same message again and again, thereby making the rewinding useless. Indeed, the main challenge is in ruling out successful rewinding in the third round.

One potential strategy for the covert verifier is to simply always *abort* in the third round. While such a strategy would indeed work if our goal was to rule out ZK against *malicious* verifiers, it does not quite work in the covert setting. This is because in this case, the simulator can simply request the prover's input (namely, the witness) from the trusted party and then use it to generate the view. Note that, this strategy works because protocol abort counts as cheating behavior on behalf of the verifier.[2]

To rule out such "trivial" simulation, we devise a covert verifier strategy that "cheats only if cheated with." More specifically, our covert verifier behaves honestly in the third round if it receives an accepting message from the prover; however, upon receiving a non-accepting message, it simply aborts. Clearly, this verifier never aborts in the real world. However, when the simulator runs this verifier, it may always find that the verifier aborts. Indeed, without the knowledge of the prover's witness, and no advantage of rewinding

[2]One may ask whether modeling protocol abort as cheating could lead to scenarios where honest parties, in the presence of network interruptions, are labeled as adversarial. We imagine that in such a scenario, one could put in place countermeasures, where, e.g., a party is allowed to prove (in zero knowledge) honest behavior in the protocol (up to the point of purported abort) to other parties in order to defend itself.

in the first two rounds, the simulator would not be able to generate an accepting third message. Moreover, querying the trusted party on the prover's input to generate the view would lead to a skewed distribution.

To establish a formal proof of impossibility, we show that the existence of a simulator can be used to break soundness of the ZK protocol. We refer the reader to Section 5 for more details.

Positive Result. We now summarize the main ideas underlying our positive result.

The main insight behind our protocol is to perform *random checking* to ensure that the parties are behaving honestly. Our strategy differs from prior works on MPC against covert adversaries in the following manner: while prior works *always* check for malicious behavior in a manner such that the adversary is caught with certain probability, *we only initiate the checking procedure with certain probability*. More specifically, in our protocol, each party initiates the checking procedure with probability q. Indeed, this is what leads to the variable round complexity of our protocol.

Armed with the above insight, we implement our protocol as follows:

- In the first round, the parties execute the first round of a two-round semi-malicious MPC protocol (semi-malicious security proposed by [AJL+12] considers semi-malicious attackers that follow the protocol specification, but may adaptively choose arbitrary inputs and random tapes for computing each of its messages). In addition, the parties also commit to their inputs and random tapes, and exchange the first round message of Zaps.

- In the second round, each party tosses a coin to determine with probability q, whether or not to vote for initiating the *verification mode*. If its vote is "yes," it announces it to the other parties. Otherwise, it computes and sends the second round of semi-malicious MPC together with a Zap proving that the second round message was computed honestly.

- If the total vote count is non-zero, then the parties execute the verification phase where each party proves in zero-knowledge that it behaved honestly in the first round using the committed input and randomness. To avoid malleability concerns, we in fact use simulation-extractable zero knowledge (SE-ZK). When the verification mode ends (or if it was never executed), the parties execute the last round of the protocol. Here, the parties who did not already vote for the verification phase complete the second round of the two-round semi-malicious MPC, and additionally prove via Zap that the second round was computed honestly.

- To enable simulation, we establish an alternative "trapdoor" witness for the Zap as follows. The parties are required to commit to 0 in the first round, and prove in the verification mode that they indeed committed to 0. Now, the trapdoor mode for Zap simply amounts to proving that a party had in fact committed to 1.

At a first glance, the above protocol suggests a seemingly simple simulation strategy:

- The simulator starts by simulating the first round of the semi-malicious MPC, and commits to 1 (instead of 0).

- Next, the simulator *always* initiates the verification phase, where it uses the simulator-extractor of SE-ZK to simulate the proofs as well as extract the adversary's input and randomness.

- Finally, it simulates the second round of the semi-malicious MPC and completes the Zap using the trapdoor witness.

A closer inspection, however, reveals several problems with the above strategy. A minor issue is that the above strategy leads to a skewed distribution since the verification phase is executed with probability 1.

This, however, can be easily resolved as follows: after completing extraction, the simulator rewinds to the end of the first round and then uses random coins to determine whether or not to execute the verification phase.

A more important issue is that the simulator may potentially fail in extracting adversary's input and randomness, possibly because of premature aborts by the adversary. Since aborts constitute cheating behavior, a natural recourse for the simulator in this case is to send the cheat signal to the ideal functionality. Now, upon learning the honest party inputs, the simulator rewinds to the beginning of the protocol, and re-computes the first round messages of the semi-malicious MPC using the honest party inputs. But what if now, when the verification phase is executed, the adversary no longer aborts? In this case, the distribution, once again, would be skewed.

To tackle this issue, we develop a procedure to sample the first round message in a manner such that the adversary aborts with roughly the same probability as before. We build on techniques from [GK96a] to devise our sampling technique, with the key difference that unlike [GK96a] which perform similar sampling with respect to a fixed prefix of the protocol, in our setting, we are sampling the very first message itself without any prefix. We refer the reader to the technical section for more details on this subject.

Now, suppose that the trusted party sends the "detect" signal to the simulator. In this case, the simulator must make sure that the verification phase is executed, and that an aborting transcript is generated. Towards this end, the simulator tries repeatedly until it is able to find such a transcript. Indeed, this is guaranteed given the aforementioned property of the re-sampled first round message. However, if the trusted party sends "non-detect" signal, the simulator may still end up with an aborting transcript. One may notice that this is not necessarily a bad thing since the simulator is able to detect cheating behavior with probability higher than what is required by the trusted party. To this end, we consider a strengthened formulation of MPC against covert adversaries where we allow the simulator to *increase* the detection probability by sending a parameter $\varepsilon_{sim} > \varepsilon$ to the trusted party. Upon receiving this parameter, the trusted party declares cheating to the honest parties with probability ε_{sim} as opposed to ε.

We now highlight another technical issue that arises during simulation. In an honest protocol execution, only a subset (and not necessarily *all*) of the honest parties may vote to initiate the verification phase. This means that when the simulator attempts to launch the verification phase for the first time in order to extract adversary's inputs, it must use random coins for the honest parties repeatedly until it leads to at least one honest party voting yes. In this case, the honest parties who do not vote for the verification phase must send their second round messages of semi-malicious MPC (as well as Zaps) *before* the extraction of adversary's input and randomness has been performed. A priori, it is unclear how this can be done without trivially failing the simulation.

We resolve this issue by using a new notion of *free simulation* for semi-malicious MPC. Roughly, we say that semi-malicious MPC protocol has free-simulation property if it possible to simulate the messages of all but one honest party using honest strategy (with respect to some random inputs) without the knowledge of adversary's inputs or output. Fortunately, as we discuss in the technical sections, the recent protocol of [BL18] based on two-round semi-malicious OT satisfies the free simulation property.

The above discussion is oversimplified and ignores several other issues that arise during simulation; we refer the reader to the technical sections for more details.

We address two natural questions that arise from our protocol

- *Can we build a protocol that has a worst case round complexity of 4?* We believe that similar ideas from our work may in fact eventually result in a protocol with a worst case complexity of four rounds. But given the complex nature of the known four round protocols [BGJ+18, HHPV18, CCG+19], any attempt to transform them in to the setting of covert adversaries is unlikely to yield a clean protocol and will only detract from the main underlying ideas of our presented protocol.

- *A malicious party can always force a worst case complexity of five rounds.* Our expected number of

rounds is in fact for honest parties. On the one hand an adversarial party can always force the protocol into *verification mode*, and such a strategy would force the adversarial party to provide a proof of honest behavior. On the other hand, an adversary can cheat undetected if the protocol does not go into *verification mode* thereby incentivizing a cheating adversary not to force *verification mode*.

1.2 Related Work

The round complexity of MPC has been extensively studied over the years in a variety of models. Here, we provide a short survey of this topic in the plain model, focusing on the dishonest majority setting. We refer the reader to [BGJ+18] for a more comprehensive survey.

Beaver et al. [BMR90] initiated the study of constant round MPC in the honest majority setting. Several follow-up works subsequently constructed constant round MPC against dishonest majority [KOS03, Pas04, PW10, Wee10, Goy11]. Garg et al. [GMPP16] established a lower bound of four rounds for MPC. They constructed five and six round MPC protocols using indistinguishability obfuscation and LWE, respectively, together with three-round robust non-malleable commitments.

The first four round MPC protocols were constructed independently by Ananth et al. [ACJ17] and Brakerski et al. [BHP17] based on different sub-exponential-time hardness assumptions. [ACJ17] also constructed a five round MPC protocol based on polynomial-time hardness assumptions. Ciampi et al. constructed four-round protocols for multiparty coin-tossing [COSV17a] and two-party computation [COSV17c] from polynomial-time assumptions. Benhamouda and Lin [BL18] gave a general transformation from any k-round OT with alternating messages to k-round MPC, for $k > 5$. More recently, independent works of Badrinarayanan et al. [BGJ+18] and Halevi et al. [HHPV18] constructed four round MPC protocols for general functionalities based on different polynomial-time assumptions. Specifically, [BGJ+18] rely on DDH (or QR or N-th Residuosity), and [HHPV18] rely on Zaps, affine-homomorphic encryption schemes and injective one-way functions (which can all be instantiated from QR).

Asharov et. al. [AJL+12] constructed three round semi-honest MPC protocols in the CRS model. Subsequently, two-round semi-honest MPC protocols in the CRS model were constructed by Garg et al. [GGHR14] using indistinguishability obfuscation, and by Mukherjee and Wichs [MW16] using LWE assumption. Recently, two-round semi-honest protocols in the plain model were constructed by Garg and Srinivasan [GS17, GS18] and Benhamouda and Lin [BL18] from two-round OT.

2 Definitions

We provide below the relevant definitions and our new definition of *Free-Simulatability*.

2.1 Secure Multi-Party Computation

A secure multi-party computation protocol is a protocol executed by n parties $P_1, \cdots, P_n$ for a n-party functionality $\mathcal{F}$. We allow for parties to exchange messages simultaneously. In every round, every party is allowed to broadcast messages to all parties. A protocol is said to have k *rounds* if the number of rounds in the protocol is k. We require that at the end of the protocol, all the parties receive the output $\mathcal{F}(x_1, \ldots, x_n)$, where x_i is the i^{th} party's input.

2.1.1 Adversarial Behavior

One of the primary goals in secure computation is to protect the honest parties against dishonest behavior from the corrupted parties. This is usually modeled using a central adversarial entity, that controls the set of corrupted parties and instructs them how to operate. That is, the adversary obtains the views of the

corrupted parties, consisting of their inputs, random tapes and incoming messages, and provides them with the messages that they are to send in the execution of the protocol.

Semi-Malicious Adversary A semi-malicious adversary is a stronger notion than semi-honest adversaries, but weaker than a malicious adversary. Formulation of this notion as presented in [AJL+12] is follows:

A semi-malicious adversary is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special witness tape. In each round of the protocol, whenever the adversary produces a new protocol message m on behalf of some party P_k, it must also write to its special witness tape some pair $(x; r)$ of input x and randomness r that explains its behavior. More specifically, all of the protocol messages sent by the adversary on behalf of P_k up to that point, including the new message m, must exactly match the honest protocol specification for P_k when executed with input x and randomness r. Note that the witnesses given in different rounds need not be consistent. Also, we assume that the attacker is rushing and hence may choose the message m and the witness $(x; r)$ in each round adaptively, after seeing the protocol messages of the honest parties in that round (and all prior rounds). Lastly, the adversary may also choose to abort the execution on behalf of P_k in any step of the interaction.

This definition captures the semi-honest adversary who always follows the protocol with honestly chosen random coins (and can be easily modified to write those on its witness tape). On the other hand, a semi malicious adversary is more restrictive than a fully malicious adversary, since its behavior follows the protocol with some input and randomness which it must know. Note that the semi-malicious adversary may choose the different input or random tape in an adaptive fashion using any PPT strategy according to the partial view it has seen.

2.1.2 Security

Secure computation that guarantees security with abort can be formalized as follows:

Real World. The real world execution begins by an adversary $\mathcal{A}$ selecting any arbitrary subset of parties $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ to corrupt. The parties then engage in an execution of a real n-party protocol Π. Throughout the execution of Π, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of Π.

At the conclusion of all the update phases, each honest party P_{i} outputs all the outputs it obtained in the computations. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, input vector $\overrightarrow{x}$, and security parameter 1^λ, we denote the output of the MPC protocol Π by

$$\mathsf{REAL}_{\mathcal{A},\Pi}\left(1^\lambda, \overrightarrow{x}, z\right).$$

Ideal World. We start by describing the ideal world experiment where n parties $\mathsf{P}_1, \cdots, \mathsf{P}_{\mathrm{n}}$ interact with an ideal functionality for computing a function F. An adversary may corrupt any subset $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ of the parties. We denote the honest parties by H.

Inputs: Each party P_{i} obtains an initial input x_{i}. The adversary Sim is given auxiliary input z. Sim selects a subset of the parties $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ to corrupt, and is given the inputs x_{k} of each party $\mathsf{P}_{\mathrm{k}} \in \mathcal{P}^{\mathcal{A}}$.

Sending inputs to trusted party: Each honest party P_{i} sends its input x_{i} to the trusted party. For each corrupted party $\mathsf{P}_{\mathrm{i}} \in \mathcal{P}^{\mathcal{A}}$, the adversary may select any value x_{i}^* and send it to the ideal functionality.

Trusted party computes output: Let $x_1^*, ..., x_{\mathrm{n}}^*$ be the inputs that were sent to the trusted party. The trusted party sends $\mathcal{F}(x_1^*, ..., x_{\mathrm{n}}^*)$ to the adversary who replies with either $\mathsf{continue}$ or abort. If the adversary's

message is abort, then the trusted party sends $\perp$ to all honest parties. Otherwise, it sends the function evaluation $\mathcal{F}(x_1^*, ..., x_n^*)$ to all honest parties.

Outputs: Honest parties output all the messages they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of the adversary's view.

The overall output of the ideal-world experiment consists of the outputs of all parties. For any ideal-world adversary Sim with auxiliary input $z \in \{0,1\}^*$, input vector $\overrightarrow{x}$, and security parameter 1^λ, we denote the output of the corresponding ideal-world experiment by

$$\mathsf{IDEAL}_{\mathsf{Sim},\mathcal{F}}\left(1^\lambda, \overrightarrow{x}, z\right).$$

Security Definition. We say that a protocol Π is a secure protocol if any adversary, who corrupts a subset of parties and runs the protocol with honest parties, gains *no information* about the inputs of the honest parties beyond the protocol output.

Definition 1. *A protocol Π is a secure n-party protocol computing $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary Sim corrupting the same parties in the ideal world such that for every initial input vector $\overrightarrow{x}$, every auxiliary input z, it holds that*

$$\mathsf{IDEAL}_{\mathsf{Sim},F}\left(1^\lambda, \overrightarrow{x}, z\right) \approx_c \mathsf{REAL}_{\mathcal{A},\Pi}\left(1^\lambda, \overrightarrow{x}, z\right).$$

2.2 Free-Simulatability

We define below the property of free-simulatability for a semi-malicious protocol. At a high level, a 2-round protocol computing f is said to satisfy the notion of *free simulatability* if there exists a simulator that can simulate the view for an adversary, consisting of second round messages for all *strict* subsets of honest parties, without querying the ideal functionality computing f. In addition, when it receives the output from the ideal functionality, it should be able to complete simulation of the view. The definition augments the definition of semi-malicious protocols (see Section 2.1) introduced in [BGJ+13].

Definition 2. *Let f be an n-party functionality. A semi-malicious protocol computing f is said to satisfy **free simulatability** if for all semi-malicious PPT adversary $\mathcal{A}$ controlling a set of parties I in the real world, there exists a PPT simulator Sim controlling the same set of parties such that for every input vector $\overrightarrow{x}$, every auxiliary input z, for every strict subset $J \subset \mathcal{H}$ of honest parties, the following is satisfied*

– ***free simulatability***

$$\left\{\left(\mathsf{REAL}_1^{\mathcal{A}}(\overrightarrow{x}, z), \mathsf{REAL}_{2,J}^{\mathcal{A}}(\overrightarrow{x}, z)\right)\right\} \approx_c \left\{\left(\gamma_1, \gamma_2\right) \middle| \begin{array}{r} (\gamma_1, \mathsf{state}_1) = \mathsf{Sim}(z) \\ (\gamma_2, \mathsf{state}_2) = \mathsf{Sim}(J, \mathsf{state}_1, z) \end{array}\right\}$$

– ***full simulatability***

$$\left\{\left(\mathsf{REAL}^{\mathcal{A}}(\overrightarrow{x}, z)\right)\right\} \approx_c \left\{\left(\gamma_1, (\gamma_2, \gamma_3)\right) \middle| \begin{array}{r} (\gamma_1, \mathsf{state}_1) = \mathsf{Sim}(z) \\ (\gamma_2, \mathsf{state}_2) = \mathsf{Sim}(J, \mathsf{state}_1, z) \\ \gamma_3 = \mathsf{Sim}(J, y, \mathsf{state}_2, z) \end{array}\right\}$$

$\mathcal{H}$ is the set of honest parties $\mathcal{H} \coloneqq [n] \setminus I$, $\mathsf{REAL}_1^{\mathcal{A}}(\overrightarrow{x}, z)$ is the view of the adversary in the first round for the real execution of the protocol, $\mathsf{REAL}_{2,J}^{\mathcal{A}}$ is the view of the adversary $\mathcal{A}$ consisting of only messages from honest parties in the set J in the second round of the real execution of the protocol, and $\mathsf{REAL}^{\mathcal{A}}(\overrightarrow{x}, z)$ is the entire view of the adversary in the real execution of the protocol. Here y is the output received from the ideal functionality computing the function f on the input vector $\overrightarrow{x}^*$ where the adversarial inputs have been replaced by $\{x_i^*\}_{i \in I}$.

Remark 1. *Note that this notion extends to any L round protocol, where instead of round 2, as in the definition above, we require the properties for partial view of the L-th round of the protocol. The basic GMW [GMW87] construction when instantiated with an appropriate semi-malicious OT (oblivious transfer) protocol gives us such an L round protocol satisfying free simulatability.*

For the 2 round setting, consider the 2 round semi-malicious protocol construction by Benhamouda and Lin [BL18]. At a very high level, the protocol in [BL18] compiles any L-round semi-malicious protocol into a 2 round protocol by garbling each next message function of the underlying L-round protocol. These garbled circuits are sent in the second round of the protocol, where the evaluation of these garbled circuits are enabled via a semi-malicious OT protocol. Importantly, to simulate the second round of their protocol, the simulator proceeds by first simulating the underlying L-round protocol, and then using the simulated transcript to simulate the garbled circuits. If the underlying L-round protocol satisfies *free simulatability*, then a transcript absent some messages from the last round is simulated without querying an ideal functionality. For the parties whose garbled circuits need to be simulated, i.e. parties in the subset J as described above, this transcript is sufficient since in the underlying L-round protocol parties send their last message of the protocol independent of the last message from the other parties. Once the full transcript is generated on querying the ideal functionality, the remaining garbled circuits are simulated. We refer the reader to [BL18] for the details.

2.3 Zero Knowledge

Definition 3. *An interactive protocol (P, V) for a language L is zero knowledge if the following properties hold:*

- **Completeness.** *For every $x \in L$,*

$$\Pr\left[\mathsf{out}_{\mathsf{V}}\left[\mathsf{P}(x, w) \leftrightarrow \mathsf{V}(x)\right] = 1\right] = 1$$

- **Soundness.** *There exists a negligible function $\mathsf{negl}(\cdot)$ s.t. $\forall x \notin L$ and for all adversarial prover P^*.*

$$\Pr\left[\mathsf{out}_{\mathsf{V}}\left[\mathsf{P}^*(x) \leftrightarrow \mathsf{V}(x)\right] = 1\right] \leq \mathsf{negl}(\lambda)$$

- **Zero Knowledge.** *For every PPT adversary V^*, there exists a PPT simulator Sim such that the probability ensembles*

$$\left\{\mathsf{view}_{\mathsf{V}}\left[\mathsf{P}(x, w) \leftrightarrow \mathsf{V}(x)\right]\right\}_{x \in L, w \in R_L(x)} \text{ and } \left\{\mathsf{Sim}(x)\right\}_{x \in L, w \in R_L(x)}$$

  are computationally indistinguishable.

Remark 2. *An interactive protocol is an argument system if it satisfies the completeness and soundness properties. We say an interactive proof is* input delayed *if both the statement and witness are required only for the computation of the last prover message.*

2.4 Input Delayed Non-malleable Zero Knowledge

Let $\Pi_{\mathsf{nmzk}} = \langle P, V \rangle$ be a input delayed interactive argument system for an NP-language L with witness relation Rel_L. Consider a PPT Man in the Middle (MiM) adversary $\mathcal{A}$ that is simultaneously participating in one left session and one right session. Before the execution starts, both P, V and $\mathcal{A}$ receive as a common input the security parameter 1^λ, and $\mathcal{A}$ receives as auxiliary input $z \in \{0,1\}^*$.

In the left session $\mathcal{A}$ interacts with P using identity $\mathtt{id}$ of his choice. In the right session, $\mathcal{A}$ interacts with V, using identity $\widetilde{\mathtt{id}}$ of his choice. In the left session, before the last round of the protocol, P gets the statement x. Also, in the right session $\mathcal{A}$, during the last round of the protocol selects the statement $\tilde{x}$ to be proved and sends it to V. Let $\mathsf{View}^{\mathcal{A}}(1^\lambda, z)$ denote a random variable that describes the view of $\mathcal{A}$ in the above experiment.

Definition 4 (Input Delayed NMZK)**.** *A input delayed argument system $\Pi_{\mathsf{nmzk}} = \langle P, V \rangle$ for NP-language L with witness relation $\mathsf{Rel}_{\mathsf{L}}$ is Non-Malleable Zero Knowledge (NMZK) if for any MiM adversary $\mathcal{A}$ that participates in one left session and one right session, there exists a PPT machine $\mathsf{Sim}(1^\lambda, z)$ such that*

1. *The probability ensembles $\{\mathsf{Sim}^1(1^\lambda, z)\}_{1^\lambda \in \mathbb{N}, z \in \{0,1\}^*}$ and $\{\mathsf{View}^{\mathcal{A}}(1^\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$ are computationally indistinguishable over 1^λ, where $\mathsf{Sim}^1(1^\lambda, z)$ denotes the first output of $\mathsf{Sim}(1^\lambda, z)$.*
2. *Let $z \in \{0,1\}^*$ and let $(\mathsf{View}, \tilde{w})$ denote the output of $\mathsf{Sim}(1^\lambda, z)$. Let $\tilde{x}$ be the right-session statement appearing in View and let $\mathtt{id}$ and $\tilde{\mathtt{id}}$ be the identities of the left and right sessions appearing in View. If the right session is accepting and $\mathtt{id} \neq \tilde{\mathtt{id}}$, then $\mathsf{Rel}_{\mathsf{L}}(\tilde{x}, \tilde{w}) = 1$.*

2.5 ZAP

Zap [DN00] are two-message public coin witness indistinguishable proofs, are defined as follows.

Definition 5. *A pair of algorithms $\langle \mathsf{P}, \mathsf{V} \rangle$ where P is PPT and V is (deterministic) polytime, is a Zap for an NP relation Rel_L if it satisfies:*

1. ***Completeness:*** *there exists a polynomial r such that for every $(x, w) \in \mathsf{Rel}_L$,*

$$\Pr_{\mathsf{P},\mathsf{r} \leftarrow\$ \{0,1\}^{r(|X|)}} [\mathsf{V}(x, \pi, \mathsf{r}) = 1 : \pi \leftarrow \mathsf{P}(x, w, \mathsf{r})] = 1$$

2. ***Adaptive Soundness:*** *for every malicious prover P^* and every $\lambda \in \mathbb{N}$:*

$$\Pr_{\mathsf{r} \leftarrow\$ \{0,1\}^{r(|X|)}} \left[\begin{array}{c} \exists x \in \{0,1\}^\lambda \setminus L \\ \pi \in \{0,1\}^* \end{array} : \mathsf{V}(x, \pi, \mathsf{r}) = 1 \right] \leq 2^{-\lambda}$$

3. ***Witness Indistinguishability:*** *for any instance $\mathcal{I} = \{(x, w_1, w_2) : w_1, w_2 \in \mathsf{Rel}_L(x)\}$ and any first message sequence $\mathcal{R} = \{\mathsf{r}_{x,w_1,w_2} : (x, w_1, w_2) \in \mathcal{I}\}$:*

$$\{\pi \leftarrow \mathsf{P}(x, w_1, \mathsf{r}_{x,w_1,w_2})\}_{(x,w_1,w_2) \in \mathcal{I}} \approx_c \{\pi \leftarrow \mathsf{P}(x, w_2, \mathsf{r}_{x,w_1,w_2})\}_{(x,w_1,w_2) \in \mathcal{I}}$$

3 Model

We describe in brief the motivation behind the model described in [AL07]. The model is intermediary to that of semi-honest and malicious. At a high level, the model allows for the adversary to be caught sufficiently often if it attempts to "cheat" during the protocol. The model formalizes the notion of "cheating" in the manner described below.

We parameterize adversarial cheating with the parameter $\varepsilon \in (0, 1]$, called the **deterrence factor**. The parameter signifies that honest parties detect cheating with probability at least ε. The notion is also referred to as **security in the presence of covert adversaries with ε-deterrent**. The ideal functionality is modified to accept a special cheat message from the adversary. On receiving the message, the ideal functionality sends the inputs of the honest parties to the adversary, and allows the adversary to set the output of honest parties. It then tosses coins internally and announces, with probability ε, to the honest parties that cheating has been detected. It should be noted that the adversary can always choose to cheat in the ideal world, but would be detected with probability ε.

The following text is taken largely verbatim from [AL07] but for a few important changes. We discuss these changes subsequent to the definition. At a high level, we strengthen the definition by allowing the simulator to override the situation that the ideal functionality did not send a cheat detection message to the honest parties, by forcing a detection message to be sent. This captures the intuition our protocol results in the adversary getting caught with a higher probability than parameterized by ε.

We also note that for simplicity we only consider the setting where all parties receive the same output and the definition naturally extends to the setting of each party receiving a different output. We denote the set of adversarial parties by I.

Input: Each party obtains an input; the party P_i's input is denoted by x_i; we assume that all inputs are of the same length denoted ℓ. The adversary receives an auxiliary-input z.

Send inputs to trusted party: An honest party P_i sends its received input x_i to the trusted party. The adversarial parties, controlled by $\mathcal{A}$, may either send their received input, or send some other input of the same length ot the trusted party. This decision is made by $\mathcal{A}$ and may depend on the values x_j for $j \in I$ and the auxiliary input z. Denote the vector of inputs sent to the trusted party by $\overrightarrow{w}$.

Abort options: If an adversarial party sends $w_j = \mathsf{abort}_j$ to the trusted part as its input, then the trusted party sends abort_j to all of the honest parties and halts. If multiple parties send abort_j, then the trusted party picks only one of them (say, the one with the smallest j).

Attempted cheat option: If an adversarial party sends $w_j = \mathsf{cheat}_j$ to the trusted party as its input, then the trusted party sends to the adversary all of the honest parties' inputs $\{\mathsf{x}_j\}_{j \notin I}$ (as above, if multiple cheat_j messages are sent, the trusted party ignores all but one). In addition,

1. With probability ε, the trusted party sends $\mathsf{corrupted}_j$ to the adversary and all of the honest parties.
2. With probability $1 - \varepsilon$, the trusted party sends $\mathsf{undetected}$ to the adversary. Following this, the adversary can either:
   - send $\mathsf{corrupted}_i$ to the trusted party on behalf of a party in I. Then the trusted party sends $\mathsf{corrupted}_i$ to all of the honest parties and halts; or
   - send output y of its choice for the honest parties to the trusted party. Then the trusted party sends y to all honest parties.

The ideal execution ends at this point.
If no w_j equals abort_j or cheat_j, the ideal execution continues below.

Trusted party answers adversary: The trusted party computes $y = f(\overrightarrow{w})$ and sends y to $\mathcal{A}$.

Trusted party answers honest parties: After receiving the output, the adversary either sends abort_j, $\mathsf{corrupted}_j$ for some $j \in I$, or $\mathsf{continue}$ to the trusted party. If the trusted party receives $\mathsf{continue}$ then it sends y to all honest parties. Otherwise, if it receives abort_j (resp. $\mathsf{corrupted}_j$) for some $j \in I$, it sends abort_j (resp. $\mathsf{corrupted}_j$) to all honest parties.

Outputs: An honest party always outputs the message it obtained from the trusted party. The adversarial parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{\mathsf{x}_j\}_{j\in I}$, the auxiliary input z, and the messages obtained from the trusted party.

The output of the honest parties and the adversary in an execution of the above ideal model is denoted by $\mathsf{IDEALC}^{\varepsilon}_{f,\mathsf{Sim}(z),I}(\overrightarrow{x}, \lambda)$ where $\overrightarrow{x}$ is the vector of inputs, z is the auxiliary input to $\mathcal{A}$, I is the set of adversarial parties, and λ is the security parameter. $\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(\overrightarrow{x}, \lambda)$ denotes the analogous outputs in a real execution of the protocol π. When we talk about the $\mathsf{corrupted}_i$ message, it is easier to think of this as a "detect" message sent to the ideal functionality to alert the honest parties of a detected cheating.

Definition 6. *Let $f : (\{0,1\}^*)^{\mathrm{n}} \to \{0,1\}^*$ be a function computed on n inputs, π be an n-party protocol, and $\varepsilon : \mathbb{N} \to [0,1]$ be a function. Protocol π is said to* ***securely compute*** *f* ***in the presence of covert adversaries with*** *ε-deterrent if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary Sim for the ideal model such that for every $I \subseteq [\mathrm{n}]$:*

$$\left\{\mathsf{IDEALC}^{\varepsilon}_{f,\mathsf{Sim}(z),I}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x},z\in(\{0,1\}^*)^{\mathrm{n}+1};\ell\in\mathbb{N}} \approx_c \left\{\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x},z\in(\{0,1\}^*)^{\mathrm{n}+1};\ell\in\mathbb{N}}$$

where every element of $\overrightarrow{x}$ is of the same length.

Remarks about our definition. We discuss the small but important changes to the definition of the ideal functionality from that in [AL07]:

- We work with the *explicit-cheat formulation* in [AL07], wherein we don't guarantee security of honest party inputs if the adversary cheats. This is opposed to the setting where the honest party inputs are secure if the adversary is caught cheating.
- In addition, we allow for the ideal world adversary to change the non-detection of cheating to a detection. Specifically, when the ideal functionality sends $\mathsf{undetected}$ to the ideal world adversary, it can choose to override this by subsequently sending a $\mathsf{corrupted}$ message to the ideal functionality which is then sent to all honest parties. We note that the adversary can only change a non-detect to detect, but not the other way round. This arguably strengthens the model allowing a cheating adversary to be detected with a higher probability.
- While we define the above model, for our proofs we find it more convenient to work with an equivalent model to the one described. In this model, the ideal world adversary can choose to send an additional parameter $\varepsilon_{\mathsf{Sim}} > \varepsilon$ in addition to the cheat message it sends to the trusted party. The trusted party now uses this new $\varepsilon_{\mathsf{Sim}}$ to determine whether or not cheating is detected by the honest parties. We still allow the ideal world adversary to override an $\mathsf{undetected}$ message. Equivalence between the models is easy to see since we essentially need to bias coins that detects with probability ε to one that detects with probability $\varepsilon_{\mathsf{Sim}}$ using the override option.

4 Protocol

Overview. The goal of covert secure computation is to provide the adversary enough of a deterrent to deviate from the protocol. We want to do this while also minimizing the number of communication rounds in our protocol. The basic idea is to run the protocol in two "modes". The *normal mode*, where an adversary can possibly get away by cheating; and the *extend mode* where the parties prove honest behavior via a (non-malleable) zero-knowledge protocol. The normal mode requires only two rounds of communication, while the extended mode requires five rounds. At the end of the first round of the *nomal mode*, parties vote to determine whether they enter *extend mode*, where the vote of a party is determined by sampling coins with parameter q. If there is even a single party that votes to go into *extend mode*, then all players run the *extend mode* prior to executing the second round of the normal mode.

While it is possible to parallelize rounds of the non-malleable zero knowledge with the first and voting rounds of the protocol, for simplicity of exposition we ignore this point for now. We shall make a note of this at the end of our protocol.

Expected number of rounds. If the probability with which each party decides to vote for the *extend mode* is q, the expected number of rounds for an honest execution of the protocol with n-parties will be $2 + 5 \cdot (1 - (1 - \mathsf{q})^{\mathrm{n}})$. We set the voting parameter to be $\mathsf{q} = \min\{2 \cdot \varepsilon, 1\}$, where ε is the deterrent parameter. This gives us a total of

$$2 + 5 \cdot \left(1 - \frac{1}{e^{2 \cdot \varepsilon \cdot \mathrm{n}}}\right)$$

rounds in expectation.

We achieve a simpler expression if we use the union bound. By the union bound, the probability of going into extend mode is upper bounded by $2 \cdot \varepsilon \cdot \mathrm{n}$. This gives us $2 + 5 \cdot (2\mathrm{n}\varepsilon)$ rounds in expectation, which is close to 2 for small values of ε.

Note that an adversary can always choose to ignore q and always go into the *extend mode* thereby forcing the worst case scenario of 7 rounds in the presence of an active adversary. As alluded to, our protocol can be easily modified to be 5 rounds in the worst case (see Remark 3).

4.1 Components

We list below the components of our protocol.

- Com is a non interactive commitment which can be instantiated from injective one-way functions. We use two instances of the protocol and denote the messages by com and icom, where the former will be used as the commitment to the "trapdoor witness" for the Zap while the latter is the commitment to the inputs. Subsequently referred to as "trapdoor commitment" and "input commitment" respectively.
- $\mathsf{NMZK} = (\mathsf{NMZK}_1, \mathsf{NMZK}_2, \mathsf{NMZK}_3, \mathsf{NMZK}_4, \mathsf{NMZK.Ver})$ is a four round non-malleable zero knowledge protocol (NMZK) which can be instantiated from collision resistant hash functions [COSV17b]. The language for the NMZK protocol is described below. The corresponding simulator-extractor for the NMZK is denoted by $\mathsf{Sim}_{\mathsf{nmzk}}$.
- $\mathsf{Zap} = (\mathsf{Zap}_1, \mathsf{Zap}_2, \mathsf{Zap.Ver})$ is Zap, i.e., a two-message public-coin witness indistinguishable proof. The language for the Zap is described below.
- $\Pi = (\Pi_1, \Pi_2, \mathsf{OUT})$ is a two round semi-malicious protocol computing the function f. OUT denotes the algorithm to compute the final output. Let Sim_{Π} denote the simulator for the protocol.

We additionally require the protocol Π to satisfy the property of **Free Simulatability**, formally defined in Appendix 2.2. The property ensures that we can simulate the second round messages of the protocol for a strict subset of the honest parties without requiring the output. Such a protocol can be instantiated from a two round semi-malicious OT protocol [BL18].

Notation. For notational convenience we make the use of the following conventions (i) we will ignore as input to the protocols the security parameter, which we assume is implicit; (ii) for multi-round protocols, we assume that the protocol maintains state and the previous round messages for the corresponding instances are implicit inputs; (iii) $\mathbf{T}_X[k]$ will indicate the transcript of protocol X for the first k round of the protocol, with suitable superscripts as required. For the underlying semi-malicious protocol Π, this corresponds to collecting all messages broadcast by the parties.

NP Languages. In our construction, we use proofs for the following NP languages:

1. Zap: We use Zap for language L, which is characterized by the following relation R

$\mathsf{st} = (\mathbf{T}_\Pi[1], \mathsf{msg}_2, \mathsf{com}, \mathsf{icom})$
$\mathsf{w} = (\mathsf{x}, \mathsf{r}, \mathsf{r}_{\mathsf{com}}, \mathsf{r}_{\mathsf{icom}})$

$R(\mathsf{st}, \mathsf{w}) = 1$ if *either* of the following conditions is satisfied:

(a) **Honest:** *all* of the following conditions hold
- icom is a commitment of Com w.r.t. to input $(\mathsf{x}, \mathsf{r}$ and randomness $\mathsf{r}_{\mathsf{icom}})$.
- msg_1 is honestly computed 1st round message of Π w.r.t. to input x and r.
- msg_2 is honestly computed 2nd round message of Π w.r.t. to input x, transcript $\mathbf{T}_\Pi[1]$ and r.

where $\mathbf{T}_\Pi[1]$ includes msg_1.

(b) **Trapdoor:** com is a commitment of Com w.r.t. to input 1 and randomness $\mathsf{r}_{\mathsf{com}}$

2. NMZK: We use NMZK for language $\widetilde{L}$, which is characterized by the following relation $\widetilde{R}$

$\widetilde{\mathsf{st}} = (\mathsf{msg}_1, \mathsf{icom}, \mathsf{com})$
$\widetilde{\mathsf{w}} = (\mathsf{x}, \mathsf{r}, \mathsf{r}_{\mathsf{icom}}, \mathsf{r}_{\mathsf{com}})$

$\widetilde{R}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{w}}) = 1$ if *all* of the following conditions hold:
- icom is a commitment of Com w.r.t. to input $(\mathsf{x}, \mathsf{r}$ and randomness $\mathsf{r}_{\mathsf{icom}})$.
- com is a commitment of Com w.r.t. to input 0 and randomness $\mathsf{r}_{\mathsf{com}}$
- msg_1 is honestly computed 1st round message of Π w.r.t. to input x and r.

Our protocol will crucially use the fact **that if relation $\widetilde{R}$ holds, then the trapdoor witness for R cannot hold.**

4.2 Protocol Description

We now describe our protocol between n parties $\mathsf{P}_1, \cdots, \mathsf{P}_\mathrm{n}$. The input of party P_i is denoted by x_i. The protocol is parameterized by q which is set to be $\min\{2 \cdot \varepsilon, 1\}$.

Round 1: P_i computes and broadcasts the following:

1. First round of the following protocols
   - Semi-malicious MPC Π: $\mathsf{msg}_{1,i} \leftarrow \Pi_1(\mathsf{x}_i, \mathsf{r}_i)$ using randomness r_i.

- Zap: for all $j \neq i$: $\mathsf{zap}_1^{j \to i} \leftarrow \mathsf{Zap}$

2. Non-interactive commitments Com
   - $\mathsf{icom}_i \leftarrow \mathsf{Com}((\mathsf{x}_i, \mathsf{r}_i); \mathsf{r}_{\mathsf{icom},i})$ to $(\mathsf{x}_i, \mathsf{r}_i)$
   - $\mathsf{com}_i \leftarrow \mathsf{Com}(0; \mathsf{r}_{\mathsf{com},i})$ to 0.

Round 2 (E.1): At the end of round 1, P_i determines whether it votes to go into *extend mode*, or directly go to *normal mode*. It does so by tossing a coin that outputs 1 with probability q. If the output is 1, set $\mathsf{voteExtend} = 1$, broadcast extend_i and go to **Extend Mode**. Else, set $\mathsf{voteExtend} = 0$ and go to **Normal Mode** and broadcast messages of **Normal Mode** described below.

Extend Mode: If there is even a single party that votes to go into the extend, then round **E.1** counts as the first round of the extend mode. The parties prove honest behavior of the first round via a four round non-malleable zero-knowledge protocol. If during the **Extend Mode**, a party does not send the expected message, P_i aborts and sets its output as corrupted.

Round E.2 P_i does the following:

1. If parties sent **Normal Mode** messages, store them for later use.
2. Compute and broadcast the following first round messages:
   - NMZK: for all $j \neq i$, $\mathsf{nmzk}_1^{j \to i} \leftarrow \mathsf{NMZK}_1$

Round E.3 P_i computes and broadcasts the following second round messages:

1. NMZK: for all $j \neq i$, $\mathsf{nmzk}_2^{i \to j} \leftarrow \mathsf{NMZK}_2$

Round E.4 P_i computes and broadcasts the following third round messages:

1. NMZK: for all $j \neq i$, $\mathsf{nmzk}_3^{j \to i} \leftarrow \mathsf{NMZK}_3$

Round E.5 P_i computes and broadcasts the following fourth round messages:

1. NMZK: for all $j \neq i$, $\mathsf{nmzk}_4^{i \to j} \leftarrow \mathsf{NMZK}_4(\widetilde{\mathsf{st}}_i, \widetilde{\mathsf{w}}_i)$ to prove that $\widetilde{R}(\widetilde{\mathsf{st}}_i, \widetilde{\mathsf{w}}_i) = 1$, where

$$\text{Statement } \widetilde{\mathsf{st}}_i := \left(\mathsf{msg}_{1,i}, \mathsf{icom}_i, \mathsf{com}_i\right)$$

$$\text{Witness } \widetilde{\mathsf{w}}_i := (\mathsf{x}_i, \mathsf{r}_i, \mathsf{r}_{\mathsf{icom},i}, \mathsf{r}_{\mathsf{com},i})$$

Normal Mode: [a] P_i does the following:
We separate out the steps performed based on how P_i arrived at the **Normal Mode**.

- If P_i sent extend_i it performs all the operations. Since we're guaranteed that the party has already completed execution of **Extend Mode**.
- If P_i did not send extend_i, we split its arrival into two cases:
  - if it comes prior to completion of **Extend Mode**, it performs all the operations other than the NMZK check.
  - if it comes after completion of **Extend Mode**, then we do only the checks for the NMZK. This is because it has already sent out its corresponding messages of **Normal Mode** when it sent out the second round message without voting to go to **Extend Mode**.

1. If $\mathsf{voteExtend} = 1$, check the NMZK proofs. If any of the checks do not succeed, abort computation and output corrupted.
   - for all $k, j \neq i$, $\mathsf{NMZK.Ver}\left(\widetilde{\mathsf{st}}_j, \mathbf{T}_{\mathsf{nmzk}}^{j \to k}[4]\right) \stackrel{?}{=} 1$
2. Compute and broadcast second round messages of the following protocols:
   - Π: $\mathsf{msg}_{2,i} \leftarrow \Pi_2\left(\mathsf{x}_i, \mathbf{T}_\Pi[1], \mathsf{r}_i\right)$
   - Zap: for all $j \neq i$, $\mathsf{zap}_2^{i \to j} \leftarrow \mathsf{Zap}_2\left(\mathsf{st}_i, \mathsf{w}_i\right)$ to prove that $R(\mathsf{st}_i, \mathsf{w}_i) = 1$, where

$$\text{Statement } \mathsf{st}_i := \left(\mathbf{T}_\Pi[1], \mathsf{msg}_{2,i}, \mathsf{com}_i, \mathsf{icom}_i\right)$$

$$\text{"Honest" Witness } \mathsf{w}_i := (\mathsf{x}_i, \mathsf{r}_i, \mathsf{r}_{\mathsf{com},i}, \perp)$$

If $\mathsf{voteExtend} = 0$, check if $\exists \mathsf{P}_j$ that sends extend_j at the end of the first round. If so, set $\mathsf{voteExtend} = 1$ and go to **Extend Mode**. In this case we will have to run the **Normal Mode** again.
If no party has voted to go into extend mode, then we're done.

Output Phase. P_i does the following,

1. Check the Zap proofs.
   - for all $k, j \neq i$, $\mathsf{Zap.Ver}\left(\mathsf{st}_j, \mathbf{T}_{\mathsf{zap}}^{j\to k}[2]\right) \stackrel{?}{=} 1$
   - If any of the Zap proofs do not verify above, do the following and abort:
     (a) if **Extend Mode** was executed, output $\perp$
     (b) else output $\mathsf{corrupted}$.
2. Compute the output of the function $y \leftarrow \mathsf{OUT}(\mathsf{x}_i, \mathsf{r}_i, \mathbf{T}_\Pi[2])$

[a] Different parties may execute parts of **Normal Mode** at different points in the protocol.

Remark 3. *It is easy to observe that we can bring down the worst case round complexity to five rounds if the computations performed in E.2 and E.3 are moved to Round 1 and 2 respectively. In this setting, the expected number of rounds are* $2 + 3 \cdot (1 - (1 - \mathsf{q})^{\mathrm{n}})$*, or alternatively* $2 + 3 \cdot (2n\varepsilon)$

Theorem 1. *Assuming the existence of injective one-way functions, collision resistant hash functions, semi-malicious oblivious transfer and ZAPs, the above protocol securely computes f in the presence of covert adversaries with deterrent ε.*

To ensure that honest parties correctly identify when an adversarial party has cheated, the proof of our protocol is quite intricate, and we describe the proof below..

4.3 Proof of Security

Consider a malicious non-uniform PPT adversary $\mathcal{A}$ who corrupts $t < n$ parties. We denote by $\mathcal{H}$ the set of uncorrupted parties. We abuse notation slightly and also consider $\mathcal{A}$ to refer to the subset of parties corrupted by the adversary. In the ideal world, the functionality $\mathcal{F}$ has deterrent parameter ε.

4.3.1 Description of the simulator

We briefly describe the high level overview of the simulator.

- The simulator starts by simulating the first round of the protocol, and always enters the extend mode to determine if the adversary cheated in the first round.
- Once in the extend mode, the simulator attempts to extract the adversary's input from the NMZK.
  - if the simulator is able to extract, it uses the extracted input to complete the simulation by restarting the protocol from after the completion of the first round. This completes the simulation.
  - if after sufficient number of attempts (to establish a high confidence), the simulator is unable to extract the inputs, it determines that the adversary is cheating with a high probability and proceeds accordingly.
- Now that the simulator has determined that the adversary has cheated, the simulator first estimates the probability that the adversary aborts, and then sends a cheat message (with the appropriate parameter) to the ideal functionality.
- The simulator then receives the honest inputs and is notified if the cheating was detected.

– With the honest inputs, the simulator samples a first round message such that the adversary aborts the same number of times, as before, in the extend mode. This is done to ensures an appropriate distribution.

– Now, based on whether on whether the ideal functionality reported cheating, the simulator samples an aborting transcript or skips the extend mode altogether. This needs to be done carefully to ensure that the distribution matches that of the real protocol.

We now give a formal description of the simulator.

Simulator Sim

Initial Phase The simulator does the following:

1. **Round 1** Sim performs the following actions:
   - Commit to inputs 0 for all honest parties via the non-interactive commitment. $\forall i \in \mathcal{H}\ \mathsf{icom}_i \coloneqq \mathsf{Com}\left(0; \mathsf{r}_{\mathsf{icom},i}\right)$
   - Commit to 1 in the non-interactive commitment to the trapdoor. $\forall i \in \mathcal{H}\ \mathsf{com}_i \coloneqq \mathsf{Com}\left(1; \mathsf{r}_{\mathsf{com},i}\right)$
   - First round message of the Zap. Specifically $\forall i \in \mathcal{H}, \forall j \in [n] \setminus \{i\}\ \mathsf{zap}_1^{j \to i} \leftarrow \mathsf{Zap}(1^\lambda)$
   - First round of the underlying protocol using input 0. $\forall i \in \mathcal{H}$, $\left(\left\{\mathsf{msg}_{1,i}\right\}_{i \in \mathcal{H}}, \mathsf{state}_1\right) \coloneqq \mathsf{Sim}_\Pi(1^\lambda, \mathcal{H})$
   - Send the computed messages to $\mathcal{A}$, and receive corresponding first round messages from $\mathcal{A}$.

2. **Go to Extend Mode.** Sim always goes to the **Extend Mode** by doing the following.
   - For each honest party, Sim samples independently a coin that outputs 1 with probability q. If all coins return 0, repeat till there is at least one honest party with output of coin toss 1.
   - For each honest party P_i
     - If coin toss above returned 1, set message to be extend_i.

     Else, let $\mathcal{V}$ be the set of parties that *did not* vote to go into extend mode. Simulate the **Normal Mode** messages of these parties as follows:
     - For the underlying protocol, use the **Free-Simulatability** property of $\mathsf{Sim}_\Pi\left(\left\{\mathsf{msg}_{2,i}\right\}_{i \notin \mathcal{V}}\right) \leftarrow \mathsf{Sim}_\Pi\left(\mathcal{V}, \mathsf{state}_2\right)$. We know that from the free simulatability property of the underlying protocol Π, and the guarantee that at least one honest party has its coin toss set to 1, the simulator of the underlying protocol can simulate the second round messages of P_i of the underlying protocol.
     - Use the trapdoor witness $\mathsf{r}_{\mathsf{icom},i}$ for the Zap $\forall i \in \mathcal{V}$, set statement
     $$\mathsf{st}_i \coloneqq \left(\left\{\mathsf{msg}_{1,j}\right\}_{j \in [n]}, \mathsf{msg}_{2,i}, \mathsf{com}_i, \mathsf{icom}_i\right)\ \mathsf{w}_i \coloneqq \left(\perp, \perp, \perp, \mathsf{r}_{\mathsf{com},i}\right)$$
     $\forall j \in [n] \setminus \{i\}$, compute $\mathsf{zap}_2^{i \to j} \leftarrow \mathsf{Zap}\left(\mathsf{st}_i, \mathsf{w}_i, \mathsf{zap}_1^{i \to j}\right)$
   - Send above set message to $\mathcal{A}$, and receive corresponding messages from $\mathcal{A}$. Since we are definitely going to **Extend Mode**, we ignore messages sent by $\mathcal{A}$.

3. **Extraction from the** NMZK **in Extend Mode.** Select a super-log parameter $\mathsf{p}_2 = \log^2 n$. Set $\mathsf{counter} = 0$
   (a) If $\mathsf{counter} = \mathsf{p}_2$, go to **Extraction Fail Phase** described below. Else, continue
   (b) Run $\mathsf{Sim}_{\mathsf{nmzk}}$ for rounds **E.2** to **E.5**. This is done by relaying messages between $\mathsf{Sim}_{\mathsf{nmzk}}$ and $\mathcal{A}$. We note that there are no other protocol messages for this period.
   (c) If $\mathsf{Sim}_{\mathsf{nmzk}}$ returns $\perp$, increment counter to be $\mathsf{counter} \coloneqq \mathsf{counter} + 1$. This happens if the extractor returns $\perp$ (for example, if the adversary aborted).
   Else, go to Step 4 to send extracted inputs to $\mathcal{F}$.
   (d) Go to step (a) and run $\mathsf{Sim}_{\mathsf{nmzk}}$ with fresh randomness.

4. **Send extracted inputs to $\mathcal{F}$.** Sim only reaches this point if $\mathsf{Sim}_{\mathsf{nmzk}}$ succeeded in extracting inputs from $\mathcal{A}$. Send extracted inputs $\{\mathsf{x}_j\}_{j\in\mathcal{A}}$ to $\mathcal{F}$ to receive output y.
5. Rewind the adversary to after the first round.
6. Sim now simulates the rest of the protocol on behalf of the honest parties as follows:
   - Sample vote for **Extend Mode** for each honest party using independent random coins, and accordingly send messages to $\mathcal{A}$.
   - If any party, honest or controlled by $\mathcal{A}$, votes to go into **Extend Mode**, complete the NMZK by simulating the proof using $\mathsf{Sim}_{\mathsf{nmzk}}$.
   - For the second round of the **Normal Mode**, simulate the messages on behalf of the honest parties, that did not send the extend message, using the output y received from $\mathcal{F}$. Specifically, $\left\{\mathsf{msg}_{2,i}\right\}_{i\in\mathcal{H}} \leftarrow \mathsf{Sim}_{\Pi}\left(\mathcal{V}, y, \mathsf{state}_2\right)$
   - Use the trapdoor witness $\mathsf{r}_{\mathsf{com},i}$ for the Zap. $\forall i \in \mathcal{H}$, set statement
   $$\mathsf{st}_i := \left(\left\{\mathsf{msg}_{1,j}\right\}_{j\in[n]}, \mathsf{msg}_{2,i}, \mathsf{com}_i, \mathsf{icom}_i\right)\ \mathsf{w}_i := \left(\perp, \perp, \perp, \mathsf{r}_{\mathsf{com},i}\right)$$
   $\forall j \in [n] \setminus \{i\}$, compute $\mathsf{zap}_2^{i\to j} \leftarrow \mathsf{Zap}\left(\mathsf{st}_i, \mathsf{w}_i, \mathsf{zap}_1^{i\to j}\right)$
7. If $\mathcal{A}$ aborts during the computation of Zap *after* computing **Extend Mode**, send abort to the ideal functionality; else if $\mathcal{A}$ aborts at any other point, send $\mathsf{corrupted}$ to the ideal functionality.
8. If the Zap and NMZK from $\mathcal{A}$ verify, send $\mathsf{continue}$ to $\mathcal{F}$ to deliver the output to the honest parties.
9. **Output Transcript.** Output transcript.

Extraction Fail Phase. Sim enters this phase only if all p_2 runs of $\mathsf{Sim}_{\mathsf{nmzk}}$ returned $\perp$. Thus, Sim has determined $\mathcal{A}$ has cheated, and needs to send cheat to $\mathcal{F}$.

1. **Estimate abort.** We need to estimate the probability (over the randomness used in the first round) that $\mathsf{Sim}_{\mathsf{nmzk}}$ fails in all p_2 attempts. This is done exactly as in [GK96a]. We denote the true probability of this event to be δ, and will denote its estimate by $\widetilde{\delta}$. We do this since we need to sample a first round message that will fail all p_2 attempts when executed with $\mathsf{Sim}_{\mathsf{nmzk}}$.
   - Sim fixes a polynomial $t = \mathsf{poly}(\lambda)$, where t is picked such that $\mathsf{Pr}\left[\frac{1}{2} \le \frac{\widetilde{\delta}}{\delta} \le 2\right] > 1 - 2^{\lambda}$.
   - Repeat Steps 1, 2 and 3 of the **Initial Phase** repeatedly using fresh randomness in the first round of the simulation until $\mathcal{A}$ causes $\mathsf{Sim}_{\mathsf{nmzk}}$ to abort in all p_2 attempts t times.
   - Estimate δ as $\widetilde{\delta} = \frac{t}{(\#\text{ of repetitions})}$.

   Note that we're not estimating this for a fixed first message.
2. **Send $\varepsilon_{\mathsf{Sim}}$ and cheat to $\mathcal{F}$.** (Refer to the remark in the model in Section 3.) We set the value of $\varepsilon_{\mathsf{Sim}}$ to be
   $$\varepsilon_{\mathsf{Sim}} := \left(1 - (1-q)^{|\mathcal{H}|}\right) \cdot \frac{1}{2}$$
   where $\left(1 - (1-\mathsf{q})^{|\mathcal{H}|}\right)$ is the probability that at least one honest party votes to go to the extend mode. $\frac{1}{2}$ loosely estimates the probability that $\mathcal{A}$ will abort in the **Extend Mode** given the first round message. This will be suitably modified subsequently. Send $\varepsilon_{\mathsf{Sim}}$ and cheat to $\mathcal{F}$. Since
   $$\left(1 - (1-q)^{|\mathcal{H}|}\right) \ge (1 - (1-q)) = q,$$
   we have $\varepsilon_{\mathsf{Sim}} \ge \frac{q}{2} = \varepsilon$ as required.
3. Receive inputs of honest parties along with either $\mathsf{corrupted}$ or $\mathsf{undetected}$.

   If received $\mathsf{corrupted}$ **then**

   Store $(\mathsf{YES}, \mathsf{YES})$.

   Else //received $\mathsf{undetected}$ from $\mathcal{F}$

Toss a coin that outputs 1 with probability $\frac{\varepsilon_{\mathsf{Sim}}}{1-\varepsilon_{\mathsf{Sim}}}$.

If output of coin toss 1 **then**

Store $(\mathsf{YES}, \mathsf{NO})$.

Else

Store $(\mathsf{NO}, \perp)$.

End

End

If the trusted party sends corrupted, we store $(\mathsf{YES}, \mathsf{YES})$ to indicate that the honest parties go into **Extend Mode** and that cheating is detected. If not, we toss a coin to determine whether we go into **Extend Mode** anyway. If so, we store this as $(\mathsf{YES}, \mathsf{NO})$. It is possible that in this case we override undetected to send corrupted. Lastly, $(\mathsf{NO}, \perp)$ is stored if we will not enter **Extend Mode** on behalf of the honest parties (the adversary can still force extend mode). Subsequent to storing these values, the simulation will restart the protocol from scratch. The strategy that the simulator follows is determined by the tuple stored above.

4. **Choose appropriate first round message.** We now restart the first round of the protocol using the honest inputs obtained from $\mathcal{F}$. But we need to pick a first round message such that the honest execution of the NMZK aborts on all p_2 attempts. We will use the estimate $\widetilde{\delta}$ for this purpose. Repeat the following procedure at most $\min\left(\frac{t}{\widetilde{\delta}}, 2^{\lambda}\right)$ times
   - Using fresh randomness and honest inputs, compute the first round messages of the protocol.
   - As in the **Initial Phase**, repeat steps 2 and 3. If $\mathsf{Sim}_{\mathsf{nmzk}}$ aborts on all p_2 threads, select the first message picked in the first round and go to Step 5.

   If no first round message picked, output fail and abort simulation. Note that the first round message comes from a different distribution as compared to earlier, since we're now behaving honestly in the first round.
5. **If** first stored value in the tuple is YES **then**
   (a) Go to **Extend Mode** as in Step 2 of **Initial Phase** of the simulations.
   (b) Run **Extend Mode** honestly on behalf of the honest parties using the honest execution of NMZK.
   (c) **If** second stored values YES (i.e. it is $(\mathsf{YES}, \mathsf{YES})$)) **then**
   Repeat from after the first round until $\mathcal{A}$ aborts in **Extend Mode**.
   //this is to ensure cheating is detected
   Else //stored value is $(\mathsf{YES}, \mathsf{NO})$
   We sample an aborted thread with probability $2\mathsf{p}-1\pm\mathsf{negl}(\lambda)$ where p is the probability that $\mathcal{A}$ aborts in the extended mode, and additionally send corrupted to $\mathcal{F}$. This is done as follows:
   i. Repeat from after the first round to generate 3 threads.
   ii. **If** all 3 threads above abort **then**
   Output the first aborted thread.
   Else
   Estimate the value p to be $\widetilde{\mathsf{p}}$. This is done by fixing the first round and repeating from after the second round with new randomness.
   – With probability $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ output the first aborted thread. If no aborted thread in the above three sampled threads, repeat process till aborted thread obtained.[a]
   – With probability $1-\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ output the first non-aborted thread.
   End
   End

Else //stored values is $(\mathsf{NO}, \perp)$
   (a) Sample coins on behalf of each honest party for the vote to enter **Extend Mode**. Repeat sampling until **no** honest party votes to go into extended mode.

(b) Run the rest of the protocol honestly using the obtained honest party inputs.

(c) **If** $\mathcal{A}$ votes to enter **Extend Mode** and then aborts **then**

Send corrupted to $\mathcal{F}$ and output the view.

Else

Complete the second round of the **Normal Mode** honestly.

End

End

6. For computing the output, recall that when a cheat message is sent to $\mathcal{F}$, $\mathcal{A}$ can choose an output of its choice. Additionally, note that Sim proceeds to **Normal Mode** only if $\mathcal{F}$ hasn't already sent corrupted to the honest parties.
   - Sim on receiving $\mathcal{A}$'s **Normal Mode** messages, computes the output as honest parties would, and sends the resultant honest party output to $\mathcal{F}$.
   - $\mathcal{A}$ can choose to abort in the second round of the **Normal Mode**. If the **Extend Mode** was not executed, send corrupted to $\mathcal{F}$ and abort. Else, send abort to $\mathcal{F}$ as the output.

[a] Since we're in the case that the adversary aborted sufficiently often and thus p is close to 1, we only need a constant number of tries in expectation.

Probability Calculation. Since our simulator's behavior is determined by various events during its execution, we compute the relevant probabilities that determine its behavior.

We start by defining the following events for the *real execution of the protocol*:

$$\begin{aligned}
\mathsf{BAD} &= \text{if first message results in } \mathsf{p}_2 \text{ aborts of } \mathsf{Sim}_{\mathsf{nmzk}} \\
\mathsf{GOOD} &= \mathsf{BAD}^c \\
\mathsf{EXTEND} &= \text{protocol goes into } \textbf{Extend Mode} \\
\mathsf{ABORT} &= \text{Adversary aborts}
\end{aligned}$$

Note that GOOD and BAD are events rather than message sets.

Remark 4. *In the real world, there is no notion of* $\mathsf{Sim}_{\mathsf{nmzk}}$. *The above event* GOOD *(and correspondingly* BAD*) indicates that if one were to hypothetically run the* $\mathsf{Sim}_{\mathsf{nmzk}}$ *extraction procedure for an honestly generated first round message in the real world, it would result in* p_2 *aborts.*

Let us denote the following probabilities,

$$\begin{aligned}
\gamma_{\mathsf{GOOD}} &= \mathsf{Pr}[\mathsf{GOOD}] \\
\gamma_{\mathsf{BAD}} &= \mathsf{Pr}[\mathsf{BAD}] \\
\mathsf{q}_{\mathsf{GOOD}} &= \mathsf{Pr}[\mathsf{EXTEND} \mid \mathsf{GOOD}] \\
\mathsf{q}_{\mathsf{BAD}} &= \mathsf{Pr}[\mathsf{EXTEND} \mid \mathsf{BAD}] \\
\mathsf{p}_{\mathsf{GOOD}} &= \mathsf{Pr}[\mathsf{ABORT} \mid \mathsf{GOOD} \wedge \mathsf{EXTEND}] \\
\mathsf{p}_{\mathsf{BAD}} &= \mathsf{Pr}[\mathsf{ABORT} \mid \mathsf{BAD} \wedge \mathsf{EXTEND}] \\
\mathsf{p} &= \mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT}]
\end{aligned}$$

Note that $\mathsf{q}_{\mathsf{GOOD}}$ and $\mathsf{q}_{\mathsf{BAD}}$ could potentially differ from q since an adversary can choose to vote for **Extend Mode** independent of the protocol parameter q.

Our interest is the probability p, as this corresponds to the probability with which honest parties output corrupted in the real execution of the protocol. We represent p in terms of the probabilities defined above.

Specifically,

$$\begin{aligned}
\mathsf{p} &= \mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT}] \\
&= \mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT}|\mathsf{GOOD}]\mathsf{Pr}[\mathsf{GOOD}] + \mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT}|\mathsf{BAD}]\mathsf{Pr}[\mathsf{BAD}] \\
&= \mathsf{Pr}[\mathsf{ABORT}|\mathsf{EXTEND} \wedge \mathsf{GOOD}]\mathsf{Pr}[\mathsf{EXTEND}|\mathsf{GOOD}]\mathsf{Pr}[\mathsf{GOOD}] \\
&\quad + \mathsf{Pr}[\mathsf{ABORT}|\mathsf{EXTEND} \wedge \mathsf{BAD}]\mathsf{Pr}[\mathsf{EXTEND}|\mathsf{BAD}]\mathsf{Pr}[\mathsf{BAD}] \\
&= \mathsf{p}_{\mathsf{GOOD}} \cdot \mathsf{q}_{\mathsf{GOOD}} \cdot \gamma_{\mathsf{GOOD}} + \mathsf{p}_{\mathsf{BAD}} \cdot \mathsf{q}_{\mathsf{BAD}} \cdot \gamma_{\mathsf{BAD}}
\end{aligned}$$

Now, in the simulated world,

$$\begin{aligned}
\mathsf{BAD} &= \text{simulator goes to } \textbf{Extraction Fail Phase} \\
\mathsf{GOOD} &= \mathsf{BAD}^c
\end{aligned}$$

the analogous probabilities are denotes as $\widehat{\gamma}_{\mathsf{GOOD}}$,$\widehat{\gamma}_{\mathsf{BAD}}$, $\widehat{\mathsf{q}}_{\mathsf{GOOD}}$, $\widehat{\mathsf{q}}_{\mathsf{BAD}}$, $\widehat{\mathsf{p}}_{\mathsf{GOOD}}$, $\widehat{\mathsf{p}}_{\mathsf{BAD}}$, $\widehat{\mathsf{p}}$.

In order to argue that the simulator outputs a transcript of the appropriate distribution, we specifically compare the probability of the adversary aborting in the **Extend Mode**, i.e. p and $\widehat{\mathsf{p}}$. From the above discussion, it suffices to show that each of the individual probabilities are negligibly close in the real and simulated worlds. In fact, for the real probabilities γ_{GOOD},γ_{BAD}, $\mathsf{q}_{\mathsf{GOOD}}$, $\mathsf{q}_{\mathsf{BAD}}$, $\mathsf{p}_{\mathsf{GOOD}}$ the corresponding probabilities in the simulated world differ only by a negligible amount. Intuitively, this will follow directly from the indistinguishability of the underlying primitives when we simulate the corresponding components, and will follow directly from our indistinguishability of the hybrids. What separates $\widehat{\mathsf{p}}_{\mathsf{BAD}}$ and $\mathsf{p}_{\mathsf{BAD}}$ from them is that in the simulated world we do a lot of intricate sampling in an attempt to ensure that we arrive at the right distribution. Therefore, we need to argue that $\widehat{\mathsf{p}}_{\mathsf{BAD}}$ and $\mathsf{p}_{\mathsf{BAD}}$ are negligibly close, i.e. we argue that the probability of aborting in the extend mode given that the bad event happens are almost identical in the real and simulated world.

Lemma 1. $\widehat{\mathsf{p}}_{\mathsf{BAD}} = \mathsf{p}_{\mathsf{BAD}} \pm \mathsf{negl}(\lambda)$

Proof. Note that in the real world, if the event BAD happens, the protocol is executed by continuing honestly. On the other hand, in the simulated world, the simulator executes the **Extraction Fail Phase**, re-samples first message and attempts to output an aborting thread with the appropriate probability (See Step 5 in the **Extraction Fail Phase**).

To do so, we first show that if we obtain a message m that results in the event BAD, the following claim holds. We denote message m resulting in event BAD as '$m \vDash \mathsf{BAD}$'.

Claim 1. *For any message m that results in the event* BAD*,*

$$\mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT} \mid m \vDash \mathsf{BAD}] = \widehat{\mathsf{q}}_{\mathsf{BAD}} \cdot \mathsf{p}_m \pm \mathsf{negl}(\lambda)$$

where $\mathsf{p}_m = \mathsf{Pr}[\mathsf{ABORT} \mid m \vDash \mathsf{BAD} \wedge \mathsf{EXTEND}]$

Proof. Given the first message m that leads to BAD, say that the probability of going into **Extend Mode** is $\widehat{\mathsf{q}}_{\mathsf{BAD}}$. We can rewrite this probability as

$$\widehat{\mathsf{q}}_{\mathsf{BAD}} = \widehat{\mathsf{q}}_{\mathsf{honestYES}} + \widehat{\mathsf{q}}_{\mathsf{honestNO}}$$

where $\widehat{\mathsf{q}}_{\mathsf{honestYES}}$ is the probability that one of the honest parties voted to go into **Extend Mode**, and $\widehat{\mathsf{q}}_{\mathsf{honestNO}}$ is the probability that all honest parties voted against going to **Extend Mode**, but the adversary voted to go to it.

Note that we, as the simulator, send $\varepsilon_{\mathsf{Sim}}$ to $\mathcal{F}$ which is identical to $\widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \frac{1}{2}$.

– stored value is $(\mathsf{YES}, \mathsf{YES})$: in this case, $\mathcal{F}$ has sent corrupted to the honest parties. Thus, the probability of aborting is

$$\varepsilon_{\mathsf{Sim}} = \widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \frac{1}{2}$$

– stored value is $(\mathsf{YES}, \mathsf{NO})$: in this case, we override undetected sent by $\mathcal{F}$ to corrupted with probability $2 \cdot \mathsf{p}_m - 1 \pm \mathsf{negl}(\lambda)$. Thus, the probability of aborting is

$$(1 - \varepsilon_{\mathsf{Sim}}) \cdot \frac{\varepsilon_{\mathsf{Sim}}}{1 - \varepsilon_{\mathsf{Sim}}} \cdot (2 \cdot \mathsf{p}_m - 1 \pm \mathsf{negl}(\lambda)) = \widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \frac{1}{2} \cdot (2 \cdot \mathsf{p}_m - 1 \pm \mathsf{negl}(\lambda))$$

– stored value is $(\mathsf{NO}, \perp)$: in this case, send corrupted only if $\mathcal{A}$ aborts in the **Extend Mode**. Thus, the probability of aborting is

$$\widehat{\mathsf{q}}_{\mathsf{honestNO}} \cdot \mathsf{p}_m$$

Therefore, the total probability of aborting is

$$\begin{aligned}
&\widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \frac{1}{2} + \widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \frac{1}{2} \cdot (2 \cdot \mathsf{p}_m - 1 \pm \mathsf{negl}(\lambda)) + \widehat{\mathsf{q}}_{\mathsf{honestNO}} \cdot \mathsf{p}_m \\
&\quad = \widehat{\mathsf{q}}_{\mathsf{honestYES}} \cdot \mathsf{p}_m + \mathsf{q}_{\mathsf{honestNO}} \cdot \mathsf{p}_m \pm \mathsf{negl}(\lambda) \\
&\quad = (\widehat{\mathsf{q}}_{\mathsf{honestYES}} + \widehat{\mathsf{q}}_{\mathsf{honestNO}}) \cdot \mathsf{p}_m \pm \mathsf{negl}(\lambda) \\
&\quad = \widehat{\mathsf{q}}_{\mathsf{BAD}} \cdot \mathsf{p}_m \pm \mathsf{negl}(\lambda)
\end{aligned}$$

□

This completes our claim. Now, we sum over all messages that lead to BAD,

$$\begin{aligned}
\mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT} \mid \mathsf{BAD}] &= \sum_{m \vDash \mathsf{BAD}} \mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT} \mid m \vDash \mathsf{BAD}] \cdot \mathsf{Pr}[m \vDash \mathsf{BAD}] \\
&= \widehat{\mathsf{q}}_{\mathsf{BAD}} \sum_{m \vDash \mathsf{BAD}} \mathsf{p}_m \cdot \mathsf{Pr}[m \vDash \mathsf{BAD}] \pm \mathsf{negl}(\lambda) \\
&= \widehat{\mathsf{q}}_{\mathsf{BAD}} \cdot \mathsf{p}_{\mathsf{BAD}} \pm \mathsf{negl}(\lambda)
\end{aligned}$$

where $\sum_{m \vDash \mathsf{BAD}} \mathsf{p}_m \cdot \mathsf{Pr}[m \vDash \mathsf{BAD}] = \mathsf{p}_{\mathsf{BAD}}$. By definition, we also know that

$$\mathsf{Pr}[\mathsf{EXTEND} \wedge \mathsf{ABORT} \mid \mathsf{BAD}] = \widehat{\mathsf{q}}_{\mathsf{BAD}} \cdot \widehat{\mathsf{p}}_{\mathsf{BAD}}$$

This gives us $\mathsf{p}_{\mathsf{BAD}} = \widehat{\mathsf{p}}_{\mathsf{BAD}} \pm \mathsf{negl}(\lambda)$. □

As discussed, this suffices to argue that Sim outputs a transcript that aborts with probability negligibly close to that of a real execution. We will see the direction application of this lemma in our hybrid indistinguishability.

Correctness of Sim**.** For correctness, we need to argue that in Step 5(c) of the simulator, the described simulation strategy indeed results in a thread that aborts with probability close to $2\mathsf{p} - 1$.

Lemma 2. *For the stored value* $(\mathsf{YES}, \mathsf{NO})$*, simulator outputs an aborted thread with probability* $2\mathsf{p} - 1 \pm \mathsf{negl}(\lambda)$.

Proof. We are required to sample an aborted thread with probability $2\mathsf{p}-1\pm\mathsf{negl}(\lambda)$ without the knowledge of p. As a first step, we estimate p as $\widetilde{\mathsf{p}}$.

Let E_1 be the event that the simulator samples an aborted thread when the stored value is $(\mathsf{YES}, \mathsf{NO})$, and E_2 corresponds to the event that the first three sampled threads all aborted. Then,

$$\begin{aligned}
\mathsf{Pr}[\mathsf{E}_1] &= \mathsf{Pr}[\mathsf{E}_2] + \mathsf{Pr}[\mathsf{E}_2^c] \cdot \frac{2\widetilde{\mathsf{p}} - 1 - \widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3} = \mathsf{p}^3 + \left(1-\mathsf{p}^3\right) \cdot \frac{2\widetilde{\mathsf{p}} - 1 - \widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3} \\
&= \mathsf{p}^3 + \left(1-\mathsf{p}^3\right) \cdot \frac{(1-\widetilde{\mathsf{p}})(\widetilde{\mathsf{p}}^2 - 1 + \widetilde{\mathsf{p}})}{1-\widetilde{\mathsf{p}}^3} = \mathsf{p}^3 + \left(1-\mathsf{p}^3\right) \cdot \frac{(\widetilde{\mathsf{p}}^2 - 1 + \widetilde{\mathsf{p}})}{1+\widetilde{\mathsf{p}}+\widetilde{\mathsf{p}}^2} \\
&= \frac{\mathsf{p}^3 \cdot \left(1+\widetilde{\mathsf{p}}+\widetilde{\mathsf{p}}^2\right) + \left(1-\mathsf{p}^3\right) \cdot (\widetilde{\mathsf{p}}^2 - 1 + \widetilde{\mathsf{p}})}{1+\widetilde{\mathsf{p}}+\widetilde{\mathsf{p}}^2} \\
&= \frac{2\mathsf{p}^3 + \widetilde{\mathsf{p}}^2 + \widetilde{\mathsf{p}} - 1}{1+\widetilde{\mathsf{p}}+\widetilde{\mathsf{p}}^2} = \frac{2\widetilde{\mathsf{p}}^3 + \widetilde{\mathsf{p}}^2 + \widetilde{\mathsf{p}} - 1 \pm \mathsf{negl}(\lambda)}{1+\widetilde{\mathsf{p}}+\widetilde{\mathsf{p}}^2} \\
&= 2\widetilde{\mathsf{p}} - 1 \pm \mathsf{negl}(\lambda) = 2\mathsf{p} - 1 \pm \mathsf{negl}(\lambda)
\end{aligned}$$

where $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ is the probability that the first aborted thread is output given that not all 3 threads aborted. The $\widetilde{\mathsf{p}}$ can be off by negligible, giving the $\pm\mathsf{negl}(\lambda)$ error in the calculation. □

Lastly, it must also be the case that the probability $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ is non-negative. We prove this below.

Probability $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ is non-negative. To output an aborted thread with probability $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$, we need to ensure that the probability is non-negative. Since $(1-\widetilde{\mathsf{p}}^3) > 0$ when $\widetilde{\mathsf{p}} < 1$, it suffices to show that $2\widetilde{\mathsf{p}} - 1 - \widetilde{\mathsf{p}}^3$ is non-negative.

$$\begin{aligned}
2\widetilde{\mathsf{p}} - 1 - \widetilde{\mathsf{p}}^3 &= \widetilde{\mathsf{p}}^2 - \widetilde{\mathsf{p}}^3 - 1 + 2\widetilde{\mathsf{p}} - \widetilde{\mathsf{p}}^2 \\
&= \widetilde{\mathsf{p}}^2(1-\widetilde{\mathsf{p}}) - (1-\widetilde{\mathsf{p}})^2 = (1-\widetilde{\mathsf{p}})(\widetilde{\mathsf{p}}^2 - 1 + \widetilde{\mathsf{p}})
\end{aligned}$$

$\widetilde{\mathsf{p}}^2 - 1 + \widetilde{\mathsf{p}}$ is non-negative for $\frac{3}{4} < \widetilde{\mathsf{p}} < 1$. We show below that this is indeed the case.

Claim 2. *Let* E *be the event that simulator goes to* ***Extraction Fail Phase*** *and then samples a first round message* m *with* $\mathsf{p}_m < \frac{3}{4}$. *Then* $\mathsf{Pr}[E] < \mathsf{negl}(\lambda)$

Proof. Let us split the proof into two cases:

- The fraction of first round messages m such that they abort with probability $> \frac{3}{4}$ in the **Extend Mode** is negligible. Then probability of aborting all p_2 attempts is $< (1-\frac{1}{4})^{\mathsf{p}_2} \approx e^{\mathsf{c}\cdot\mathsf{p}_2} = e^{\mathsf{c}\cdot\omega(\log n)}$ which is negligible.

- The fraction of first round messages m such that they abort with probability $> \frac{3}{4}$ in the **Extend Mode** is noticeable. Let this be some distribution D_1 that we're sampling from. When we do the re-sampling we sample from an indistinguishable distribution D_2. If the fraction of first round messages m such that they abort with probability $> \frac{3}{4}$ in the **Extend Mode** is noticeable when sampled from D_1, then this must be true for D_2 as well. If not, we can distinguish between the two distributions.

  For the sake of contradiction, assume that $\mathsf{Pr}[E] = \gamma$ is noticeable.

  Then there exists a fixed polynomial $q = q(\lambda)$ such that in q attempts with probability at least $1 - \frac{\gamma}{2}$, we find a message m by re-sampling from D_2 such that it aborts in all p_2 iterations of **Extend Mode**.

Let F be the event that we sampled a message that m that aborts in all p_2 iterations of **Extend Mode**. Now we get that from the union bound,

$$\begin{aligned}\mathsf{Pr}[E] &= \mathsf{Pr}[E|F]\mathsf{Pr}[F] + \mathsf{Pr}[E|F^c]\mathsf{Pr}[F^c] = \mathsf{Pr}[E|F]\left(1-\frac{\gamma}{2}\right) + \mathsf{Pr}[E|F^c]\frac{\gamma}{2} \\ &< \mathsf{negl}(\lambda)\left(1-\frac{\gamma}{2}\right) + 1\cdot\frac{\gamma}{2} < \frac{\gamma}{2} + \mathsf{negl}(\lambda)\end{aligned}$$

which gives us a contradiction.

□

Running time of Sim**.** We sketch below the proof of Sim's running time.

Claim 3. Sim *runs in expected time that is polynomial in* λ*.*

Proof. It is easy to see that other than the estimation and sampling of the first round message, Sim runs in $\mathsf{poly}(\lambda)$ time. We repeat the analysis in [GK96a] to show that this step too runs in expected polynomial time.

The probability that Sim goes to **Extraction Fail Phase** is δ. The expected number of iterations during estimation is $\frac{t}{\widetilde{\delta}}$, and the cut off point before Sim outputs fail is $\frac{t}{\widetilde{\delta}} < \frac{t}{2\delta}$. Note that this step may still take time 2^λ when the estimation $\widetilde{\delta}$ is incorrect, but this only happens with probability $2^{-\lambda}$ Hence, other than with negligible probability the expected running time of the estimation phase and sampling of first round message is given by

$$\mathsf{poly}(\lambda)\cdot\delta\cdot\left(\frac{t}{\delta} + \left(1-\frac{1}{2^\lambda}\right)\frac{2t}{\delta} + \left(\frac{1}{2^\lambda}\cdot 2^\lambda\right)\right) \leq \mathsf{poly}(\lambda)$$

□

4.3.2 Hybrids

We describe here the hybrids for our proof of indistinguishability.

Hybrid Hyb_0**:** This is the real execution of the protocol.

Hybrid Hyb_1**:** Hyb_1 is the same as Hyb_0 but the output thread is picked in the following manner:

- **Initial Phase.** Always enter **Extend Mode** on behalf of honest players by sampling coins repeatedly until at least one honest party votes to go into **Extend Mode**.
  1. Repeat **Extend Mode** p_2 times.
  2. If there is at least one execution thread where $\mathcal{A}$ did not abort, keep round 1 and restart from after round 1 behaving honestly. Else, go to **Extraction Fail Phase**.
- **Extraction Fail Phase.** If all p_2 threads in the **Initial Phase** aborted, we need to estimate the probability that all p_2 threads were aborted. This is done exactly as in Step 1 of Sim. Repetitions are done by sampling fresh randomness for round 1 messages. Now repeat the following $\min\left(\frac{t}{\widetilde{\delta}}, 2^\lambda\right)$
  1. Always enter **Extend Mode** on behalf of honest players by sampling coins repeatedly until at least one honest party votes to go into **Extend Mode**.

2. Repeat **Extend Mode** p_2 times honestly.
3. If $\mathcal{A}$ aborts in all p_2 threads, keep the round 1 and exit the loop. Else, continue.

If we fail to to find such a first round message, we output fail. Else, restart from after picked first round message and follow honest strategy.

For the next few hybrids, we make changes only to the **Initial Phase**.

Hybrid Hyb_2**:** Hyb_2 is identical to Hyb_1 except for the following changes:

- In the **Initial Phase** use the simulator-extractor $\mathsf{Sim}_{\mathsf{nmzk}}$ in the **Extend Mode**.
- This includes all p_2 threads and potentially one more if the **Extend Mode** is voted for by the parties subsequent to the running of the p_2 threads of the **Initial Phase**.

The **Extraction Fail Phase** remains unchanged.

Hybrid Hyb_3**:** Hyb_3 is identical to Hyb_2 except that in the **Initial Phase**, we commit to 1 in com_i on behalf of all honest parties P_i.
The **Extraction Fail Phase** remains unchanged.

Hybrid Hyb_4**:** Hyb_4 is identical to Hyb_3 except that we switch the witness used in the Zap sent by the honest parties in the **Normal Mode** of the **Initial Phase**. Recall that the **Normal Mode** of the **Initial Phase** is only reached if at least one of the p_2 threads in the **Initial Phase** did not abort.
The **Extraction Fail Phase** remains unchanged.

Hybrid Hyb_5**:** Hyb_5 is identical to Hyb_4 except that we commit to input 0 in icom_i on behalf of every honest party P_i.
The **Extraction Fail Phase** remains unchanged.
Note that we order this hybrid here since we can only make changes to the input commitment once we're using the "trapdoor witness" in the Zap.

Hybrid Hyb_6**:** Hyb_6 is identical to Hyb_5 except that we now simulate the underlying protocol messages using Sim_Π. First, the first round Π component is replaced by the one returned by Sim_Π. Then, if the **Initial Phase** leads to at least a single non-aborting thread, we send the extracted adversarial inputs (using the NMZK) to the ideal functionality, and receive the output. The output is then used to simulate the second round messages of the underlying protocol. Recall that a non-aborting **Initial Phase** implies that the first round messages of the underlying protocol were computed honestly.
The **Extraction Fail Phase** remains unchanged.

We now make changes to the **Extraction Fail Phase** while the **Initial Phase** remains unchanged.

Hybrid Hyb_7**:** Hyb_7 is identical to Hyb_6 except that we now send a cheat to the ideal functionality when we reach the **Extraction Fail Phase**. On receiving the honest parties' inputs, use it in the entire execution of the **Extraction Fail Phase** (excluding the estimation).

At this point, we've stopped using honest inputs completely, other than those received from $\mathcal{F}$. But we need to modify how the honest party outputs are computed. To this end, we make these changes via the following two hybrids.

Hybrid Hyb_8**:** Hyb_8 is identical to Hyb_7 except that we determine the parameter $\varepsilon_{\mathsf{Sim}}$ sent to the ideal functionality along with subsequent overrides to corrupted. This is done identically to the simulation as follows:

1. Set the value of $\varepsilon_{\mathsf{Sim}}$ to be
$$\varepsilon_{\mathsf{Sim}} \coloneqq \left(1-(1-q)^{|\mathcal{H}|}\right)\cdot\frac{1}{2}$$
where $\left(1-(1-\mathsf{q})^{|\mathcal{H}|}\right)$ is the probability that at least one honest party votes to go to the extend mode. $\frac{1}{2}$ loosely estimates the probability that $\mathcal{A}$ will abort in the **Extend Mode** given the first round message.
2. Receive inputs of honest parties along with either corrupted or undetected.
If received corrupted **then**
Store $(\mathsf{YES},\mathsf{YES})$.
Else //received undetected from $\mathcal{F}$
Toss a coin that outputs 1 with probability $\frac{\varepsilon_{\mathsf{Sim}}}{1-\varepsilon_{\mathsf{Sim}}}$.
If output of coin toss 1 **then**
Store $(\mathsf{YES},\mathsf{NO})$.
Else
Store $(\mathsf{NO},\perp)$.
End
End
3. **If** first stored value in the tuple is YES **then**
   (a) Go to **Extend Mode** as in Step 2 of **Initial Phase** of the simulations.
   (b) Run **Extend Mode** honestly on behalf of the honest parties using the honest execution of NMZK.
   (c) **If** second stored values YES (i.e. it is $(\mathsf{YES},\mathsf{YES})$) **then**
   Repeat from after the first round until $\mathcal{A}$ aborts in **Extend Mode**.
   //this is to ensure cheating is detected
   Else //stored value is $(\mathsf{YES},\mathsf{NO})$
   We sample an aborted thread with probability $2\mathsf{p}-1\pm\mathsf{negl}(\lambda)$ where p is the probability that $\mathcal{A}$ aborts in the extended mode, and additionally send corrupted to $\mathcal{F}$. This is done as follows:
   i. Repeat from after the first round to generate 3 threads.
   ii. **If** all 3 threads above abort **then**
   Output the first aborted thread.
   Else
   Estimate the value p to be $\widetilde{\mathsf{p}}$. This is done by fixing the first round and repeating from after the second round with new randomness.
   – With probability $\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ output the first aborted thread. If no aborted thread in the above three sampled threads, repeat process till aborted thread obtained.[a]
   – With probability $1-\frac{2\widetilde{\mathsf{p}}-1-\widetilde{\mathsf{p}}^3}{1-\widetilde{\mathsf{p}}^3}$ output the first non-aborted thread.
   End
   End
Else //stored values is $(\mathsf{NO},\perp)$
   (a) Sample coins on behalf of each honest party for the vote to enter **Extend Mode**. Repeat sampling until **no** honest party votes to go into extended mode.

(b) Run the rest of the protocol honestly using the obtained honest party inputs.
(c) **If** $\mathcal{A}$ votes to enter **Extend Mode** and then aborts **then**
Send corrupted to $\mathcal{F}$ and output the view.
Else
Complete the second round of the **Normal Mode** honestly.
End
End

[a] Since we're in the case that the adversary aborted sufficiently often and thus p is close to 1, we only need a constant number of tries in expectation.

Hybrid Hyb_9**:** Hyb_9 is identical to Hyb_8 except for how the outputs of the honest parties are computed. Previously the output of the honest parties was computed following the honest strategy. Since the ideal functionality expects an output to send to the honest parties if $\mathcal{A}$ does not abort, we pass the computed output to the ideal functionality.

Note that Hyb_9 is identical to our simulator Sim.

4.3.3 Indistinguishability of Hybrids

$\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$**:** We separate the analysis into two cases:

- If least one of the p_2 threads in **Initial Phase** do not abort in **Extend Mode**, then the views output in the two hybrids are the same. So in this case, they are identically distributed.
- If all p_2 threads in **Initial Phase** abort, then the **Extraction Fail Phase** is executed. Since the round 1 message for the view is sampled from the same distribution, the views are identical except for the case that fail is output. Below we prove that this happens only with negligible probability.

  Claim 4. *Probability* fail *is output is negligible.*

  Proof. Here, the estimation of p_2 is done by running the **Extend Mode** of **Initial Phase**, where the **Extend Mode** is run using the honest non-malleable zero knowledge (NMZK). When we sample a first round message in **Extraction Fail Phase**, we run the honest NMZK too. Thus, the estimation and the selection of the first round message come from the same distribution.

  By our choice of parameters for estimation, we know that the estimate is correct up to a factor of 2 other than with negligible probability. Thus, with only negligible probability will we fail to sample the appropriate first round message in the **Extraction Fail Phase**. Thus, the probability that fail is output is negligible. □

$\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$**:** As before, we separate our analysis into two case. This follows from the fact that the changes are only made in the **Initial Phase**.

- If at least one of the p_2 threads in **Initial Phase** do not abort in **Extend Mode**, we output a view that contains either:
  - A transcript containing a single execution of the non-malleable zero knowledge (NMZK) output by $\mathsf{Sim}_{\mathsf{nmzk}}$, if any party votes to go the **Extend Mode**.

– A transcript without an execution of the NMZK. This occurs when no party votes to go to **Extend Mode**.

In the first case, the view indistinguishability follows from the security of the NMZK when simulating with $\mathsf{Sim}_{\mathsf{nmzk}}$. In the second case, the view output is identical

– If all p_2 threads in **Initial Phase** abort, then the subsequent thread is executed.

Claim 5. *Probability* fail *is output is negligible.*

Proof. Here, the estimation of p_2 is done by running the **Extend Mode** of **Initial Phase**, where the **Extend Mode** is run using the simulator-extractor $\mathsf{Sim}_{\mathsf{nmzk}}$. On the other hand, when we sample a first round message in the **Extraction Fail Phase**, we run the honest non-malleable zero knowledge.

This is exactly the analysis captured by [GK96a]. We denote by δ, the probability that adv aborts in all p_2 attempts when using the simulated NMZK in the **Extend Mode**. Let ρ denote the probability of $\mathcal{A}$ aborting in all p_2 attempts when using the honest execution of NMZK. From the security of the NMZK, we require that $|\rho - \delta|$ is negligible. Else we break simulation-extraction security.

$$\begin{aligned}
\mathsf{Pr}[\mathsf{Sim\ outputs\ fail}] &= q \sum_i \left(\mathsf{Pr}\left[\frac{1}{\frac{\delta}{\delta}} = i \right] \right) (1-\rho)^{t \cdot i} \\
&\leq \delta \mathsf{Pr}\left[\frac{\delta}{\frac{\delta}{\delta}} \geq \frac{1}{2} \right] (1-\rho)^{\frac{t}{\delta}} + \delta \mathsf{Pr}\left[\frac{\delta}{\frac{\delta}{\delta}} < \frac{1}{2} \right] \\
&\leq \delta (1-\rho)^{\frac{t}{2\delta}} + \mathsf{negl}(\lambda) \qquad (1)
\end{aligned}$$

To show that the above equation is negligible in λ, we split the analysis into two cases:

– **Case 1:** $\rho \geq \frac{\delta}{2}$. Substituting, we get

$$(1-\rho)^{\frac{t}{2\delta}} \leq \left(1 - \frac{\delta}{2}\right)^{\frac{t}{2\delta}} < e^{-\frac{t}{4}}$$

which is negligible is λ since t is polynomial in λ.

– **Case 2:** $\rho < \frac{\delta}{2}$. To the contrary, let us assume Equation (1) is non-negligible. Then, there is a polynomial $\mathsf{poly}(\lambda)$ and infinitely many values λ such that

$$\delta \geq \delta (1-\rho)^{\frac{t}{2\delta}} + \mathsf{negl}(\lambda) > \frac{1}{\mathsf{poly}(\lambda)}.$$

Thus $\delta > \frac{1}{\mathsf{poly}(\lambda)}$ for some polynomial $\mathsf{poly}(\lambda)$. This gives us

$$|\delta - \rho| > \left|\frac{\delta}{2}\right| > \frac{1}{2\mathsf{poly}(\lambda)}.$$

This breaks the security of the non-malleable zero-knowledge.

Thus Sim outputs fail with only negligible probability. □

$\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$**:** The only change made is to the "trapdoor commitment" in the first round of the **Initial Phase**. The analysis follows similarly to that of $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

– If at least one of the p_2 threads in **Initial Phase** do not abort in **Extend Mode**, we output a view that contains the "trapdoor commitment" to 1. The rest is identical to the previous hybrid. The view indistinguishability follows from the hiding property of the commitment scheme.

– If all p_2 threads in **Initial Phase** abort, then the subsequent thread is executed. Since there is no change in the **Subsequent Mode**, the views are identical except for the case that fail is output. This follows identically from the analysis of $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

$\mathsf{Hyb}_3 \approx \mathsf{Hyb}_4$**:** Recall that this change is only made in the **Initial Phase** if the Extraction Fail Phase is not executed. The change is only made in the last round of the protocol by switching the witness in Zap. The indistinguishability of the view follows from the witness indistinguishability of Zap.

Since the changes made are after the completion of the **Extend Mode**, it has no bearing on the estimation probability used in the **Extraction Fail Phase**. Thus the analysis of the views output in the **Extraction Fail Phase** remain unchanged.

$\mathsf{Hyb}_4 \approx \mathsf{Hyb}_5$**:** Since the only change is made to the non-interactive commitment to the input in the first round of the **Initial Phase**, the analysis follows identically to that of the analysis used in proving $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$.

$\mathsf{Hyb}_5 \approx \mathsf{Hyb}_6$**:** We split our analysis into two cases:

– If at least one of the p_2 threads in **Initial Phase** do not abort in **Extend Mode**, we output a view that contains the simulated of the underlying protocol Π using the output received from querying the ideal functionality.

We remark that we additionally require the property of free-simulatability from the underlying protocol. This is because we sample coins honestly to go to **Extend Mode** conditioned on at least a single party voting for it. We need to send the second round messages of the underlying protocol on behalf of the honest parties that do not vote to extend the protocol. Thus, free simulatability of the underlying protocol guarantees that privacy of honest parties is maintained when a strict subset of second round messages are sent without knowledge of the output.

– If all p_2 threads in **Initial Phase** abort, then the subsequent thread is executed. Since there is no change in the **Subsequent Mode**, the views are identical except for the case that fail is output. This follows identically from the analysis of $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$ where the only change arises due to the estimation using the simulated first round messages.

$\mathsf{Hyb}_6 \approx \mathsf{Hyb}_7$**:** Here, we start making changes to the **Extraction Fail Phase**. Note that instead of using the honest parties inputs as in the previous hybrid, a cheat message is sent to $\mathcal{F}$ which returns the honest parties inputs. Ignore any additional information sent by $\mathcal{F}$. These returned inputs are then used to run the **Extraction Fail Phase** honestly. At this point we've stopped using the initialized honest inputs completely. The hybrid simulator continues to output the value computed by the honest parties.

Since the only difference is how we receive the honest inputs, Hyb_7 is identical to Hyb_6.

$\mathsf{Hyb}_7 \approx \mathsf{Hyb}_8$**:** We need to show that the probability of $\mathsf{corrupted}$ has only negligible difference from Hyb_7. This follows immediately from Lemma 1, and the discussion preceding the lemma.

$\mathsf{Hyb}_8 \approx \mathsf{Hyb}_9$**:** In this hybrid, we simply change how the outputs of the honest party are computed. The computed output on receiving the last round from the adversary are sent to $\mathcal{F}$. This is identical to the case that $\mathsf{corrupted}$ was not sent to $\mathcal{F}$. In the case that $\mathsf{corrupted}$ was sent to $\mathcal{F}$, the output of the honest parties remains identical.

5 3 Round Lower Bound

In this section, we show that it is impossible to achieve security against covert adversaries in *fixed* 3 rounds via black-box simulation. We shall do this by considering the **zero knowledge (ZK) functionality in the simultaneous message model**, where both the prover and verifier send messages in each round, including the last. Similar to the notion of promise zero-knowledge in [BGJ+18], zero-knowledge is only required to hold against verifiers that do not send invalid messages, *even in the last round of the protocol*.[3]

There are two main challenges in ruling out a protocol secure against covert adversaries in the simultaneous message model: (i) the setting of covert adversaries allows the simulator to send a cheat message to the ideal functionality if the verifier attempts to cheat. This allows the simulator to receive the prover's input to the functionality, the witness w, thereby allowing an easy completion of simulation; (ii) the setting of simultaneous message model allows for a simulator to potentially use the third round of the verifier to extract some form of trapdoor in order to produce a simulated transcript.

To rule out the latter simulation strategy while preventing the simulator to take the "easy way out" by sending a cheat message to the ideal functionality, we define an adversarial verifier V^* with the strategy "cheat if cheated with", which roughly translates to an adversarial verifier that will abort protocol execution if the protocol messages received are "incorrect". First we see why this prevents a simulator from taking the easy way out. In the real world, V^* behaves honestly while interacting with honest provers. This prevents the simulator from sending a cheat message to the trusted party to receive the prover's input since it would result in a distinguishable distribution from the real execution, where the V^* does not cheat.

Next, the rough idea to rule out any simulation strategy is the following: to simulate such a covert adversarial verifier V^*, the simulator needs to provide an accepting proof in the third round, else the rushing adversarial verifier V^* will always abort, thereby constraining the simulator to rewinding only the first or second rounds of the protocol. Rewinding the first round is not useful to the simulator because a rushing verifier can always restart afresh with new randomness. Thus the simulator can only rewind the second round. But the verifier can choose its second round message to be independent of the simulator's second round message, as in the honest protocol. Thus, the simulator seemingly gains no advantage by rewinding the second round. At best the simulator can make its own second round depend on the adversarial verifier's second round message.

Below we formally describe the strategy of the cheating verifier V^*. Next, we show that if there exists a simulator Sim^* that succeeds in simulating the view for adversarial verifier V^*, then the language must be in BPP. The proof borrows ideas from [GK96b], and we have kept notation largely the same for those familiar with the work in question.

Description of adversarial verifier V^*: We describe below the strategy of a rushing V^* that is parameterized by h a hash function from the family of t-wise independent hash functions. The value of t will be appropriately set. We denote by α_i and β_i the messages sent by the prover and verifier respectively. Let $\rho_{\mathsf{V}}(x, \mathsf{r}, \alpha_1, \alpha_2, \alpha_3) = \mathsf{ACCEPT}/\mathsf{REJECT}$ correspond to the decision output by the verifier V with randomness r.

1. Wait for first round message from the prover α_1.
2. Apply h on the received message to obtain randomness used to compute the first round of the protocol. Specifically, $\mathsf{r}_{\mathsf{V}^*} := h(\alpha_1)$ and the first round message computed as $\beta_1 := \mathsf{V}_1(1^\lambda; \mathsf{r}_{\mathsf{V}^*})$. Send β_1.
3. Compute the second round message of the protocol honestly independent of the second round received. Specifically, $\beta_2 := \mathsf{V}_2(\alpha_1; \mathsf{r}_{\mathsf{V}^*})$. Send β_2.

[3] This in particular means we cannot ignore the verifier's last message to transform the protocol to a three round protocol in the alternating message model as in [GMPP16].

4. If in the third (last) round of the protocol doesn't receive an accepting proof, abort. Else, send honest message in the third round.

We now formalize the intuition described earlier. For simplicity, we enforce the prover messages α_i to also contain all preceding messages. Any protocol can be transformed to this setting by the use of delimiters.

Consider a simulator Sim^* that succeeds in simulating the view of the verifier V^* described above. We describe below the procedure to run the simulator Sim^* as a subroutine without access to V^*. Looking ahead, we will build a decider for the language L that runs Sim^* as a subroutine.

Executing Sim^*. In order to execute the simulator Sim^* one needs to provide it the appropriate inputs where Sim^*'s output is determined by the input x, its random tape $\mathsf{R}_{\mathsf{Sim}^*}$ and the strings sent by the verifier V^*. Let the running time of the simulator be upper bounded by $t = t(\lambda)$. Fix t random strings $\mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}$, each of length $\ell(\lambda)$, where $\ell(\lambda)$ is the polynomial upper bound on the randomness needed by the V^*. [4]

When Sim^* outputs a pair (y, α) intended for V^*, abort if α is not of the specified format. Recall that the simulator can initialize the verifier with an input y potentially different from x. If α is of the specified format, we use one of the random strings $\mathsf{r}^{(i)}$ to respond to α in the following manner:

1. If α is the i-th unique first round message, denote this by $\alpha_1^{(i)}$; then use random string $\mathsf{r}^{(i)}$ to compute the response $\beta_1 = \mathsf{V}_1(y; \mathsf{r}^{(i)})$ and send it to Sim^*.

2. If α is a second round message, it contains within it the first round message. Check to see if the first round has already been queried. If not, abort. Else, let $\alpha_1^{(i)}$ correspond to the (previously queried i-th unique) first round message in α. Compute the response as $\beta_2 = \mathsf{V}_2(y, \alpha_1^{(i)}; \mathsf{r}^{(i)})$.

Note (i) the response β_2 depends only on the corresponding first round component of α; and (ii) the i-th unique first round message $\alpha_1^{(i)}$ is determined by x, $\mathsf{R}_{\mathsf{Sim}^*}$ and $\mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(i-1)}$.

We denote by $\mathsf{conv}_{\mathsf{Sim}}\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right) = (x, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3)$[5] the conversation output by the simulator Sim^* when executed with the specified inputs. Recall that we don't care about the third round response of the verifier since Sim^* needs to simulate verifiers that do not send their third round unless the proof is accepting. Thus, the simulator must be able to simulate without requiring the third round message of the verifier. This is important in the context of covert adversaries since in the real execution of the protocol, the verifier continues to behave honestly.

Definition 7. *We say that a vector* $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$ *is* Sim*-good if* $\mathsf{conv}_{\mathsf{Sim}}\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right) = (x, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3)$ *and* $\rho_{\mathsf{V}}\left(x, \mathsf{r}^{(i)}, \alpha_1, \alpha_2, \alpha_3\right) = \mathsf{ACCEPT}$ *where* i *is such that* $\alpha_1^{(i)} = \alpha_1$*, and the first and second round of the verifier are* $\beta_1 = \mathsf{V}_2(x, \mathsf{r}^{(i)})$ *and* $\beta_2 = \mathsf{V}_2(x, \alpha_1^{(i)}, \mathsf{r}^{(i)})$ *respectively. According to this value* i*, we call the conversation* (Sim^*, i)*-good or* i*-good.*

Theorem 2. *Let* $\langle \mathsf{P}, \mathsf{V} \rangle$ *be a 3 round zero-knowledge proof in the simultaneous message model for a language* L*. Suppose* $\langle \mathsf{P}, \mathsf{V} \rangle$ *is black-box simulatable against covert adversaries, and let* Sim^* *be the black-box simulator. Then,*

- *if* $x \notin L$*, only a negligible fraction of vectors are* $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$ Sim^**-good.*
- *if* $x \in L$*, a non-negligible fraction of vectors are* $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$ Sim^**-good.*

Before we look at the proof, let us look at the corresponding corollary which is relevant to us.

[4] We note that while we consider only strict polynomial time here, for an expected polynomial time simulator, we can convert it to a strict polynomial time simulator while only affecting the probability of success by a noticeable factor as shown in [GK96b].

[5] w.l.o.g. we assume the conversation outputs x, otherwise the resultant transcript would be distinguishable.

Corollary 1. *The corollary of the above theorem is that the language* $L \in \mathsf{BPP}$.

Proof. The decider works in the following manner. On input x;

1. select random values $\mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}$
2. if $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$ is Sim^*-*good*, then $x \in L$, else $x \notin L$.

Since Sim^* runs in polynomial time, our decider for L runs in polynomial time too. □

Proof. We now prove Theorem 2.

Negligible fraction when $x \notin L$. We prove this by contradiction. If there are a non-negligible fraction of vectors that are Sim^*-*good*, then we construct a cheating prover P^* violating the soundness of $\langle \mathsf{P}, \mathsf{V} \rangle$. Since there are a non-negligible fraction of vectors that are Sim^*-*good*, there is an index i such that non-negligible fraction of vectors are i-*good*. We now describe the cheating prover strategy.

P^*, interacting with an external verifier, picks a random index i^* from $[t]$. For all $i \in [t]$ such that $i \neq i^*$, pick random strings $\mathsf{r}^{(i)}$. Other than the i^*-th unique α_1 query, we answer the rest internally using $\mathsf{r}^{(i)}$. The i^*-th unique α_1 is forwarded to the external verifier. The response from the external verifier is fed as input to Sim^*.

If Sim^* queries a second round α_2 that has the same first round component as another second round message, we can just replay the same message β_2 sent by the external verifier. At the end of this process, we can use $\mathsf{conv}_{\mathsf{Sim}^*}$ to compute an accepting transcript. Since we picked the index i^* at random, with noticeable probability we now have an accepting proof for an external V^*. This violates the soundness condition since $x \notin L$.

Non-negligible fraction when $x \in L$. Consider the behavior of cheating verifiers described above. Each of them is associated with a hash function sampled from a family of $t = t(\lambda)$-wise independent hash functions. This guarantees that rewound verifiers will use "new randomness" at each call.

For each hash function $h \in H_\lambda$ we associate an adversarial verifier V^*_h, which responds to the prover's message by setting the randomness to be used to be $h(\alpha_1)$ where α_1 is the first round message received. Let $\nu(x, \mathsf{R}_{\mathsf{Sim}^*}, h) = (x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)})$.

Claim 6. *For $x \in L$, there are a non-negligible fraction of pairs $(\mathsf{R}_{\mathsf{Sim}^*}, h)$ such that $\nu(x, \mathsf{R}_{\mathsf{Sim}^*}, h)$ is Sim^*-good.*

Proof. For any input x, let p_x be the probability that P convinces an honest verifier V to accept x, where the probability is over the random coins of both P and V. From the completeness property, p_x is non-negligible.

Since the output of h is uniformly distributed, the probability that P and honest verifier V output an accepting conversation p_x is the same as the probability that P and constructed verifier V_h output an accepting proof.

Since the simulator Sim^* succeeds in simulating the transcript for all V^*_h, the probability that Sim^* outputs an accepting conversation when interacting with h is $p_x - \mathsf{negl}(\lambda)$. Thus, the probability over $\mathsf{R}_{\mathsf{Sim}^*}, h$ that $\nu(x, \mathsf{R}_{\mathsf{Sim}^*}, h)$ is Sim^*-*good* is $p_x - \mathsf{negl}(\lambda)$, which is non-negligible. □

Now, from the $t(\lambda)$-wise independence property of the family H_λ, we have that $\nu(x, \mathsf{R}_{\mathsf{Sim}^*}, h)$ is uniformly distributed over $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$. This trivially follows from the fact that the hash h is applied only to distinct elements.

This completes the proof since this establishes that non-negligible fraction of $\left(x, \mathsf{R}_{\mathsf{Sim}^*}, \mathsf{r}^{(1)}, \cdots, \mathsf{r}^{(t)}\right)$ are Sim^*-*good*.

□

Acknowledgments

Vipul Goyal is supported in part by the NSF award 1916939, a gift from Ripple, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

Arka Rai Choudhuri and Abhishek Jain are supported in part by DARPA/ARL Safeware Grant W911NF-15-C-0213, NSF CNS-1814919, NSF CAREER 1942789, Samsung Global Research Outreach award and Johns Hopkins University Catalyst award. Arka Rai Choudhuri is also supported by NSF Grants CNS-1908181, CNS-1414023, and the Office of Naval Research Grant N00014-19-1-2294.

References

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 468–499, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[BGJ+13] Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 316–334, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BGJ+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 459–487, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 645–677, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[CCG+19] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. Cryptology ePrint Archive, Report 2019/216, 2019. https://eprint.iacr.org/2019/216.

[COSV17a] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 711–742, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[COSV17b] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Four-round concurrent non-malleable commitments from one-way functions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 127–157, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[COSV17c] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 678–710, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st FOCS*, pages 283–293, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, June 1996.

[GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 448–476, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 695–704, San Jose, CA, USA, June 6–8, 2011. ACM Press.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599, Berkeley, CA, USA, October 15–17, 2017. IEEE Computer Society Press.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 488–520, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 132–150, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 578–595, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *36th ACM STOC*, pages 232–241, Chicago, IL, USA, June 13–16, 2004. ACM Press.

[PW10] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from sub-exponential one-way functions. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 638–655, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st FOCS*, pages 531–540, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.