On the Round Complexity of Covert Computation

Vipul Goyal
Microsoft Research, India
vipul@microsoft.com

Abhishek Jain
UCLA
abhishek@cs.ucla.edu

Abstract

In STOC'05, von Ahn, Hopper and Langford introduced the notion of covert computation. In covert computation, a party runs a secure computation protocol over a covert (or steganographic) channel without knowing if the other parties are participating as well or not. At the end of the protocol, if all parties participated in the protocol and if the function output is "favorable" to all parties, then the output is revealed (along with the fact that everyone participated). All covert computation protocols known so far require a large polynomial number of rounds. In this work, we first study the question of the round complexity of covert computation and obtain the following results:

- There does not exist a constant round covert computation protocol with respect to black box simulation even for the case of two parties. (In comparison, such protocols are known even for the multi-party case if there is no covertness requirement.)
- By relying on the two slot non-black-box simulation technique of Pass (STOC'04) and techniques from cryptography in NC^0 (Applebaum et al, FOCS'04), we obtain a construction of a constant round covert multi-party computation protocol.

Put together, the above adds one more example to the growing list of tasks for which non-black-box simulation techniques (introduced in the work of Barak in FOCS'01) are necessary.

Finally, we study the problem of covert multi-party computation in the setting where the parties only have point to point (covert) communication channels. We observe that our covert computation protocol for the broadcast channel inherits, from the protocol of Pass, the property of secure composition in the *bounded concurrent setting*. Then, as an application of this protocol, somewhat surprisingly we show the existence of covert multi-party computation with point to point channels (assuming that the number of parties is a constant).

1 Introduction

A conventional steganographic framework allows for encoding hidden messages within larger, seemingly innocent messages. This allows a group of parties to realize a "covert channel" on which they can meaningfully communicate without leaking even the existence of such a communication to "undesirable entities". The notion of *covert computation* was introduced by von Ahn et al [vAHL05] as a generalization of this framework. Covert computation allows a group of parties to securely perform arbitrary computation without leaking the existence of this computation to undesirable entities (some of whom may even be potential participants of this computation).

To understand the notion of covert computation, consider the example of secret handshake: a problem well studied in the security community (see [ABK07, BDS+03] and the references therein). Consider the scenario where two CIA agents, both of whom possess some signed credentials, wish to converse with each other. However, none of them wishes to reveal the fact that he is a CIA agent unless the other one is a CIA agent as well. It seems that in such a setting, one can run a well known "ordinary" secure computation protocol [GMW87, Yao82] which would output 1 if both of them are CIA agents and 0 otherwise. However, for this to work, one of the parties needs to come forward and ask the question: "shall we run a protocol to see if both of us are CIA agents?". This already reveals the information that this party is probably a CIA agent! Is it possible for the agents to run a secure computation protocol such that even the existence of the protocol execution is not known until and unless the credentials of both agents are verified and valid (and they both participated in the protocol)?

von Ahn et al [vAHL05] gave a construction for covert two-party computation satisfying a *game based* definition. Subsequent to this work, Chandran et al [CGOS07] proposed a cleaner simulation based definition of covert computation by strengthening the "standard" secure computation ideal world requirements. Chandran et al [CGOS07] also proposed a construction of *covert multi-party computation* satisfying this definition assuming the existence of a (covert) communication channel shared among all participants (that is, a broadcast channel).

Very informally, the scenario of covert computation is as follows. There are various parties who carry out normal "innocent" conversations with each other. There might be parties which are interested in carrying out a protocol for a particular task. Such *participating* parties compute outgoing messages as per the protocol specifications and encode these messages in the innocent looking messages of the channel (using steganographic techniques). We refer to this as a "covert communication channel". Upon receipt of a message, a participating party runs the decoding procedure to get the next incoming message of the protocol. At the end of the protocol, if every party was participating and the output of the computation was "favorable"[1], the parties may learn the output and the fact that everyone participated. A covert computation protocol should satisfy the following security properties (in addition to those already satisfied by standard secure computation protocols): (1) No outsider can tell if the parties are running a protocol or just communicating normally, (2) Even if one party does not take part in the protocol, or the output of the computation is unfavorable, then the transcript of the conversation looks indistinguishable from "innocent looking" conversations *even to a party participating in the protocol*. It is the second property which makes the construction of covert computation protocols interesting and highly non-trivial.

It was shown in [vAHL05] that using standard steganographic techniques, the task of designing a covert computation protocol can be reduced to the task of designing an ordinary secure computation protocol where a message sent by a party looks indistinguishable from random to all other parties (and the outside world) until and unless the output is favorable to all parties (and all parties participated as opposed to simply sending messages drawn from the uniform distribution). von Ahn et al [vAHL05] discuss several applications of covert computation such as *dating* (where the existence of computation is revealed only if both parties are interested in each other) or bit agreement in general (to see if all the parties are like-minded), cheating in Internet games and so on.

The notion of covert computation is theoretically intriguing (independent of its applications). Covert computation can be seen to have the same relationship with ordinary secure computation as steganographic

[1] In the secret handshake example above, this would amount to the credentials of both parties being valid.

communication has with encrypted communication. Covert computation is an example of a protocol problem that is formulated by a strengthening of the ideal world security requirements (as opposed to specifications of a new real-world setting in which protocols providing ordinary ideal world security are desired). Covert computation comes with various fundamental technical challenges unrelated to the kind encountered before in the literature of secure computation. Standard techniques like **zero knowledge proofs inherently break down** in the setting of covert computation. This is because if a party is able to "verify" a proof given by another party, the fact that the other party is participating is immediately revealed.

There are a few natural questions that arise while considering the notion of covert computation. These issues were discussed in [vAHL05], and we briefly recall them here:

Synchronization. One question we may ask is "don't parties have to tell each other that they are interested in running a covert computation protocol for function f anyway?". This is not the case as discussed in [vAHL05]: As part of protocol specification, we could have information about how and when to initiate a particular protocol. This could be in the form of "If we wish to compute function f, then the protocol begins with the first message exchanged after 2 p.m.". If a party is interested in computing function f, then he starts hiding the protocol messages in the "ordinary looking" messages starting from 2 p.m. Of course, it could be the case that other parties are not interested, in which case some parties would be executing the protocol, while others would just be carrying on "normal" conversations. The protocol execution does not take place in this case, and knowledge about which parties intended to take part in the protocol is not revealed to anyone. (Instead, other parties will just observe ordinary-seeming conversations.)

MPC + Steganography = Covert MPC? Is it the case that by combining multi-party computation and steganography, we can obtain covert computation protocols? Again, as argued by [vAHL05], this is not the case: Recall that steganographically encoding all messages of a standard secure multi-party computation protocol would yield a protocol for which no outside observer can determine if the protocol for computing the function is being run or not. This however provides no guarantee whatsoever to the participants of the protocol itself. Covert computation must guarantee that the protocol execution remains hidden from participating parties.

We assume that all parties know a common circuit for the function f and further, they know the roles they will play in the protocol. We call a party a non-participating party if it does not play the required role during protocol execution. The parties also possess information about the synchronization of the protocol. Adversarial parties also know all such details.

1.1 Round Complexity of Covert Computation

Round complexity is a parameter of special importance in the setting of covert computation. In this setting, the issues of round complexity may relate to questions about *feasibility* of secure computation as opposed to just efficiency. This is because, a party may not be able to exchange hundreds of messages back and forth with another party without raising the suspicion that a computation is going on. In general, *the covert communication channel might restrict the number of messages which the parties are allowed to send.*

The protocols constructed in [vAHL05, CGOS07] require a large polynomial number of rounds in the security parameter as well as the number of parties. In order to obtain more realistic protocols for covert computation, it is important to ask whether this is inherent.

1.2 Our Contributions

We first study the question of the round complexity of covert computation and obtain the following results:

1. There does not exist a constant round covert computation protocol with respect to black box simulation even for the case of two parties. (In comparison, such protocols are known even for the multi-party case if there is no covertness requirement.)

2. Next, we build upon on the two slot non-black-box simulation technique of Pass [Pas04], and the techniques from Applebaum et al [AIK04] to construct (based on standard assumptions) a "covert"

encryption scheme with a constant depth circuit for the encryption function. Using these techniques, we obtain a construction of a constant round covert multiparty computation protocol.

Put together, the above adds one more example to the growing list of tasks for which non-black-box simulation techniques (introduced in the work of Barak [Bar01]) are necessary.

Finally, we initiate the study of covert multiparty computation in the setting where the parties only have point to point covert communication channels. This setting is fundamentally different from the one where a covert broadcast channel is available. This is because now a party may not have an innocent reason to send the same message to multiple other parties. Doing such might immediately reveal that the party is trying to carry out a protocol. Thus, the known techniques of implementing broadcast using point-to-point channels inherently fail. At any stage in the protocol, a non-trivial agreement between more than two parties on the same message implies the loss of covertness.

Towards that end, we observe that our covert computation protocol for the broadcast channel inherits, from the protocol of Pass [Pas04], the property of composition in the bounded concurrent setting. Then, as an application of this protocol, we provide the first construction of covert multiparty computation with point to point channels assuming that the number of parties is a constant.

1.3 Our Techniques

We now give a high-level overview of the techniques used in each of our results.

Impossibility of Constant-Round Covert Computation with Black Box Simulation. One of the key requirements for successful simulation in the setting of secure two-party computation is the following. The simulator must be able to extract the input of the adversary (and send it to the trusted party in the ideal world). Black-box simulators are restricted to do that by means of rewinding (i.e., by making extra oracle calls to the adversary). In particular, a black-box simulator may run multiple "threads of execution" with the adversary and finally output a single thread, called the "main-thread". Any other thread is referred to as a "look-ahead" thread. A black-box simulator "will fail" if: (a) the main thread is honestly executed (e.g., the adversary does not abort at any point during the main thread), but (b) the simulator obtains no "useful information" in *any* look-ahead thread. In order to prove our impossibility result, we will construct an adversary for which any black-box simulator "fails" (in the aforementioned sense) with noticeable probability.

Recall that in the setting of covert two-party computation, a party p_1 may either send a meaningful message (i.e., a message that encodes a protocol message) or a "non-participating" message (i.e., a message drawn uniformly from the broadcast channel). The covertness property requires that these two events must be indistinguishable to the other party p_2. In particular, for any secure covert computation protocol, the covertness property must hold even if p_1 and p_2 (as described above) are in fact the adversary and the simulator algorithms respectively. Then, the central idea to our impossibility result is the inability of any black-box simulator (say) $\mathcal{S}$ of a covert computation protocol to determine whether an adversarial party is participating in the main thread or not (at least until additional information can be extracted via rewinding). First, consider a simple adversarial party $\mathcal{A}$ who does not participate in the covert computation *at all* (i.e., sends messages drawn randomly from the communication channel). However, if $\mathcal{A}$ does not participate at all, at some point (after rewinding $\mathcal{A}$ a number of times), $\mathcal{S}$ must give up in trying to extract its input (and conclude that $\mathcal{A}$ is not participating).

Now, consider the case when $\mathcal{A}$ participates in each round of the covert computation protocol with some sufficiently low probability. In this case, the following could happen with non-negligible probability (if the protocol is constant round). $\mathcal{A}$ might participate honestly throughout in the "main thread"; however, $\mathcal{S}$ will not have any evidence of this fact (in keeping with the *covertness* property of the protocol). Further, when $\mathcal{S}$ rewinds $\mathcal{A}$ to create a number of "look-ahead" threads, $\mathcal{A}$ might not participate in any of them with noticeable probability. Thus, $\mathcal{S}$ might give up as in the first case and conclude that $\mathcal{A}$ is not participating at all (as opposed to running till it is able to extract the input).

We note that our proof only works for the constant round case. This is because if the number of the rounds in the protocol are non-constant, very roughly speaking, the simulator may have a large number of

rewinding opportunities. As a result, the probability that the adversary doesn't participate in any of the look aheads (but participates in the entire main thread) may become negligible.

To summarize, since the simulator is unable to determine whether the adversary participated in the main thread or not (at least until one look-ahead thread is successful), it runs the risk of creating "too few" look ahead threads (such that the adversary does not participate in any of them) and then giving up in extraction. Roughly speaking, if the simulator is still successful, then it must be essentially "straight-line". Then, by using standard techniques, we can derive a contradiction based on this fact.

Constant-Round Covert Computation with Non Black Box Simulation. In many works, constant round multi-party computation protocols have been designed using techniques from Beaver et al [BMR90]. The design of the protocol can be viewed as having two high level steps:

Semi-honest protocol. The first step is to design a constant round protocol secure only against semi-honest adversaries. The basic idea is to have the parties run a secure computation protocol whose round complexity is linear in the depth of the circuit it evaluates (e.g., the semi-honest GMW protocol). Such a secure computation protocol is run to jointly generate a garbled circuit [Yao82] for the desired (final) functionality. The parties can then evaluate this garbled circuit offline on their own later on. Note that the gate tables for all the gates in the garbled circuit are generated in *parallel*; then, if the circuit to generate any gate table has *constant* depth, the secure computation protocol will only have a *constant* number of rounds.

This is indeed the case in [BMR90] since they simply use the XOR function as the encryption scheme to generate the gate tables. However, note that fresh keys must be used for each XOR operation. Then, to avoid blowing up the size of the keys exponentially, the parties now need to run a local preprocessing phase in which they expand their keys by application of a pseudorandom generator (PRG). However, the basic BMR technique breaks down in the covert setting. The high level reason can be understood as follows. The PRG evaluations in local preprocessing phase (in which the parties expand their keys) have to be later repeated at the time of garbled circuit evaluation. Then, the fact that the computation done in the preprocessing phase "conforms" to the computation done while evaluating the received garbled circuit leaks the fact that all parties are participating in the protocol (even if the output is not favorable). In summary, the XOR function coupled with the PRG evaluations does not preserve covertness of the parties.

To solve this problem, we construct a new covert encryption scheme[2] in NC^0 using techniques from Applebaum et al [AIK04]. We then use this encryption scheme (instead of the XOR function as in [BMR90]) to compute the gate tables. This allows us to remove the BMR local processing phase entirely.

Forgetting our goal of obtaining a covert computation protocol, consider the protocol in which we use such an encryption scheme in gate tables and use standard GMW for gate table generation. We believe this protocol is of independent interest since it also gives an arguably cleaner alternative to the Beaver et al [BMR90] protocol (which in turn has been used widely in the study of round complexity of secure computation).

Compiling with Zero-Knowledge Proofs. Next, we "compile" the above semi-honest secure protocol with constant round *simulation-sound*[3] zero knowledge proofs.

In order to adopt such an approach to our setting, we first note that zero knowledge proofs break down in the setting of covert computation since they are "verifiable". To this end, [CGOS07] introduced the notion of *zero knowledge proofs to garbled circuits*. Roughly speaking, a zero knowledge proof to garbled circuit is a protocol between two parties—a sender and a receiver—who share common input (x, L) while the sender additionally has a private input v. If the receiver can prove that $x \in L$ to a garbled circuit prepared by the

[2]Intuitively, the encryption keys and the ciphertexts produced by a covert encryption scheme are indistinguishable from the uniform distribution.

[3]In order to ensure that the resultant protocol is constant-round (in the multi-party case), not only must the zero-knowledge proofs be constant-round, but they must also be executed in *parallel*. We note that in such a scenario, we run the risk of mauling attacks (for instance, an adversary who acts as the verifier in a "left" execution may be able to use this interaction in order to prove a statement in a "right" execution). The notion of simulation-soundness, introduced by Sahai [Sah99], guarantees robustness against such attacks; therefore using simulation-sound zero knowledge proofs (instead of standard zero-knowledge) is crucial in the constant-round setting.

sender, then it will receive a private value v, else it will receive a random value. Intuitively, this protocol can be seen as covert conditional oblivious transfer, where a value is sent from a sender to a receiver conditioned upon the fact that a given NP statement (for e.g., an assertion that a party behaved honestly "so far" in the protocol) is true. In this protocol, the sender does not learn whether or not the receiver was able to prove the statement correctly to the garbled circuit (hence preserving the covertness of the zero-knowledge prover).

Our impossibility result implies that non black-box techniques are necessary to construct such a gadget (henceforth referred to as ZKSend) in the constant-round setting. To this end, we use the non black-box simulation technique of Pass [Pas04] (which in turn builds on the work of Barak [Bar01]). We note that the protocol of Pass is a *simulation-sound* zero knowledge argument system.

We stress that a naive attempt to adopt the techniques of [Pas04] in the construction of ZKSend of [CGOS07] does not work. The main difficulty is in proving the (stand-alone) soundness of the new construction. Complications arise in our setting because the construction of ZKSend in [CGOS07] crucially uses a *proof* system, while the protocol in [Pas04] is an *argument* system. We refer the reader to section 5.1 for more details.

Finally, we note that zero-knowledge proofs to garbled circuits cannot be applied to achieve secure covert multi-party computation as "easily" as ordinary zero-knowledge proofs can be applied to achieve secure (ordinary) multi-party computation. This problem was also faced by Chandran et al [CGOS07] who introduced what one can call a "delayed verification" technique (where the usage of zero-knowledge protocols is deferred to the end of the protocol). Here we are able to adopt the techniques from [CGOS07] to our setting without major difficulties.

Covert Computation over Point-to-Point Channels. There exists a rich body of literature on designing secure computation protocols over point to point channels (see [KK07],[KKK08] and references therein). However, to our knowledge, a common theme in all these works is a party sending the same message to multiple (or all) other parties over the pairwise private channels. Unfortunately, such techniques are inherently bound to fail in our scenario. The key challenge in our setting is to design a protocol where the messages exchanged between a pair of parties look indistinguishable from random even given the messages exchanged between all other pairs of parties (till the point when it is clear that all the parties are participating and that the output is favorable).

The basic idea of our construction is as follows. As part of the protocol specifications, the n parties are grouped into $n/2$ pairs. Each pair of parties run a covert two-party computation protocol (say) Σ to emulate a *virtual party*. This leads to a total of $n/2$ virtual parties. These virtual parties are further grouped into $n/4$ pairs and each pair of virtual parties run Σ to emulate another virtual party. By applying this idea recursively, there would eventually be a single virtual party which has all the required inputs and thus computes the output. Very informally, even if a single (real) party behaves honestly in the protocol, the final single virtual party would be "honest" as well.

We model a virtual party in the form of a *reactive* functionality. In particular, we define a hierarchy of reactive functionalities where the functionality at level 0 has all the required inputs and can compute the output while each functionality at level $\log(n)$ defines the algorithm of a real party. Further, we show how a pair of virtual parties at some level can communicate with each other through parties at the level just "below". We now briefly outline some of the key challenges in realizing this construction.

Note that in the above construction, there would be multiple uncoordinated executions of the covert two-party computation protocol Σ. To this end, we will require Σ to be secure in the *bounded concurrent* setting. Fortunately, our constant-round covert computation protocol (over the broadcast channel) inherits, from the protocol of Pass [Pas04], the property of composition in the bounded concurrent setting. Then, by making some necessary changes to this protocol, we can obtain a covert two-party computation protocol Σ for reactive functionalities in the bounded-concurrent setting. Further, note that the communication channel of the virtual parties (as described above) can, in general, be controlled by a man in the middle. We employ techniques from the work of Barak et al [BCL$^+$05] to solve this problem.

We note, however, that our construction only works for a constant number of parties. This is because the

computational overhead of our protocol is $O(k^{\log(n)}) \cdot |C|$ where k is the security parameter, n is the number of parties (and hence $\log(n)$ is the depth of the tree) and $|C|$ is the size of the circuit of the functionality. One approach to extend our construction to work for any polynomial number of parties would be to obtain a covert two-party computation protocol with *constant computational overhead* [IKOS08].

1.4 Organization

We start by describing our model of covert computation in section 2. In section 3, we present our impossibility result. Next, in section 4, we introduce some of the basic building blocks used in our positive result. We then present our positive result for the broadcast channel in section 5. Finally, we present our positive result for point-to-point channels in section 6. Additionally, in appendix D, we provide a (relatively) short description of all our results. This is recommended for a reader who only wishes to understand the key ideas in our results without all the details.

2 Preliminaries and Model

In this section, we describe our model for covert computation. Specifically, we consider the simulation based definition for covert computation introduced by Chandran et al [CGOS07]. The text in this section is taken verbatim from [CGOS07].

2.1 Network Model

We consider a system of n parties who interact with each other. Each of the parties could either be trying to compute some function jointly with other parties (hoping that other parties are interested in computing this function too), or could just be carrying out normal day to day conversation. As in [vAHL05, CGOS07], we envision that the protocol messages are hidden by the parties in "ordinary" or "innocent-looking" messages (using stenographic techniques). Ordinary communication patterns and messages are formally defined in a manner similar to the channels used by [HLvA02] and [vAH04]. Similar to [CGOS07], we use a *broadcast channel* shared by all n participants in the protocol. For a more detailed description, see Appendix A.

Similar to [vAHL05], we note that it is enough to construct covert computation protocols only for the uniform channel. This is due to the following lemma:

Lemma 1 *If* Π *covertly realizes the functionality* f *for the uniform channel, then there exists a protocol* Σ^{Π} *that covertly realizes* f *for the broadcast channel* B*.*

The intuition behind why this lemma is true is that once we construct a protocol where the messages of the parties look indistinguishable from random, we can use steganographic techniques to embed such messages into innocent conversations of any other channel. Upon decoding a given conversation for the channel, a party would only see a random looking message regardless of whether a message was embedded in the conversation or not. We refer the reader to [vAHL05] for a description of Σ^{Π} and a proof of the statement.

Here onwards, we shall concentrate only on constructing covert computation protocols for which it can be shown that all messages in the protocol are indistinguishable from messages drawn at random from the uniform distribution. By using the above compiler, we can then obtain a covert computation protocol for any broadcast channel B.

2.2 Covert Computation

We consider a system of n parties (say, $\mathcal{P} = \{P_1, \ldots, P_n\}$), of which some may be trying to run the covert computation protocol while others could just be carrying out regular conversation. This gives rise to so called participating parties and non-participating parties. If any of the parties is non-participating, we would like the covert computation protocol to just output $\perp$ to all parties (hiding which party participated and which did not). Each participating party P_i holds an input x_i. Our (possibly randomized) function

f to be computed by the n parties is denoted by $f : (\{0,1\}^*)^n \to \{0,1\}^*$, that is, for a vector of inputs $\vec{x} = (x_1, \ldots, x_n)$, the output is $f(\vec{x})$ (the case where different parties should get different outputs can be easily handled using standard techniques). Note that we may also view f as a deterministic function when the randomness is explicitly supplied as input. Additionally, we have a function $g : (\{0,1\}^*)^n \to \{0,1\}$ which determines whether the output is favorable to all the parties or not. In particular, if $g(\vec{x}) = 0$, the output $f(\vec{x})$ is said to be non-favorable. In that case, the covert computation protocol should just produce $\perp$. No one can tell whether any party participated at all or not even after the protocol is over. However, if every party is a participating party and $g(\vec{x}) = 1$, we would like the output of the protocol to be $f(\vec{x})$ (and thus the fact that everyone participated becomes public).

To formalize the above requirement, we extend the standard paradigm for defining secure multi-party computation. We define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model.

In a given execution of the protocol we assume that all inputs have length κ, the security parameter. We consider a static adversary who corrupts up to $n-1$ of the players before execution of the protocol. We also assume a synchronous network with rushing. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time.

IDEAL MODEL. In the ideal model there is a trusted party which computes the desired functionality based on the inputs and the participation data handed to it by the players. An execution in the ideal model proceeds as follows:

Inputs Each participating party P_i has input x_i. We represent the vector of inputs by $\vec{x}$.

Send inputs to trusted party Honest participating parties always send their inputs to the trusted party. Honest non-participating parties are assumed to send $\perp$ to the trusted party (looking ahead, this corresponds to sending regular uniformly selected messages according to the broadcast channel in the real world). Corrupted parties, on the other hand, may decide to send modified values or $\perp$ to the trusted party. We do not differentiate between malicious participating parties and malicious non-participating parties (and assume that malicious parties are free to behave whichever way they want).

Trusted party computes the result If any of the parties sent a $\perp$ as input, the trusted party sets the result to be $\perp$. Otherwise, let $\vec{x}$ denote the vector of inputs received by the trusted party. Now, the trusted party checks if $g(\vec{x}) = 0$ and sets the result to be $\perp$ if the check succeeds. Otherwise, the trusted party sets the result to be $f(\vec{x})$. It generates and uses uniform random coin if required for the computation of $g(\vec{x})$ or $f(\vec{x})$.

Trusted party sends results to adversary The trusted party sends the result (either $f(\vec{x})$ or $\perp$) to the adversary.

Trusted party sends results to honest players The adversary, depending on its view up to this point, prepares a (possibly empty) list of honest parties which should get the output. The trusted party sends the result to the parties in that list and sends $\perp$ to others.

Outputs An honest participating party P_i always outputs the response it received from the trusted party. Non-participating parties and corrupted parties output $\perp$, by convention. The adversary outputs an arbitrary function of its entire view (which includes the view of all malicious parties) throughout the execution of the protocol.

Observe that the main difference between our ideal model (for covert computation) and the ideal model for standard secure multi-party computation (MPC) is with respect to participation data. In standard MPC, it is implicitly assumed that every player is taking part in the protocol and that this information is public. In covert computation, if the result from the trusted party is $\perp$, the adversary cannot tell whether this is

because some party did not participate (or if any honest party participated at all) or because the output was not favorable. If all the parties participate and the output is favorable, only then the adversary learns the output and the fact that everyone did participate. We do not consider fairness (of getting output or learning about participation) in the above model. As pointed out above, we do not differentiate between malicious participating parties and malicious non-participating parties (and assume that malicious parties are free to behave whichever way they want). However, differentiation between honest non-participating parties and malicious parties is obviously necessary since the adversary should not know which parties are non-participating and which are participating among the honest party set (while the adversary fully controls and knows everything about malicious parties).

For a given adversary $\mathcal{A}$, the *execution of* (f, g) *in the ideal model* on participation data $\vec{PD}$ (which contains the information about which of the parties are participating) and input $\vec{x}$ (assuming $\perp$ as the input of non-participating parties) is defined as the output of the parties along with the output of the adversary resulting from the process above. It is denoted by $\text{IDEAL}_{f,g,\mathcal{A}}(\vec{PD}, \vec{x})$.

REAL MODEL. Honest participating parties follow all instructions of the prescribed protocol, while malicious parties are coordinated by a single adversary and may behave arbitrarily. Non-participating parties are assumed to draw messages uniformly from the broadcast channel. At the conclusion of the protocol, honest participating parties compute their output as prescribed by the protocol, while the non-participating parties and malicious parties output $\perp$. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol. For a given adversary $\mathcal{B}$ and protocol Π for covertly computing f, the *execution of* Π *in the real model* on participation data $\vec{PD}$ and input $\vec{x}$ (denoted $\text{REAL}_{\Pi,\mathcal{B}}(\vec{PD}, \vec{x})$) is defined as the output of the parties along with the output of the adversary resulting from the above process.

Having defined these models, we now define what is meant by a secure protocol. By *probabilistic polynomial time* (PPT), we mean a probabilistic Turing machine with non-uniform advice whose running time is bounded by a polynomial in the security parameter κ. By *expected probabilistic polynomial time* (EPPT), we mean a Turing machine whose *expected* running time is bounded by some polynomial, for *all* inputs.

Definition 1 *Let f, g and Π be as above. Protocol Π is a* t-secure protocol for computing (f, g) *if for every* PPT *adversary $\mathcal{A}$ corrupting at most t players in the real model, there exists an* EPPT *adversary $\mathcal{S}$ corrupting at most t players in the ideal model, such that:*

$$\left\{\text{IDEAL}_{f,g,\mathcal{S}}(\vec{PD}, \vec{x})\right\}_{\vec{PD}\in(\{0,1\})^n, \vec{x}\in(\{0,1\}^*)^n} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}}(\vec{PD}, \vec{x})\right\}_{\vec{PD}\in(\{0,1\})^n, \vec{x}\in(\{0,1\}^*)^n}.$$

3 Impossibility of constant-round covert computation with a black-box simulator

In this section, we show the existence of probabilistic polynomial-time (PPT) computable covert two-party functionality for which there does not exist any constant-round covert two-party computation protocol, with a black-box simulator. We stress that our impossibility result rules out any expected polynomial time simulator which uses the adversarial algorithm as an oracle.

Before proceeding to the formal proof, we first give some intuition behind our impossibility result. The central idea to our proof is the inability of any black-box simulator (say) $\mathcal{S}$ of a covert computation protocol to determine whether an adversarial party is participating in the "main thread" or not. First, consider a simple adversarial party $\mathcal{A}$ who does not participate in the (covert) computation *at all*. Then, black-box simulators for two-party computation protocols have to first extract the adversary's input; however, if $\mathcal{A}$ does not participate at all, at some point, $\mathcal{S}$ must give up in trying to extract its input (and conclude that $\mathcal{A}$ is not participating).

Now, consider the case when $\mathcal{A}$ participates in the (covert) computation with some sufficiently low probability. In this case, the following could happen with non-negligible probability (if the protocol is constant round). $\mathcal{A}$ might participate honestly throughout in the "main thread"; however, $\mathcal{S}$ will not have

any evidence of this fact (in keeping with the *covertness* property of the protocol). Further, when $\mathcal{S}$ rewinds $\mathcal{A}$ to create a number of "look-ahead" threads, $\mathcal{A}$ might not participate in any of them with noticeable probability. Thus, $\mathcal{S}$ might give up as in the first case and conclude that $\mathcal{A}$ is not participating at all (as opposed to running till it is able to extract the input). In fact, as we show later on, this happens even if the simulator chooses the main thread *adaptively*. Informally speaking, this means that the simulator fails to get any useful information from any look-ahead thread with noticeable probability. However, a simulator for any two-party computation protocol must have some additional power over a real adversary, and the only additional power awarded to a *black-box* simulator is essentially the ability to rewind the adversary. We therefore conclude that black-box simulators (even expected polynomial-time) cannot exist for constant-round covert two-party computation protocol, as stated in theorem 1 below.

We note that our proof only works for the constant round case. This is because if the number of the rounds in the protocol are non-constant, very roughly speaking, the simulator may have a large number of rewinding opportunities. As a result, the probability that the adversary doesn't participate in any of the look aheads (but participates in the entire main thread) may become negligible. We now formally state our claim.

Theorem 1 *There exist PPT computable two-party functionalities for which there do not exist any* constant-round *covert two-party computation protocol as per Definition 1 with respect to* black-box *simulators (unconditionally).*

We prove the above theorem in the next subsection. We refer the reader to appendix D.1 for a concise proof sketch.

3.1 Proof of Theorem 1

We will organize our proof into two main parts.

1. First, consider any covert two-party functionality F. Let Π be any *constant*-round covert two-party computation protocol that securely realizes F with respect to a black-box simulator. Then, we first construct an adversary for Π and derive a lower bound on the probability p_{fail} with which *every* black-box simulator for Π gets full participation from the adversary in the "main thread", but fails to get any "useful" information from the rewindings. Proving this lower bound is in fact the crux of our proof.

2. Next, we consider a *specific* covert two-party functionality $\hat{F}$ (described later). Let $\hat{\Pi}$ be any constant-round covert two party computation protocol that securely realizes $\hat{F}$. Let $\mathcal{S}$ be any black-box simulator for $\hat{\Pi}$. We first show that for the case of functionality $\hat{F}$, probability p_{fail} (as defined above) is in fact noticeable (polynomially related to the security parameter κ). We then show how to construct a cheating party that internally uses $\mathcal{S}$ in an execution of $\hat{\Pi}$ such that the output distributions of the real world and the ideal world executions are distinguishable with non-negligible probability. This contradicts the assumption that $\hat{\Pi}$ securely realizes $\hat{F}$.

Combining the two parts, we conclude that for the covert two-party functionality $\hat{F}$ (as described later), there do not exist any constant-round covert two-party computation protocol that securely realizes $\hat{F}$ as per Definition 1 with respect to black-box simulators. We note that this is sufficient to prove theorem 1. We now give more details on both parts of our proof.

PART 1. Analyzing the probability of simulator failing in all rewindings
Let F be any covert functionality for two parties (P_1, P_2). Let Π be any constant-round covert two party computation protocol that securely realizes F with respect to a black-box simulator. We will first construct a cheating P_1^* that is of the following form: P_1^* behaves exactly as the honest party P_1, except that it replies honestly in each round of Π only with some probability $p < 1$. With probability $1 - p$, P_1^* simply sends a message as if it were not participating in the protocol. We will set the probability p such that P_1^* still

participates honestly in Π with some non-negligible probability. We will then derive an expression for the lower bound on the probability p_{fail} with which any black-box simulator $\mathcal{S}$ for Π gets full participation from the adversary in the "main thread", but fails to get any "useful" information from the rewindings.

We now give more details. We first introduce some notations and conventions for the remainder of the proof. We will borrow some of our notations and conventions from [BL04].

Without loss of generality, we assume that P_2 is the protocol initiator in Π. Let c be the number of rounds in Π, where one round consists of a message from P_2 followed by a reply from P_1. Let $\{\mathcal{T}_k\}_k$ be a family of q-wise independent predicates, where $t \in \mathcal{T}_k$ maps $\{0,1\}^{\leq \text{poly}(\kappa)}$ to $\{0,1\}$ such that on any randomly chosen valid input β, t outputs 1 with probability $1/q^2$. Here q is parameter polynomial in the security parameter (to be determined later). We give an explicit construction for $\mathcal{T}_k$ based on q-wise independent hash functions. Let $h : \{0,1\}^{\leq \text{poly}(\kappa)} \rightarrow \{0,1\}^n$ be chosen from a family of q-wise independent hash functions H_k. Then, on any input β, $t(\beta) = 1$ iff the first $2\log(q)$ bits of $h(\beta)$ are equal to 0.

Now, recall that a black-box simulator $\mathcal{S}$ for any two-party computation protocol has 'oracle access' to the real world adversary $\mathcal{A}$. Formally, we consider $\mathcal{A}$ as a non-interactive algorithm that gets as input the history of the interaction, and outputs the next message that $\mathcal{A}$ would send in an execution of Π in which it sees this history. Further, $\mathcal{S}$ can query $\mathcal{A}$ with any sequence of messages of the form $(\beta_1, \ldots, \beta_i), i \leq c$ (i.e., the query contains the history of the interaction), and it will receive back the next message that $\mathcal{A}$ would send in any execution of Π in which it received this sequence of messages. For the sake of simplicity, we will assume without loss of generality, that the simulator $\mathcal{S}$ always follows the following two conventions:

1. It never asks the same query twice.
2. If $\mathcal{S}$ queries $\mathcal{A}$ with β, then it must have queried $\mathcal{A}$ with all the proper prefixes of β prior to this query.
3. $\mathcal{S}$ outputs a view $v = (\gamma_1^{\mathcal{S}}, \gamma_1^{\mathcal{A}}, \ldots, \gamma_c^{\mathcal{S}}, \gamma_c^{\mathcal{A}})$. Then, v represents the "main thread" of the interaction between $\mathcal{S}$ and $\mathcal{A}$.

We note that it is easy to modify any black-box simulator such that it follows the above conventions, without affecting its output distribution.

P_1^* strategy. We assume that P_1^* has the required input and random tape as would an honest party. We describe P_1^* as a non-interactive algorithm that gets as an input query the next message and the history of messages sent by P_2, and outputs the next message that P_1^* would send in a real execution of Π in which it sees this message history. Let x be the initial input and r be the random tape of P_1^*. Then, on receiving as input a series of messages $\beta = (\beta_1, \ldots, \beta_i)$, P_1^* performs the following steps:

1. Compute $t(\beta')$ for every prefix $\beta' = (\beta_1, \ldots, \beta_j)$ of β, where $j \in [1, i]$.
2. Send a message drawn from the uniform distribution unless for every j, $t(\beta_1, \ldots, \beta_j) = 1$. Note that this is a sanity check to ensure that P_1^* replies honestly to β only if it would not have stopped participating on receiving any query sent prior to β.
3. If the decision is to not send a message drawn from the uniform distribution, but instead to send an honest response, then run the code for the honest party P_1 on initial input x and random tape r, on input messages $(\beta_1, \ldots, \beta_i)$, and output its response.

Since the number of rounds in Π is a constant c, it follows that P_1^* participates in Π with noticeable probability $(= (1/q^2)^c)$. Let $\mathcal{S}$ be any black-box simulator for protocol Π. Now we use a hybrid argument to derive a lower bound on the probability p_{fail} defined as follows. Let $v = (\gamma_1^{\mathcal{S}}, \gamma_1^{\mathcal{A}}, \ldots, \gamma_c^{\mathcal{S}}, \gamma_c^{\mathcal{A}})$ be the view output by $\mathcal{S}$ at the end of its interaction with $\mathcal{A}$. Then, p_{fail} is the probability that $t(\gamma) = 1$ for every $\gamma = (\gamma_1^{\mathcal{S}}, \ldots, \gamma_i^{\mathcal{S}})$, $i \in [1, c]$, and $t(\gamma') = 0$ for *all* other queries γ' made by $\mathcal{S}$. We construct a series of hybrids $\mathcal{H}_i$, $i \in [0, 3]$, where each hybrid represents the interaction between an adversarial P_1 (referred to as $\mathcal{A}$) with a strategy we define and the simulator $\mathcal{S}$. The adversary $\mathcal{A}$ in $\mathcal{H}_3$ is identical to P_1^*. In each hybrid, we define and analyze the winning probability of $\mathcal{A}$ such that p_{fail} is the probability with which $\mathcal{A}$ wins in $\mathcal{H}_3$.

Let $r_{\mathcal{A}}$ be the random tape of $\mathcal{A}$. We now describe the hybrid experiments.

Hybrid $\mathcal{H}_0$. In this experiment, $\mathcal{A}$ simply sends a message drawn from the uniform distribution on every query from $\mathcal{S}$. Let q be the median of the number of queries that $\mathcal{S}$ makes. That is, with probability $1/2$ (where probability is taken over all the coins of $\mathcal{S}$ and $\mathcal{A}$), $\mathcal{S}$ makes at most q queries. Note that $q = \text{poly}(\kappa)$.

We now define a set Ψ as follows. Consider the tuple $(r_{\mathcal{S}}, r^1_{\mathcal{A}})$, where $r_{\mathcal{S}}$ is the random tape of $\mathcal{S}$ and $r^1_{\mathcal{A}}$ is the random tape that $\mathcal{A}$ uses to draw messages from the uniform distribution. Note that any such tuple defines an interaction between $\mathcal{A}$ and $\mathcal{S}$ in $\mathcal{H}_0$. Then, Ψ is the set of all such tuples $\mu = (r_{\mathcal{S}}, r^1_{\mathcal{A}})$ such that $\mathcal{S}$ makes at most q queries in the interaction defined by μ.

In addition to $r^1_{\mathcal{A}}$, $\mathcal{A}$ has a random tape $r^2_{\mathcal{A}}$ that is used to solely decide the winner of the experiment. More specifically, $\mathcal{A}$ uses the random tape $r^2_{\mathcal{A}}$ to choose a predicate t (as defined earlier). Then, we define a winning criterion for $\mathcal{A}$ in this experiment, as follows. Let $\{\gamma^{\mathcal{S}}_1, \gamma^{\mathcal{A}}_1, \ldots, \gamma^{\mathcal{S}}_c, \gamma^{\mathcal{A}}_c\}$ be the view output by $\mathcal{S}$ at the end of the experiment, where $\gamma^{\mathcal{S}}_i$ and $\gamma^{\mathcal{A}}_i$ are the messages in the ith round of Π from $\mathcal{S}$ and $\mathcal{A}$ respectively. Then, we say that $\mathcal{A}$ wins if $t(\gamma) = 1$ for every query $\gamma = (\gamma^{\mathcal{S}}_1, \ldots, \gamma^{\mathcal{S}}_i)$, $i \in [1, c]$, and $t(\gamma') = 0$ for *all* other queries γ' made by $\mathcal{S}$. In all other cases, we say that $\mathcal{S}$ wins.

We now analyze the winning probability of $\mathcal{A}$ over the choices of $r^2_{\mathcal{A}}$, i.e., for a fixed tuple $(r_{\mathcal{S}}, r^1_{\mathcal{A}})$. Let E_1 be the event that $t(\beta) = 1$ for every query $\beta = (\beta_1, \ldots, \beta_i)$ from $\mathcal{S}$, where $i \in [1, c]$ and $\{\beta_1, \gamma_1, \ldots, \beta_c, \gamma_c\}$ is the view output by $\mathcal{S}$ at the end of the experiment. Further, let E_2 be the event that $t(\beta') = 0$ for all other queries β' made by $\mathcal{S}$. We observe that E_1 and E_2 are independent events conditioned on $(r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi$ since in this case t essentially behaves as a random function (recall that t is constructed from a q-wise independent hash function) and the queries made by $\mathcal{S}$ (and its view in general) are independent of the choice of t. Then, we have:

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &\geq \Pr[\mathcal{A} \text{ wins} | (r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] \cdot \Pr[(r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] \\
&\geq \Pr[E_1 | (r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] \cdot \Pr[E_2 | (r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] \cdot \Pr[(r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] \\
&\geq (\frac{1}{q^2})^c \cdot (1 - \frac{1}{q^2} \cdot q) \cdot \frac{1}{2} \\
&= \frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q}),
\end{aligned}$$

where $\Pr[E_2 | (r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] < \Pr[t(\beta) = 0 \ \forall \beta | (r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi] = 1 - (\frac{1}{q^2}) \cdot q$ (β is any query from $\mathcal{S}$ during its interaction with $\mathcal{A}$). Note that $\Pr[\mathcal{A} \text{ wins}]$ is noticeable in κ. Further, this probability is same even for random choices of $(r_{\mathcal{S}}, r^1_{\mathcal{A}}, r^2_{\mathcal{A}})$ (such that $(r_{\mathcal{S}}, r^1_{\mathcal{A}}) \in \Psi$).

Hybrid $\mathcal{H}_1$. Same as $\mathcal{H}_0$ except the following. On receiving any query $(\beta_1, \ldots, \beta_{i-1}, \beta_i)$, $\mathcal{A}$ first checks whether $t(\beta) = 1$, for every $\beta = (\beta_1, \ldots, \beta_j)$, $j \in [1, i]$. If that is the case, then $\mathcal{A}$ further checks if it had earlier received another query $\beta' = (\beta_1, \ldots, \beta_{i-1}, \beta'_i)$, where $\beta'_i \neq \beta_i$ and $i \geq 1$, such that $t(\beta') = 1$. If all the above checks (in the sequel, we will refer to these combined checks as the *stopping condition*) succeed, then $\mathcal{A}$ stops the experiment and we say that $\mathcal{S}$ wins.

Otherwise, if any check fails, then $\mathcal{A}$ simply sends a message drawn from the uniform distribution (as in $\mathcal{H}_0$). In this case, let $\{\gamma^{\mathcal{S}}_1, \gamma^{\mathcal{A}}_1, \ldots, \gamma^{\mathcal{S}}_c, \gamma^{\mathcal{A}}_c\}$ be the view output by $\mathcal{S}$ at the end of the experiment, where $\gamma^{\mathcal{S}}_i$ and $\gamma^{\mathcal{A}}_i$ are the messages in the ith round of Π from $\mathcal{S}$ and $\mathcal{A}$ respectively. Then, we say that $\mathcal{A}$ wins if $t(\gamma) = 1$ for every query $\gamma = (\gamma^{\mathcal{S}}_1, \ldots, \gamma^{\mathcal{S}}_i)$, $i \in [1, c]$, and $t(\gamma') = 0$ for *all* other queries γ' made by $\mathcal{S}$. In all other cases, we say that $\mathcal{S}$ wins.

We now analyze the winning probability of $\mathcal{A}$ in this experiment. We first observe that the set of random tuples $(r_{\mathcal{S}}, r^1_{\mathcal{A}}, r^2_{\mathcal{A}})$ for which $\mathcal{A}$ wins are identical in $\mathcal{H}_0$ and $\mathcal{H}_1$. This is because the only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that for some random tuples $(r_{\mathcal{S}}, r^1_{\mathcal{A}}, r^2_{\mathcal{A}})$, $\mathcal{A}$ might stop the experiment in $\mathcal{H}_1$ (and hence lose); however, note that for all these tuples, $\mathcal{A}$ would have lost in $\mathcal{H}_0$ as well. Therefore, we conclude that the winning probability of $\mathcal{A}$ is identical in $\mathcal{H}_0$ and $\mathcal{H}_1$.

Hybrid $\mathcal{H}_2$. Same as $\mathcal{H}_1$, except that on receiving any query $(\beta_1, \ldots, \beta_i)$ from $\mathcal{S}$, if $t(\beta) = 1$ for every prefix $\beta = (\beta_1, \ldots, \beta_j)$, $j \in [1, i]$, but the *stopping condition* is false, then $\mathcal{A}$ sends an honest reply (as it would if it were honestly participating in Π). In order to do so, $\mathcal{A}$ internally runs P_1 on its initial input and

randomness, and the simulator's query β, and sends its output to $\mathcal{S}$. However, if the *stopping condition* is true, it continues to stop the experiment as in $\mathcal{H}_1$.

Let $\{\gamma_1^{\mathcal{S}}, \gamma_1^{\mathcal{A}}, \ldots, \gamma_c^{\mathcal{S}}, \gamma_c^{\mathcal{A}}\}$ be the view output by $\mathcal{S}$ at the end of the experiment, where $\gamma_i^{\mathcal{S}}$ and $\gamma_i^{\mathcal{A}}$ are the messages in the ith round of Π from $\mathcal{S}$ and $\mathcal{A}$ respectively. Then, we say that $\mathcal{A}$ wins if $t(\gamma) = 1$ for every query $\gamma = (\gamma_1^{\mathcal{S}}, \ldots, \gamma_i^{\mathcal{S}})$, $i \in [1, c]$, and $t(\gamma') = 0$ for *all* other queries γ' made by $\mathcal{S}$. In all other cases, we say that $\mathcal{S}$ wins.

We now try to bound from below $\mathcal{A}$'s winning probability in $\mathcal{H}_2$ by considering the following experiment. Consider a machine M that interacts with $\mathcal{S}$ by using an external party P_1 in the following manner. We assume that P_1 has some initial input while M does not. M begins the experiment by choosing a predicate t. Then, on receiving any query $(\beta_1, \ldots, \beta_i)$ from $\mathcal{S}$, if $t(\beta) = 1$ for every prefix $\beta = (\beta_1, \ldots, \beta_j)$, $j \in [1, i]$ but the *stopping condition* is false, then M forwards this query to P_1. When P_1 sends a response, M forwards it to $\mathcal{S}$. Otherwise, M simply sends a message drawn from the uniform distribution to $\mathcal{S}$. Note that the external party P_1 interacts in just execution of Π with M.

Further observe that M forwards at most c queries $(\gamma_1, \ldots, \gamma_c)$ to P_1 where each $\gamma_i = (\beta_1, \ldots, \beta_i)$ (i.e., γ_i contains γ_{i-1} as a prefix). Hence M either simply sends a message drawn from the uniform distribution or gets an answer from P_1 in response to any query from $\mathcal{S}$.

Now consider the following two cases:

1. All the replies of P_1 are drawn from the uniform distribution. In this case, the combination of P_1 and M is equivalent to $\mathcal{A}$ in $\mathcal{H}_1$.

2. P_1 sent an honest reply to each query. In this case, the combination of P_1 and M is equivalent to $\mathcal{A}$ in $\mathcal{H}_2$.

Let $\mathcal{D}$ be a polynomial-time machine that can distinguish between the above two cases with some probability ϵ. Then, the winning probability of $\mathcal{A}$ in $\mathcal{H}_2$ must be at least $\frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q}) - \epsilon$ (recall that the winning probability of $\mathcal{A}$ in $\mathcal{H}_1$ is at least $\frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q})$). Note that this probability is non-negligible if ϵ is negligible in the security parameter.

Hybrid $\mathcal{H}_3$. Same as $\mathcal{H}_2$, except that if the *stopping condition* is true, then $\mathcal{A}$ sends an honest reply (as it would if it were participating honestly), instead of stopping the experiment. Further, $\mathcal{A}$ continues to use the same (modified) strategy to answer further queries from $\mathcal{S}$. More specifically, $\mathcal{A}$ uses the following strategy in this experiment. On receiving any query $\beta = (\beta_1, \ldots, \beta_i)$ from $\mathcal{S}$, $\mathcal{A}$ computes $t(\beta')$ for every prefix $\beta' = (\beta_1, \ldots, \beta_j)$ of β, where $j \in [1, i]$. Then, if $t(\beta') = 1$ for all $j \in [1, i]$, $\mathcal{A}$ sends a message to $\mathcal{S}$ as if it were participating honestly in the protocol. Otherwise, it sends a message drawn from the uniform distribution. We observe that by definition, $\mathcal{A}$ in this experiment is identical to P_1^*.

Let $\{\gamma_1^{\mathcal{S}}, \gamma_1^{\mathcal{A}}, \ldots, \gamma_c^{\mathcal{S}}, \gamma_c^{\mathcal{A}}\}$ be the view output by $\mathcal{S}$ at the end of the experiment, where $\gamma_i^{\mathcal{S}}$ and $\gamma_i^{\mathcal{A}}$ are the messages in the ith round of Π from $\mathcal{S}$ and $\mathcal{A}$ respectively. Then, we say that $\mathcal{A}$ wins if $t(\gamma) = 1$ for every query $\gamma = (\gamma_1^{\mathcal{S}}, \ldots, \gamma_i^{\mathcal{S}})$, $i \in [1, c]$, and $t(\gamma') = 0$ for *all* other queries γ' made by $\mathcal{S}$. In all other cases, we say that $\mathcal{S}$ wins. We observe that the by definition, p_{fail} is identical to the winning probability of $\mathcal{A}$ $(= P_1^*)$ in $\mathcal{H}_3$.

We now analyze the winning probability of $\mathcal{A}$ in this experiment. We first observe that the set of random tuples $(r_{\mathcal{S}}, r_{\mathcal{A}})$ (where $r_{\mathcal{A}} = r_{\mathcal{A}}^1, r_{\mathcal{A}}^2, r_{\mathcal{A}}^3$) for which $\mathcal{A}$ wins are identical in $\mathcal{H}_2$ and $\mathcal{H}_3$. This is because the only difference between $\mathcal{H}_2$ and $\mathcal{H}_3$ is that for some random tuples $(r_{\mathcal{S}}, r_{\mathcal{A}})$, $\mathcal{A}$ might stop the experiment in $\mathcal{H}_2$ (and hence lose); however, note that for all these tuples, $\mathcal{A}$ will lose in $\mathcal{H}_3$ as well. Therefore, we conclude that the winning probability of $\mathcal{A}$ is identical in $\mathcal{H}_2$ and $\mathcal{H}_3$. Hence, we have that $\mathsf{p}_{fail} \geq \frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q}) - \epsilon$.

Further, we observe that with probability at least p_{fail}, the output of P_1^* in this experiment is the same as the output (say) z from the trusted party on inputs x and y, where x is the input of P_1^* and y is the input of P_2 in the ideal world. This is because (by the security of covert computation) P_1^* must obtain the same output z (as in the ideal world) if it participates honestly (with input x) in a real world execution of Π with P_2 (having intput y).

PART 2. Non-existence of a constant-round covert computation protocol that securely realizes $\hat{F}$ with respect to black-box simulators

Let us consider a covert two-party functionality $\hat{F} = (f, g)$, where f and g are defined as follows.

$$\begin{aligned} g(x,y) &= \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise} \end{cases} \\ f(x,y) &= (1,1) \end{aligned}$$

Consider two parties P_1,P_2 with inputs x and y respectively. Then, on an input pair (x, y), the functionality $\hat{F}$ outputs $f(x, y)$ if $g(x, y) = 1$, otherwise $\perp$.

Let $\hat{\Pi}$ be a constant-round covert two-party computation protocol that securely realizes $\hat{F}$. Let $\mathcal{S}$ be any black-box simulator for $\hat{\Pi}$. Recall the lower bound s from the previous part for such a simulator $\mathcal{S}$. We first show that for the case of functionality $\hat{F}$, s is in fact non-negligible. Intuitively, this means that $\mathcal{S}$ is *essentially straight-line* with non-negligible probability s. To see this, recall the definition of probability ϵ from experiment $\mathcal{H}_2$. Now consider an interaction between an honest P_1 (with input x) and an adversarial P_2 in protocol $\hat{\Pi}$. Then, ϵ is the probability with which P_2 can distinguish whether or not P_1 is participating in the real world protocol. However, note that in the ideal world, P_2 (that does not have the correct input $y = x$) can only distinguish with negligible probability whether or not P_1 is participating. Therefore, we conclude that ϵ is negligible (in the security parameter). It now follows that p_{fail} is noticeable.

We will now construct an adversary P_2^* that internally uses the simulator $\mathcal{S}$ to interact with honest P_1 (with input x) in an execution of $\hat{\Pi}$. We show that with non-negligible probability at least p_{fail}, P_2^* obtains a y such that $x = y$. However, observe that any cheating P_2^* cannot obtain such a y in an ideal world execution; therefore, the output distributions of the real world and ideal world executions must be distinguishable with probability at least p_{fail}.

P_2^* strategy. We are now ready to construct the adversary P_2^*. Loosely speaking, with noticeable probability, P_2^* will be able to obtain a y, such that $x = y$, from its interaction with an honest P_1 (with input x). P_2^* will internally run the simulator $\mathcal{S}$ and forward some of its queries to P_1. As we will show later, with noticeable probability, $\mathcal{S}$ will work in exactly the same way as when it is given oracle access to P_1^* (where P_1^* is an adversary for $\hat{\Pi}$ that follows the same strategy as described earlier in part 1). We now give a formal description of P_2^*.

P_2^* chooses a predicate t (as defined earlier) and runs $\mathcal{S}$. At any point of time, P_2^* keeps the history of messages that it has sent to P_1 so far in the execution. Now, when $\mathcal{S}$ makes a query $\beta = (\beta_1, \ldots, \beta_i)$, P_2^* performs the following steps:

1. Follow the same strategy as P_1^* in order to decide whether or not to send an honest answer (i.e., send a message drawn from the uniform distribution unless for every $j \in [1, i]$, $t(\beta_1, \ldots, \beta_j) = 1$).

2. If the above decision was to send an honest reply, then check whether the history of messages sent so far to P_1 consists exactly of $(\beta_1, \ldots, \beta_{i-1})$. If this is the case, then send β_i to P_1 and forward P_1's response to $\mathcal{S}$. Otherwise, there must have been a previous query that is not a prefix of β but was answered with a message drawn from the uniform distribution; abort in this case.

Finally, if $\mathcal{S}$ sends a query x' to the trusted party in the ideal world, then P_2^* receives x' (since P_2^* plays the role of the trusted party to $\mathcal{S}$) and outputs this value. All we need to show now is that with probability at least p_{fail}, $x' = x$, where x is the input of P_1.

We first observe that with probability at least p_{fail}, the interaction between P_2^* (using P_1) and $\mathcal{S}$ is identical to that between P_1^* and $\mathcal{S}$. Then, with probability at least p_{fail}, $\mathcal{S}$ outputs a view such that all the simulator queries in the view were answered by P_1. Now observe that the input of P_1^* emulated by P_2^* (using P_1) is in fact the input of P_1, i.e., x. Further, with probability at least p_{fail}, the view generated by $\mathcal{S}$ contains full participation from P_1^* with input x. Then, as explained earlier in $\mathcal{H}_3$, with probability at least p_{fail}, the output of the emulated P_1^* must be the same as the output of the trusted party on inputs x and

y, where x is the input of P_1^* and y is the input of P_2 in the ideal world. In other words, the output of P_1^* must be 1 (since $y = x$ for honest parties P_1^* and P_2). Then it follows that with probability at least p_{fail}, the query from $\mathcal{S}$ to the trusted party must be $x' = x$.

4 Covert Computation primitives

In this section, we briefly describe the main primitives that we use in our construction. The text in this section is taken almost verbatim from [CGOS07] (except the part about the covert encryption scheme in NC^0; see below). We refer the reader to appendix B for a detailed description of each primitive.

Covert Commitments [GL89] : In a covert commitment scheme, the messages of a party in the commitment phase look indistinguishable from random to the other party. The commitment scheme based on the Goldreich-Levin hard-core predicate is a covert bit commitment scheme that is perfectly binding. Let $x = x[1], \ldots, x[k]$ be any string. $Com(x)$ denotes a covert commitment to the bits $x[1], \ldots, x[k]$. Further, $Open(Com(x))$ denotes the decommitment to $Com(x)$ (which includes x along with the randomness used in computing $Com(x)$).

Covert 1-out-of-2 and 1-out-of-4 Oblivious Transfer [vAHL05, CGOS07, NP01] : In covert 1-out-of-2 and 1-out-of-4 oblivious transfer (OT) protocols, we require that messages of a party throughout the protocol look indistinguishable from random to the other party. Covert 1-out-of-2 OT is presented in [vAHL05], while a covert 1-out-of-4 OT is given in [CGOS07].

Covert Yao's Garbled Circuit [vAHL05, Yao82] : This is a version of garbled circuit [Yao82] in which the garbled circuit *program* (which is simply a series of encrypted gate tables [Yao82, BMR90]) can be evaluated in a "non-verifiable" manner.

Covert GMW protocol [CGOS07, GMW87] : This is the same as the standard (semi-honest) GMW protocol [GMW87], except that (a) *covert* 1-out-of-4 OT is used, and (b) the final output share broadcast phase is excluded. That is, in covert GMW protocol, the parties initially finally hold the *shares of the output* (rather than the output itself). Note that zero-knowledge proofs are not used in this protocol (and thus the protocol by itself does not provide any guarantees against malicious adversaries).

Covert Encryption scheme in NC^0 [AIK04] : In a covert encryption scheme, we require that the ciphertext be indistinguishable from random. Given a covert encryption scheme in NC^1 (which can be constructed using standard techniques), we use techniques from Applebaum et al [AIK04] to construct a covert encryption scheme whose encryption function is in NC^0.

5 Constant Round Covert Multiparty Computation

At a high level, our constant-round covert computation protocol can be seen as the result of a two step process: (a) First, construct a *constant*-round semi-honest covert computation protocol adopting techniques from the work of Beaver et al [BMR90]. (b) Next, the semi-honest protocol is "compiled" with a gadget known as *zero knowledge proofs to garbled circuits* in order to guarantee security against malicious adversaries. Here we adopt some techniques from Chandran et al [CGOS07] to our setting. In the subsection below, we first discuss the notion of zero-knowledge proof to garbled circuit as introduced by Chandran et al [CGOS07], and then give a constant-round construction for the same with some additional security properties (that are necessary when using this gadget in the constant-round setting). Later, we will use our construction of constant-round zero knowledge proof to garbled circuit in presenting our constant-round covert computation protocol.

5.1 Zero Knowledge Proofs to Garbled Circuits

Zero Knowledge proofs have been established as a basic building block for constructing multi-party computation protocols secure against active adversaries. However, in the setting of covert computation, this technique does *not* work because if one party "verifies" that another party is executing the protocol honestly, then covertness is immediately compromised. To this end, Chandran et al [CGOS07] introduced the notion of zero knowledge proofs to garbled circuits, where a party gives a proof of its honest behavior to a *garbled circuit* prepared by another party. More specifically, consider two parties (sender, receiver) who share a common input (x, L). The sender wishes to give the receiver a private value v, only if $x \in L$ and the receiver has a valid witness (for $x \in L$). Chandran et al [CGOS07] gave a protocol for this setting based on Blum's 3-round (public-coin) ZK proof for Graph Hamiltonicity. In their protocol, the parties first exchange the first two messages of Blum's protocol. Then the sender (verifier) prepares and sends a garbled circuit to the receiver (prover); this garbled circuit takes as input the last prover message and outputs v if the verification is successful, else it outputs a random value. Since Blum's protocol is a zero knowledge *proof* with soundness $1/2$, if the theorem is false, there does not exist (with probability $1/2$) a "correct" last prover message. [CGOS07] also show how to improve the soundness of this basic protocol.

As implied by the results in the previous section, non black-box techniques are necessary to construct such a gadget (henceforth referred to as ZKSend) in the constant-round setting. To this end, we use the non black-box simulation technique of Pass [Pas04] (which in turn builds on the work of Barak [Bar01]). Fortunately, in the zero knowledge protocol of Pass [Pas04], except for the last message, the prover only sends commitments and the verifier only sends random strings. Then, using the same idea as above, we can modify Pass' protocol such that the receiver (prover) sends the last message to a garbled circuit prepared by the sender (verifier), and receives an output value depending upon whether or not the verification was successful.

However, this naive attempt fails since Pass' protocol is an *argument* system. In particular, even if the receiver (prover) was dishonest, a satisfying last message might exist which, very informally speaking, allows the dishonest receiver (prover) to get the sender's input value out of the garbled circuit (even though the receiver does not have such a message explicitly). Then, to be able to reduce the security of ZKSend in such a case to the soundness of Pass' protocol, it seems that the garbled circuit evaluation sub-protocol would need to be able to support extraction of the inputs. However, as implied by the impossibility result in the previous section, very informally speaking, such a sub-protocol cannot work in a constant number of rounds (using a black-box extractor).

To solve the above problem, we observe that such an extraction of the input will not be required by the simulator of our final covert computation protocol constructed using ZKSend as a building block. Instead, such an extraction would only be required to prove a separate lemma that reduces the following security property of ZKSend to the soundness of Pass' protocol: assuming that the statement is false, the view of a cheating receiver (prover) must be indistinguishable across the two cases where an honest sender (verifier) uses a fixed input value in the ZKSend execution in the first case and a random value as its input in the ZKSend execution in the second case. (Looking ahead, such a lemma would be used in the hybrid experiments to prove the indistinguishability of the simulated view from the view in the real protocol execution.) Hence, the extraction procedure is only required to work with a noticeable probability (as opposed to overwhelming probability). This is because of the following. Say we can extract the input to the garbled circuit (which is the last prover message in Pass' protocol) with a noticeable probability. Then we can use that message to violate the soundness of the protocol of Pass with a noticeable probability.

Using these ideas, we now describe our protocol. We first introduce some notation.

Notation. Let cZK denote the zero knowledge argument of Pass instantiated with the covert commitment scheme *Com* (i.e., *Com* is the commitment scheme used throughout the protocol in cZK). A sub-protocol of cZK is the 5-round witness indistinguishable universal argument (WI-UARG) of Barak and Goldreich [BG02] instantiated with the commitment scheme *Com*. We will denote it with cWI-UARG. Let V denote the honest verifier algorithm for cZK. In the following text, we borrow some notations from [Pas04].

We first recall Barak's **NTIME**$(T(\kappa))$ relation R_{Sim} [Bar01]. Let $T : N \to N$ be a "nice" function that satisfies $T(\kappa) = \kappa^{\omega(1)}$. Let $T' : N \to N$ be a function such that $T'(\kappa) = T(\kappa)^{\omega(1)}$. Suppose that there exist a $T'(\kappa)$-collision resistant hash functions ensemble $\{\mathcal{H}_\kappa\}_{h\in\{0,1\}^\kappa}$ where h maps $\{0,1\}^*$ to $\{0,1\}^\kappa$. Let the triple $\langle h, c, r\rangle$ be the input to R_{Sim}. Further, consider a witness that consists of a program Π, a string $y \in \{0,1\}^{(|r|-\kappa)}$, and string s. Then $R_{Sim}(\langle h, c, r\rangle, \langle \Pi, s, y\rangle) = 1$ if and only if:

1. $c = Com(h(\Pi); s)$.
2. $\Pi(c, y) = r$ within $T(k)$ steps.

Here *Com* is the covert commitment scheme (see appendix B).

Now let P_i and P_j be two parties that share as common input a statement x_j of a language L_j. Additionally, P_i has a private input v. Let $\ell(\kappa)$ be a length parameter (we discuss how to fix it later). We now describe our protocol $ZKSend_{ij}$, where P_i and P_j play the roles of the *sender* and *receiver* respectively (we will follow this convention throughout the text). If P_j has a witness for $x_j \in L_j$, then it will obtain P_i's private input v at the end of the protocol.

5.1.1 The Protocol.

The protocol $ZKSend_{ij}$ proceeds in the following steps.

Phase I.

Stage 1 **(Setup)** : P_i sends $h \xleftarrow{R} \mathcal{H}_\kappa$ to P_j.

Stage 2 **(Slot 1)** :

1. P_j creates a covert commitment $c_1 = Com(0^\kappa)$ and sends it to P_i.
2. P_i responds by sending the first challenge $r_1 \xleftarrow{R} \{0,1\}^{j\ell(\kappa)}$.

Stage 3 **(Slot 2)** :

1. P_j creates a covert commitment $c_2 = Com(0^\kappa)$ and sends it to P_i.
2. P_i responds by sending the second challenge $r_2 \xleftarrow{R} \{0,1\}^{(n+1-j)\ell(\kappa)}$.

Stage 4 **(Main Proof Body)** :

1. P_i and P_j exchange (only) the first 4 messages of a 5-round cWI-UARG where P_j proves the OR of the two statements:
   - There exists $w \in \{0,1\}^{\text{poly}(|x|)}$ such that $R_{L_j}(x_j, w) = 1$.
   - There exists a triple $\langle \Pi, s, b\rangle$ such that $R_{Sim}(h, c_b, r_b, \langle \Pi, s\rangle) = 1$.

   Let m_{final} denote the final message of this cWI-UARG.
2. Let $k = \omega(log(\kappa))$. P_j chooses random strings $\{\alpha_i^0\}_{i=1}^k$, $\{\alpha_i^1\}_{i=1}^k$ such that $\alpha_i^0 \oplus \alpha_i^1 = m_{final}$ for all i. Using the covert commitment scheme *Com*, P_j creates commitments to all these random shares of m_{final} and sends them to P_i. Let $\{A_i^0\}_{i=1}^k$, $\{A_i^1\}_{i=1}^k$ denote these commitments.

Stage 5 **(Challenge-Response)** : Now, P_i and P_j engage in a 2-round challenge-response protocol:

1. $P_i \to P_j$: Send challenge bits $z_1, \ldots, z_k$.
2. $P_j \to P_i$: Send $\alpha_1^{z_1}, \ldots, \alpha_k^{z_k}$. /* *$P_j$ does not send openings to the commitments, but simply the shares selected by P_i* */

Phase II.

Stage 6 **(Garbled Circuit)** : Finally, P_i sends a covert garbled circuit, $Gar[i \rightarrow j]$, with the following description:

1. $Gar[i \rightarrow j]$ has (built in) the transcript T of stage $4(b)$ and stage 5 in phase I.
2. It takes as input, decommitments to $\{A_i^0\}_{i=1}^k$, $\{A_i^1\}_i^k$. Let $\{\hat{\alpha}_i^0\}_{i=1}^k$, $\{\hat{\alpha}_i^1\}_{i=1}^k$ denote the decommitted values. This input is provided by P_j.
3. $Gar[i \rightarrow j]$ performs the following steps:
   - Check whether the decommitments are *correct* and that the decommitted values are *consistent* (i.e., $\hat{\alpha}_i^0 \oplus \hat{\alpha}_i^1$ is identical for all $i \in [1,k]$). Let $\hat{m}_{final} = \hat{\alpha}_i^0 \oplus \hat{\alpha}_i^1$.
   - Check whether $\hat{\alpha}_i^{z_i} = \alpha_i^{z_i}$ for all $i \in [1,k]$, where $\{\alpha_i^{z_i}\}_{i=1}^k$ are the values sent by P_j in stage 5(b) in transcript T.
   - If either of the above checks fail, then simply output a uniformly chosen random number. Otherwise, run the final step of the honest verifier algorithm for cZK and if it outputs accept on $\hat{m}_{final}$, then output v, else output a uniformly chosen random number.
4. P_j executes covert 1-out-of-2 OT with P_i as necessary in order to evaluate the garbled circuit $Gar[i \rightarrow j]$ and obtain an output.

Excluding the challenges r_1 and r_2, we note that the length of the sender messages (including the garbled circuit) and the receiver messages in $ZKSend_{ij}$ can be bounded by $2\kappa^3$ if we use the specific WI-UARG of Barak and Goldreich [BG02]. Let Σ denote our covert multi-party computation protocol (see section 5) and let γ denote the total number of executions of $ZKSend_{ij}$ inside Σ (looking ahead, we note that $\gamma = 2n(n-1)$ where n is the number of parties in the protocol). Then, the total length of messages sent by a party, not including the challenges r_1,r_2, is bounded by $(\gamma \cdot 2\kappa^3 + \mathsf{length}(\Sigma))$. Here $\mathsf{length}(\Sigma)$ is the total length of the messages in Σ excluding the messages of all ZKSend executions. We now set the length parameter $\ell(\kappa) = (\gamma \cdot 4\kappa^3 + \mathsf{length}(\Sigma))$. Then, as in [Pas04], it follows that if the simulator can give a description of the various challenges r_1, r_2 (each being of length at least $\ell(\kappa)$) that is shorter than k^3, it will always succeed in the simulation of all executions of $ZKSend_{ij}$ even if simultaneously acting as the sender in some of these protocols. As in [Pas04], this is done by letting the simulator use a pseudorandom generator in order to generate the sender's messages in $ZKSend_{ij}$ (in particular, the challenge strings r_1 and r_2).

5.1.2 Security Properties of ZKSend

We prove the following four security properties of our ZKSend construct. The formal claims and their respective proofs are given in appendix C.1.

1. Consider a ZKSend execution between a sender (verifier) and a receiver (prover) who share a common input (x, L). The sender's input in the protocol is either a fixed value v or a random value. Then, we show that unless the receiver (prover) "knows a witness for $x \in L$", it cannot distinguish between the two cases where the sender (verifier) is using v in the first case and a random value as its input in the second case. As noted earlier, we prove this security property by reducing it to the soundness of Pass' protocol.

2. Next, we show that an adversarial sender (verifier) who is running a polynomially-bounded number of concurrent executions of ZKSend cannot distinguish whether it is "interacting" with honest receivers (provers) or the simulator. Our simulator relies on the simulator of Pass' protocol to "simulate" the ZKSend executions; as such, we prove this security property by reducing it to the zero knowledge property of Pass' protocol. Looking ahead, we stress that even though our covert computation protocol (given in next subsection) consists of only *parallel* executions of ZKSend, we consider the more general setting of *bounded-concurrency* as it proves to be useful in the construction of a covert computation protocol over *point-to-point* channels (see section 6).

3. Now consider a polynomially-bounded number of concurrent executions of ZKSend between an adversary and the simulator, where the adversary is acting as the sender (verifier) in some "left" executions (which are being "simulated" by the simulator) and as the receiver (prover) in a "right" execution with a false theorem as the common input. Then, we show that any such adversary cannot distinguish whether the simulator uses a fixed input value v or a random value as its input in the "right" execution. We stress that we cannot reduce this security property into the simulation soundness of Pass' protocol for the following reasons. Such a proof would require us to construct an adversary who proves a false theorem in Pass' protocol, which in turn, requires rewinding our adversary in the "right" execution (to extract the last message of Pass' protocol). However in this case, very informally speaking, the simulator who is "simulating" the "left" executions, may also get rewound and hence no longer work. Solving this problem requires going into the details of the simulation soundness proof technique of Pass to prove this security property *directly*. In particular, we show that the two slot simulation technique of Pass is powerful enough to handle this scenario as well.

4. Finally, we show that our ZKSend protocol preserves the covertness of both the sender and the receiver. In particular, we show that messages of a sender (resp. receiver) are indistinguishable from random to a receiver (resp. sender).

5.2 Our Protocol

Let $P_1, \ldots, P_n$ be n parties that hold inputs $x_1, \ldots, x_n$ respectively. Let $\mathcal{F} = (f, g)$ be a covert functionality that they wish to compute. $\mathcal{F}$ outputs $f(x_1, \ldots, x_n)$ if the function output is favorable to all parties, else it outputs $\perp$. Here $g(\cdot)$ is the function that determines whether the function output is favorable ($g(x_1, \ldots, x_n) = 1$) or not ($g(x_1, \ldots, x_n) = 0$). We now give the construction of a constant-round covert multi-party computation protocol Π that securely realizes $\mathcal{F}$.

Overview. At a high level, our protocol Π consists of three main phases: (a) *Input Commitment phase*: In this phase, all parties commit to their inputs and randomness (note that we do not require coin flipping in our protocol) and run an "extract enable" phase with each other. Intuitively, the purpose of this phase is to enable the simulator to extract the input and randomness of malicious parties during the simulation. (b) *Garbled Circuit Generation phase*: In this phase, the parties run the covert version of the GMW protocol to jointly construct a covert garbled circuit that evaluates the appropriate function (that the parties wish to compute). Each party only obtains an individual share of the garbled circuit as the output of covert-GMW protocol. (c) *Output Exchange phase*: In this phase, parties exchange their garbled circuit shares with each other if and only if all the parties behaved honestly till the end of the previous phase. By incorporating some validity checks in this phase, we are able to ensure output correctness (without compromising covertness).

Intuitively, the garbled circuit generation phase can be viewed as a semi-honest covert computation protocol. Here we adopt techniques from [BMR90] to ensure that the protocol is constant-round. Then, by adding the other two phases (adopting techniques from Chandran et al [CGOS07]) , we essentially "compile" the semi-honest protocol (with zero-knowledge proofs to garbled circuits, as will be evident from the description of each phase) to obtain security against active adversaries. In each phase, the key challenge is to ensure that the number of rounds are only a constant.

We now give the detailed description of our protocol Π. Throughout the protocol description, the intuition and key ideas in each phase in the protocol are enclosed within the symbols /* and */. We refer the reader to appendix D.2 for a succinct description of the protocol with an emphasis on only the key ideas.

5.2.1 Input Commitment phase

1. *InputRandomCommit :* Let $X_i = (x_i, r_i, K_i)$, where x_i is the input of party P_i and r_i is a random tape that P_i will contribute to the garbled circuit (which may require randomness to compute its output in case the functionality is randomized) that is jointly constructed by the parties later in the protocol. K_i is a random secret key chosen by P_i. This key will be used by P_i to check the correctness of the

function output in the later stages. P_i additionally generates random tapes s_i, t_i and u_i. Here s_i is the randomness that P_i will use in subprotocol *ExtractEnable*, t_i is the randomness that P_i will use in the Garbled Circuit Generation phase, and u_i is the randomness that P_i will use in the Output Exchange phase.

P_i now commits to X_i, s_i, t_i and u_i using the commitment scheme Com.

2. *ExtractEnable :* In this stage, each pair of parties engage in an execution of the ZKSend protocol. Intuitively, the purpose of this stage is to enable the simulator to obtain (X_j, t_j) for each malicious party P_j during the simulation.

   Consider a party P_i. To execute this stage, P_i uses the random tape s_i wherever required. Let $IR_i = (Open(Com(X_i)), Open(Com(t_i)))$. Then P_i executes the following sub-protocol with each party P_j, in *parallel*.

   (a) P_i picks a random string r and computes $y = f_{owp}(r)$, where f_{owp} is a one-way permutation. It sends y to P_j.

   (b) P_j and P_i now engage in an execution of $ZKSend_{ij}$, where P_i is the sender with input IR_i and P_j is the receiver, on the following NP statement as the common input: "$\exists r$ such that $y = f_{owp}(r)$" (for a specific witness relation such that any witness for this NP statement contains such an r). Note that an honest P_j will not have a witness for this NP statement, therefore it simply sends random messages during the execution of $ZKSend_{ij}$.

   We stress that all the executions of the above sub-protocol are carried out in *parallel*.

   /* As noted earlier, the *ExtractEnable* protocol will help the simulator $\mathcal{S}$ to obtain the input X_i and randomness u_i for each malicious party P_i during the protocol simulation. Let z be the NP statement that there exists an r such that $y = f_{owp}(r)$, where y is the random string that P_i sent to an honest party P_j in the first step. Then $\mathcal{S}$ will simulate the zero knowledge argument of knowledge for the NP statement z inside $ZKSend_{ij}$ in order to obtain IR_i (and therefore, X_i and t_i as well). However, note that $\mathcal{S}$ succeeds in extracting the inputs and randomness of the malicious parties *if none of them deviate* from the protocol. Of course, the malicious parties could deviate from the protocol specification and either give incorrect commitments or incorrect openings. In such a case (i.e., whenever the simulator fails to extract), later stages of our protocol will ensure that malicious parties do not learn any information (in a computational sense). Jumping ahead, the later stages will force the output of *all* the parties to be $\perp$ in this case.

   We remark that Chandran et al [CGOS07] used a challenge-response phase of linear round-complexity to enable black-box extraction. However, our impossibility result from the previous section rules out such an approach in the constant-round setting. To this end, we use our construction of zero-knowledge proof to garbled circuit (as described in the last subsection) to enable extraction, as described above.*/

5.2.2 Garbled Circuit Generation phase

Consider the following function $F(\cdot)$ that takes as input $\{X_1, \ldots, X_n\}$ where $X_i = (x_i, r_i, K_i)$).

$$F(X_1, \ldots, X_n) = \begin{cases} f(x_1, \ldots, x_n), K_1, K_2, ..., K_n & \text{if } g(x_1, \ldots, x_n) = 1 \\ R & \text{otherwise} \end{cases}$$

Here R is a random string of appropriate length. If necessary, the randomness used by this function is $\bigoplus_{i=1}^{n} r_i$. Let C denote the circuit for the function F.

In this phase, the parties jointly construct a covert garbled circuit GC for the circuit C. As a sub-protocol in this phase, the parties execute the covert-GMW protocol (see Section 4) in order to build the *gate tables* for the garbled circuit GC. The randomness used by P_i in this phase comes from t_i. At the end

of this phase, each party P_i will hold a share GC_i of the garbled circuit GC (such that $\bigoplus_{i=1}^{n} GC_i = GC$) and other necessary information required to evaluate GC (but not GC itself).

/* We note that here we use essentially the same approach as Beaver et al [BMR90] to ensure that our protocol is constant round. There are however some important differences in our approach from that of Beaver et al [BMR90], described as follows. We first note that in order to keep the protocol constant round, the encryption scheme used in [BMR90] to generate the gate tables of the garbled circuit is the simple XOR function. Further, to avoid blowing up the size of the wire keys exponentially, the parties run a preprocessing phase locally in which (among other things) they expand their wire keys using a pseudorandom generator (PRG). Unfortunately, such a preprocessing phase fails in the setting of covert computation. This is because the PRG evaluations done locally by a party during garbled circuit generation will have to be performed again (locally) while evaluating the resulting garbled circuit. Thus, the fact that the computation done in the preprocessing phase "conforms" to the computation done while evaluating the received garbled circuit leaks the fact that all parties are participating in the protocol (even if the output is not favorable).

To solve this problem, we eliminate the preprocessing phase and move the required cryptographic operations (involved in generating the gate tables) into the circuit of the underlying secure computation protocol. However this presents the following problem. If we use covert-GMW as the underlying secure computation protocol, the number of rounds required will be linear in the depth of the circuit (which generates a gate table) being evaluated. Thus, any cryptographic operations done by this circuit should be in NC^0 for our protocol to be constant rounds.

Towards that end, to still keep the secure computation protocol constant round, we construct an encryption scheme in NC^0 using techniques from Applebaum et al [AIK04]. We would require that the encryption keys and the ciphertexts produced by such an encryption scheme are indistinguishable from the uniform distribution. We construct such an encryption scheme based on standard assumptions.[4][5] Forgetting our goal of obtaining a covert computation protocol, consider the protocol in which we use such an encryption scheme in gate tables and use standard GMW for gate table generation. We believe this protocol is of independent interest since it also gives a arguably cleaner alternative to the Beaver et al [BMR90] protocol (which in turn has been used widely in the study of round complexity of secure computation).*/

We now describe this phase in detail. First recall that in a garbled circuit GC, each wire w has two *wire keys* (denoted by $k^{w,0}$ and $k^{w,1}$): one corresponding to the bit on wire w being 0 and the other to bit being 1 (during the actual evaluation of the garbled circuit, a party would only be able to find one of these keys for every wire). Further, there is *wire mask* λ^w associated with each wire w. The wire mask determines the correspondence between the two wire keys and the bit value associated with them. For example, the key $k^{w,b}$ corresponds to the bit $b \oplus \lambda^w$.

Now note that every player P_i is responsible for a subset J_i of the input wires in the circuit C. In particular, P_i holds an input bit x^w for each $w \in J_i$. Then the Garbled Circuit Generation phase consists of the following steps.

1. Each party P_i generates a wire mask share $\lambda_i^w \in \{0,1\}$ for every wire w of the circuit C. Note that $\lambda^w = \bigoplus_{i=1}^{n} \lambda_i^w$ is the wire mask for wire w. Further, for every wire w of the circuit, P_i generates two random key shares $k_i^{w,0}$ and $k_i^{w,1}$. The actual wire keys $k^{w,0}$ and $k^{w,1}$ for wire w are defined as: $k^{w,0} = \bigoplus_{i=1}^{n} k_i^{w,0}$ and $k^{w,1} = \bigoplus_{i=1}^{n} k_i^{w,1}$.

2. P_i broadcasts the wire mask shares λ_i^w for all wires w, where w is either an output wire in C, or w is an input wire that belongs to any party other than itself (i.e., for $w \notin J_i$).

[4] Applebaum et al [AIK04] show that there do not exist any encryption schemes such that the decryption function is in NC^0. However, fortunately, for our purpose, we only require the encryption function to be in NC^0.

[5] The option of simply moving the pre-processing phase of [BMR90] to the secure computation protocol circuit would require that circuit to evaluate a PRG. We however, note that a PRG with appropriate stretch exists in NC^0 based only on non-standard assumptions [AIK08].

3. For each input wire $w \in J_i$, P_i computes $\lambda^w = \bigoplus_{i=1}^n \lambda_i^w$ (note that P_i would have received the wire mask shares for $w \in J_i$ by now). Then, for every $w \in J_i$, P_i broadcasts the bit $b^w = \lambda^w \oplus x^w$ and the key share k_i^{w,b^w}. Once all the key shares are broadcast, P_i computes the key $k^{w,b^w} = \bigoplus_{i=1}^n k_i^{w,b^w}$ for each input wire w.

   At this point, all the parties hold the appropriate wire keys for all the input wires and the wire masks for all the output wires in the circuit.

4. Consider a gate g in the circuit C. In order to construct a *gate table* for g, the wire keys of the outgoing wire are encrypted with the wire keys of the incoming wires. Now, if all the parties use their wire mask shares and key shares as inputs in a covert GMW protocol execution, then they can jointly construct all the gate tables. We will use a covert encryption scheme in NC^0 to generate the gate tables. Note that we would require that the encryption keys and the ciphertexts produced by such an encryption scheme are indistinguishable from random. We construct such an encryption scheme based on standard assumptions using techniques from Applebaum et al [AIK04]. See appendix B.5 for the construction details. Let (G, E, D) be such an encryption scheme.

   The players now engage in an execution of the covert-GMW protocol to evaluate a simple circuit that generates the gate tables for circuit C. More details are given as follows.

   For any gate g in the circuit C, let a, b be the two input wires and c be the output wire for the gate g, and denote the operation performed by the gate g by $\otimes$ (e.g. AND, OR, NAND, etc). Before the protocol starts, P_i holds the following inputs: key shares $k_i^{a,\ell}, k_i^{b,\ell}, k_i^{c,\ell}$ where $\ell \in \{0,1\}$, along with the wire mask shares $\lambda_i^a, \lambda_i^b, \lambda_i^c$. Let $E_k(m)$ denote the encryption of a message m with key k using the covert encryption scheme (G, E, D). Then P_i runs the covert GMW protocol along with the other parties to compute a gate table of the following form (for each gate g in C):

$$
\begin{aligned}
A_g &= \begin{cases} E_{k^{a,0}}(E_{k^{b,0}}(k^{c,0}, 0)) & \text{if } \lambda^a \otimes \lambda^b = \lambda^c \\ E_{k^{a,0}}(E_{k^{b,0}}(k^{c,1}, 1)) & \text{otherwise} \end{cases} \\
B_g &= \begin{cases} E_{k^{a,0}}(E_{k^{b,1}}(k^{c,0}, 0)) & \text{if } \lambda^a \otimes \bar{\lambda^b} = \lambda^c \\ E_{k^{a,0}}(E_{k^{b,1}}(k^{c,1}, 1)) & \text{otherwise} \end{cases} \\
C_g &= \begin{cases} E_{k^{a,1}}(E_{k^{b,0}}(k^{c,0}, 0)) & \text{if } \bar{\lambda^a} \otimes \lambda^b = \lambda^c \\ E_{k^{a,1}}(E_{k^{b,0}}(k^{c,1}, 1)) & \text{otherwise} \end{cases} \\
D_g &= \begin{cases} E_{k^{a,1}}(E_{k^{b,1}}(k^{c,0}, 0)) & \text{if } \bar{\lambda^a} \otimes \bar{\lambda^b} = \lambda^c \\ E_{k^{a,1}}(E_{k^{b,1}}(k^{c,1}, 1)) & \text{otherwise} \end{cases}
\end{aligned}
$$

   Note that since all the gate tables can be computed in *parallel*, the covert GMW protocol execution only takes a constant number of rounds.

   When the covert-GMW protocol terminates, each party P_i obtains a share of each gate table as generated above. We define GC_i to be P_i's share of the garbled circuit GC, where GC_i is simply the concatenation of all the gate table shares of P_i.

5. Now, each party P_i broadcasts a covert commitment to its garbled circuit share GC_i. Let $Com(GC_i)$ denote the commitment value sent by P_i.

At the end of the Garbled Circuit Generation phase, each party P_i holds a share GC_i of the garbled circuit GC, along with the wire masks λ^w for each output wire w and the appropriate wire keys k^{w,b^w} and bit b^w (where $b^w = \lambda^w \oplus x^w$) for each input wire. In addition, P_i also holds a covert commitment to the garbled circuit share GC_j of each party j.

/* In this phase, we use the covert-GMW protocol of [CGOS07]. The covert-GMW protocol is similar to the semi-honest GMW protocol, except that it uses a specific covert secure 1-out-of-4 OT rather than a semi-honest 1-out-of-4 OT, and does not consist of the output broadcast phase. The natural question is, what guarantees could it possibly provide when the parties might deviate from the protocol executions arbitrarily?

Intuitively, the malicious parties (even if they deviate from the protocol arbitrary) do not learn any information (in a computational sense) in covert GMW protocol because of the following. The covert-GMW protocol consists of only two kinds of message: one where parties broadcast $(n-1)$ random shares of their input to other parties, and the second where parties engage in a covert 1-out-of-4 OT protocol with others. The first type of message is simply a random string, therefore it does not reveal any information about GC. In the second type of messages, a party gives away only one of the 4 bits (when acting as a sender in a covert 1-out-of-4 OT protocol). However, all the four bits are *individually* indistinguishable from random. This is because while preparing these four bits, a single bit of randomness is used [GMW87] (making the four bits *individually* random). */

5.2.3 Output Exchange phase

In the previous two stages, no information about the output (or participation) was revealed to any party (even if they deviate arbitrarily from the protocol). In other words, the messages exchanged between the parties in the previous stages were "not valuable" for deducing any potentially unknown information. In this phase, the parties actually exchange valuable messages having information so as to be able to get the function output at the end.

Informally, in this phase, the parties exchange their shares (held at the end of Garbled Circuit Generation phase) of garbled circuit *conditioned* upon the fact that *every party* was honest till the end of the Garbled Circuit Generation phase. This phase heavily relies on the various properties of our zero-knowledge to garbled circuit construction.

The randomness that a party P_i uses in this phase comes from u_i. This phase proceeds in the following steps.

1. Every party P_i breaks his share of GC (namely GC_i) further into n shares: $\{GC_i^j, \ldots, GC_i^n\}$. P_i then engages in an execution of $ZKSend_{ij}$ with every party P_j in *parallel*. Here, the input of P_i is GC_i^j while P_j proves the truthfulness of an NP statement z which asserts that the party P_j was honest in the protocol up to the end of the Garbled Circuit Generation phase. More specifically, z is the following statement: There exists $X_j = (x_j, r_j, K_j)$, s_j and t_j such that,

   - There exist strings for the commitments $Com(X_j), Com(s_j)$ and $Com(t_j)$ (broadcast in the Input Commitment phase) that open the commitments to X_j, s_j and t_j respectively.
   - Given all the incoming messages to P_j (over the broadcast channel), the outgoing messages of P_j were honestly computed as per the protocol specifications in (a) *ExtractEnable* subprotocol using randomness s_j, (b) Garbled Circuit Generation phase using randomness t_j.

   Upon obtaining $\{GC_1^i, \ldots, GC_n^i\}$ after executing $ZKSend$ with other parties to get $(n-1)$ shares (the share GC_i^i is already held by P_i), the party P_i broadcasts $GC^i = \bigoplus_{j=1}^n GC_j^i$. Upon receiving GC^i for all i, all parties compute $GC = \bigoplus_{i=1}^n GC^i$.

   We now give some intuition on the rationale behind this sub-protocol. Recall that a party P_i holds a share of the garbled circuit at the end of the Garbled Circuit Generation phase. P_i would like to give its share out only if the other $n-1$ parties were honest. However, if any party deviated from the prescribed protocol during the Input commitment or Garbled Circuit Generation phase, it could be potentially dangerous for P_i to give out its garbled circuit share (since then garbled circuit is not guaranteed to be correctly constructed). Hence, P_i breaks its share further into n subshares and transfers these subshares in *parallel* to other parties P_j using $ZKSend_{ij}$. If any of the $n-1$ parties was dishonest previously, it is guaranteed that at least one of those n subshares will be lost (since

ZKSend will output a randomly selected value instead of the right subshare to a dishonest party). Thus, in case cheating occurred in previous stages, the garbled circuit for evaluating the functionality is essentially "lost".

We stress that all the ZKSend executions in this step are done in *parallel*. Therefore, this step contributes only a constant number of rounds to our protocol.

2. Each party P_i now evaluates the garbled circuit GC on its own (recall that P_i has the appropriate wire keys for all the input wires as well as the wire masks for all the output wires) to obtain the output $F(X_1, \ldots, X_n)$. P_i further checks if K_i is contained in $F(X_1, \ldots, X_n)$. Now, one of the following two cases occur:

   (a) If the check succeeds (i.e., K_i is contained in $F(X_1, \ldots, X_n)$), then P_i broadcasts the opening to the commitment $Com(GC_i)$ that it had broadcast earlier in the Garbled Circuit Generation phase. Then, having received the openings for each commitment $Com(GC_j)$, $j \neq i$, P_i first verifies their correctness. If all the openings are correct, then P_i computes the garbled circuit from the decommitted values (as well as its own share GC_i). Let $\hat{GC}$ denote this garbled circuit. Then, if $\hat{GC} = GC$, P_i returns the output of GC as its own output, else it outputs $\perp$.

   (b) Otherwise, P_i broadcasts a random string. In this case, the output of P_i is $\perp$.

/* As argued earlier, if any party was dishonest in the Input Commitment phase or the Garbled Circuit Generation phase, then the garbled circuit is "lost", which in turn implies that the keys K_i are "lost" as well. Therefore, each party will output $\perp$ in this case since the check on the correctness of K_i in step (2) will fail. Now consider the case that all the parties in the protocol were honest in the Input Commitment phase as well as the Garbled Circuit Generation phase. Now consider the following two cases:

1. First consider the case when the output is not favorable. Note that at the end of the Garbled Circuit Generation phase, the parties will hold shares of a garbled circuit that outputs a random value. Then, the secret key K_i is indistinguishable from random even if an adversary is given all the shares of this garbled circuit. Therefore, it follows from the security property of garbled circuits that this garbled circuit (or any other garbled circuit derived from it) cannot output K_i. Therefore, in this case the check on the correctness of K_i in step (2) will fail.

2. Now consider the case when the output is favorable. Note that in this case, it is acceptable for each party to send out the opening to the commitment of its garbled circuit share (since if the output is favorable, it follows that each party participated in the protocol, and therefore compromising covertness by sending out the opening to the commitment is ok). We stress that step 2(a) is crucial in ensuring the correctness of the final output in this case. This is because an adversary who behaved honestly in the Input Commitment phase as well as the Garbled Circuit Generation phase may still be able to force a modified garbled circuit (by cheating in the ZKSend executions) on the other parties that produces an incorrect, yet favorable output. The checks performed in step 2(a) rule out this possibility since any attempt on the part of the adversary to modify the resultant garbled circuit in step (1) will be detected. */

This completes the description of our covert multi-party computation protocol Π. The fact that Π is constant round follows simply by construction. In the next subsection, we formally prove its security.

5.3 Proof of Security

Theorem 2 *The proposed protocol* Π *is a secure (constant-round) covert multi-party computation protocol as per definition 1.*

We prove theorem 2 via the following three step process. We first give the construction of a simulator $\mathcal{S}$ that is able to simulate the view of any adversary $\mathcal{A}$ in an execution of Π. We then prove some specific

security properties of our ZKSend construct (described in section 5.1). Finally, (by relying on the security properties of ZKSend and other primitives used in the construction of Π) we will argue that the output distributions of the real and ideal world executions are computationally indistinguishable.

In this section, we only give the construction of the simulator $\mathcal{S}$. The remaining parts of our proof are given in appendix C.

5.3.1 Description of the Simulator

Simulator $\mathcal{S}$ receives a list of honest parties H. The set of malicious parties is M. We describe the strategy of $\mathcal{S}$ in each of the three phases of the protocol:

Input Commitment phase.

1. *InputRandomCommit*
   (a) For every $i \in H$, $\mathcal{S}$ picks X_i, s_i, t_i and u_i at random and broadcasts $Com(X_i), Com(s_i), Com(t_i)$ and $Com(u_i)$. Note that $X_i = (x_i, r_i, K_i)$, where x_i, r_i and K_i are random.
   (b) Let $IR_i = (Open(Com(X_i)), Open(Com(t_i)))$.
2. *ExtractEnable* We first describe the strategy of $\mathcal{S}$ in all executions of $ZKSend_{ij}$ depending upon whether it is acting as a sender or a receiver. Consider any execution of $ZKSend_{ij}$ where P_i is the sender and P_j is the receiver.
   (a) When $i \in H$, $\mathcal{S}$ uses the honest sender strategy with input IR_i (as defined above), except that in stage 2 and 3 of phase I, $\mathcal{S}$ uses a pseudo-random generator to generate the challenge strings r_1, r_2. This is to ensure that $\mathcal{S}$ has a short description of length at most κ^3 for each challenge string (this will be necessary when $\mathcal{S}$ is simulating the executions of $ZKSend_{ij}$ on behalf of honest receivers; see below).
   (b) Now consider the case when $i \in M$ and $j \in H$. Let us first recall the structure of $ZKSend_{ij}$. Note that an execution of $ZKSend_{ij}$ is identical to cZK except that instead of simply sending the last message m_{final} of cWI-UARG, P_j sends covert commitments to random shares of m_{final}, followed by a challenge-response phase and a garbled circuit evaluation. That is, prior to sending the covert commitments to the shares of m_{final}, the communication between P_j and P_i in $ZKSend_{ij}$ is identical to that between P and V in cZK. Let $\mathcal{S}_{\text{cZK}}$ be the simulator for the cZK protocol.

      In this case, $\mathcal{S}$ internally runs the algorithm for $\mathcal{S}_{\text{cZK}}$ to compute outgoing messages for P_j. More specifically, on receiving any message from P_i, $\mathcal{S}$ uses $\mathcal{S}_{\text{cZK}}$ to prepare P_j's response. Note that $\mathcal{S}_{\text{cZK}}$ will require the code of the adversary (controlling P_i) along with short descriptions of all the messages of $\mathcal{S}$ contained[6] in slot 1 or slot 2 (this includes the messages sent by $\mathcal{S}$ in other executions of ZKSend, in particular the long challenge strings r_1, r_2) in order to compute the outgoing messages. We observe that $\mathcal{S}$ already has all the necessary information for $\mathcal{S}_{\text{cZK}}$. In this manner, $\mathcal{S}$ is able to compute all the outgoing messages for P_j until stage $4(a)$. Now, when $\mathcal{S}_{\text{cZK}}$ prepares the final message m_{final} of cWI-UARG, then instead of passing it directly to P_i, $\mathcal{S}$ computes its random shares and sends covert commitments to these shares to P_i. $\mathcal{S}$ now executes the rest of $ZKSend_{ij}$ honestly with P_i by using m_{final}. In particular, $\mathcal{S}$ first executes a challenge-response protocol with P_i and reveals the shares to m_{final} honestly as chosen by P_i. Then, $\mathcal{S}$ evaluates the garbled circuit (also executes OT protocols with P_i) and obtains an output.

In the ExtractEnable phase, $\mathcal{S}$ uses the following strategy. For every pair (P_i, P_j),

[6] Messages that are exchanged between the commitment c_b and the challenge string r_b are said to be 'contained' in slot b.

- When $i \in H$, $\mathcal{S}$ first chooses a random r and sends $y = f_{owp}(r)$ (where f_{owp} is a one-way permutation). It then engages in an execution of $ZKSend_{ij}$ with P_j on the following NP statement as the common input: "$\exists r$ such that $y = f_{owp}(r)$". $\mathcal{S}$ uses input IR_i and follows the sender strategy described above.
- Now consider the case when $i \in M$ and $j \in H$. On receiving a value y from P_i, $\mathcal{S}$ will attempt to extract IR_i from P_i by simulating $ZKSend_{ij}$ (for the NP statement: "$\exists r$ such that $y = f_{owp}(r)$") using the simulation strategy described above. Note that if P_i behaved honestly, then except with negligible probability, $\mathcal{S}$ would obtain IR_i as the output. Otherwise, $\mathcal{S}$ might fail to extract the input and randomness of P_i.

Garbled Circuit Generation phase. Let C denote the circuit for the functionality F. Then, for every honest party $i \in H$, $\mathcal{S}$ picks random wire mask shares λ_i^w and key shares $k_i^{w,\ell}$ ($\ell \in \{0,1\}$) for all wires w in the circuit C. Now, $\mathcal{S}$ honestly executes the Garbled Circuit Generation phase (except the last step where the parties broadcast covert commitments to their garbled circuit shares) on behalf of all the honest parties using the wire masks and key shares generated as above (as well as the random inputs chosen in the input commitment phase). We now consider two cases:

1. **Case 1:**

   First consider the case where the following two conditions hold:

   (a) $\mathcal{S}$ was successful in extracting IR_i for all $i \in M$.

   (b) All the parties were honest in the Garbled Circuit Generation phase. $\mathcal{S}$ checks this as follows. Note that if (a) is true, then $\mathcal{S}$ has the input X_i and the randomness u_i for all i (since it successfully extracted IR_i for all $i \in M$). Since these values deterministically define each message of the Garbled Circuit Generation phase, $\mathcal{S}$ can simply check if all the parties behaved honestly.

   In this case, $\mathcal{S}$ queries the Ideal functionality with the inputs (x_i extracted from P_i, for all $i \in M$) of all $i \in M$. The Ideal functionality returns either $f(x_1, \ldots, x_n)$ or $\perp$. Now $\mathcal{S}$ constructs a simulated garbled circuit (as in the protocol of Beaver et al [BMR90], see also [LP04]). If the Ideal functionality returned $f(x_1, \ldots, x_n)$, then the output of this garbled circuit is the value $f(x_1, \ldots, x_n), K_1, \ldots, K_n$ (where $K_1, \ldots, K_n$) are the keys chosen by all parties in the input commitment phase). For all $i \in M$, $\mathcal{S}$ uses K_i from IR_i. Otherwise, if the ideal functionality returned $\perp$, then the output of this garbled circuit is a random string R. Let GC be the simulated garbled circuit. $\mathcal{S}$ then changes the values of garbled circuit shares GC_i of honest parties in such a way that $(\bigoplus_{i \in H} GC_i) \oplus (\bigoplus_{i \in M} GC_i) = GC$. Note that, since $\mathcal{S}$ knows the garbled circuit shares GC_i of malicious parties, it picks shares at random for honest parties such that the above conditions hold.

2. **Case 2:**

   If $\mathcal{S}$ failed to extract inputs and randomness of all malicious parties, or if some party was dishonest in the Garbled Circuit Generation phase, then $\mathcal{S}$ changes the garbled circuit share of each honest party to a random value.

Finally, $\mathcal{S}$ broadcasts a covert commitment $Com(GC_i)$ for each honest party $i \in H$. Here GC_i is the garbled circuit share of P_i determined by $\mathcal{S}$ depending upon whether case 1 or 2 was true.

Output Exchange phase. First recall the strategy of $\mathcal{S}$ to simulate $ZKSend_{ij}$ (on behalf of the receiver), as explained in the Input Commitment phase. We now explain the strategy of $\mathcal{S}$ in the Output Exchange phase for each honest party $i \in H$.

1. $\mathcal{S}$ first breaks GC_i into n random shares $GC_i^1, \ldots, GC_i^n$.

2. In all executions of ZKSend where i is the sender, $\mathcal{S}$ honestly executes $ZKSend_{ij}$ with every party $j \neq i$ on the input GC_i^j. On the other hand, in all executions of ZKSend where i is the receiver, $\mathcal{S}$ simulates $ZKSend_{ij}$ to extract GC_j^i from P_j.

   We stress that $\mathcal{S}$ simulates all the ZKSend executions in *parallel*.

3. Upon receiving $\{GC_1^i, \ldots, GC_{i-1}^i, GC_{i+1}^i ...., GC_n^i\}$ (note that P_i has GC_i^i already), $\mathcal{S}$ broadcasts $GC^i = \bigoplus_{j=1}^n GC_j^i$ on behalf of P_i. Upon receiving all the broadcast values, $\mathcal{S}$ computes the garbled circuit $\hat{GC} = \bigoplus GC^i$.

4. $\mathcal{S}$ now evaluates $\hat{GC}$ (recall that P_i must have the appropriate keys for input wires as well as the wire masks for the output wires from the Garbled Circuit Generation phase) and checks if K_i is present in the output. Now one of the following two cases occur:

   (a) If K_i is present in the output, then $\mathcal{S}$ broadcasts the opening of $Com(GC_i)$. Upon receiving all the broadcast values, for every $j \in M$, $\mathcal{S}$ verifies the opening to the commitment sent by P_j at the end of the Garbled Circuit Generation phase. If all the openings are correct, then $\mathcal{S}$ computes the garbled circuit from the decommitted values (as well as its own share GC_i). Let GC denote this garbled circuit. $\mathcal{S}$ now checks whether $GC = GC'$. If the check succeeds, then $\mathcal{S}$ instructs the trusted party to return the correct output to the honest parties. Otherwise, $\mathcal{S}$ instructs the trusted party to return $\perp$.

   (b) Otherwise $\mathcal{S}$ broadcasts a random string on behalf of each honest party. Finally, $\mathcal{S}$ instructs the trusted party to return $\perp$.

This completes the description of the simulator strategy $\mathcal{S}$. The remaining parts of our proof are given in appendix C.

6 Covert Multi-Party Computation over Point-to-Point Channels

In this section, we consider the network model where no broadcast channel is available, instead each pair of parties only share a private communication channel. We describe a constant-round covert computation protocol for a *constant* number of parties in this network model.

6.1 A Bounded-Concurrent Covert Computation Crotocol for Reactive Functionalities

As an essential ingredient to the construction of our new protocol, we require a bounded-concurrent covert two-party computation protocol for *reactive* functionalities. To this end, we first discuss the design of a bounded-concurrent covert multi-party computation protocol Σ for reactive functionalities over a broadcast channel (later in our construction, we will use Σ for the two-party case only). Here, a (covert) reactive functionality (say) F is a (covert) ideal functionality that can be invoked a fixed number (say t) of times. In each of the first $t-1$ queries, if the output is not favorable or if some party is not participating, then F outputs a random value instead of $\perp$. We note that this is necessary to preserve covertness of the parties in the first $t-1$ invocations. On the last query, however, F returns $\perp$ if either the output is not favorable or some party is not participating. We note that this is necessary to guarantee the correctness of the *final* output.

Now consider a covert reactive functionality F that can be queried t times. Then Σ (that realizes F in the bounded-concurrent case) consists of t sub-protocols $\Pi_1, \ldots, \Pi_t$ (described below), where each sub-protocol corresponds to one query to F in the ideal world. Now note that in the ideal world, F would carry a "state" from one query to another. Then, in order to emulate F correctly, each sub-protocol in Σ will output the state of F which is then carried over to the next sub-protocol. To this end, the parties jointly compute a secret key prior to the first sub-protocol execution in Σ. In particular, each party P_i commits to a random key share K_i using the covert commitment scheme Com. Let $K = \bigoplus_{i=1}^n K_i$. Then, K is

used to encrypt and authenticate[7] the state of F when it is transferred from one sub-protocol to another. Note that any pseudo-random function can be used as a covert MAC for authentication (since its output is indistinguishable from random).

We now describe the sub-protocols $\Pi_1, \ldots, \Pi_t$. We first observe that using standard techniques (see [Pas04]), the covert computation protocol in section 5 can be modified such that the resultant protocol can securely realize a (covert) non-reactive ideal functionality in the *bounded-concurrent* setting. Let Π denote the modified protocol. We now outline the main steps to modify Π into Π_i, $i \in [1, t]$.

1. In the Garbled Circuit Generation phase, the parties jointly construct a garbled circuit GC that is the same as the one described in section 5, except the following:
   (a) It has (built-in) covert commitments $Com(K_i), i \in [1, n]$.
   (b) It addition to the inputs of the parties, it takes as input the openings to the commitments to the key shares as well as the (encrypted and authenticated) state of F after the i^{th} query.
   (c) It returns the correct output only if the openings are correct and the input state is authenticated (else it outputs a random string). The output of GC consists of the output of F after the i^{th} query as well as its (encrypted and authenticated) state along with the index i.
2. No output correctness checks are performed in the Output Exchange phase, except in the final sub-protocol Π'_t. For each of the first $t-1$ sub-protocols, we summarize the necessary changes as follows:
   (a) The parties no longer keep a random secret key for checking the output correctness. Consequently, the key validation check in the Output Exchange phase is now void.
   (b) The parties neither broadcast the commitment to their garbled circuit shares (at the end of the Garbled Circuit Generation phase), nor their openings (in the Output Exchange Phase). Consequently, the garbled circuit verification step in the Output Exchange phase is now void.

We now describe our covert multi-party computation protocol over point-to-point channels. Let F be the covert functionality that we wish to realize. In our construction, we will be using Σ as a black-box only.

6.2 The New Protocol

Consider parties $(P_i, \ldots, P_n)$ with inputs $(x_1, \ldots, x_n)$. Without loss of generality, assume that $n = 2^\delta$ for some constant δ. As a part of the protocol specification, the n parties are grouped into $n/2$ pairs, each consisting of 2 parties. Then, informally speaking, the protocol execution is defined by a tree that consists of $\log(n)+1$ levels where each level $i \in [0, \log(n)]$ consists of 2^i reactive functionalities. The n functionalities at $\log(n)$-level are emulated by the protocol parties. A pair of protocol parties (as specified by the protocol) at level $\log(n)$ together run the protocol Σ to realize a functionality at level 1. Similarly, a pair of reactive functionalities (virtual parties) at level i together run Σ to realize a functionality at level $(i-1)$. Then, the output of the protocol is simply the output of the functionality at level 0. We now describe the functionalities at each level, starting from the root at level 0.

0-Level Functionality: This functionality takes as input two vectors $\vec{X}_1 = (x_1, \ldots, x_{n/2})$ and $\vec{X}_2 = (x_{(n/2)+1}, \ldots, x_n)$. The output of this functionality is $F(x_1, \ldots, x_n)$ (where F is the ideal functionality that the protocol realizes).

1-Level Functionality: There are two functionalities at the first level "interacting" with each other. In other words, the output of one functionality is routed (via other functionalities, see below) to the other other functionality as its input. Then the output of the other functionality is routed back to the first functionality as its input. The 1-level functionalities are described by the (virtual) parties running the two-party protocol Σ to realize the 0-level functionality. We will refer to them as the *partner* functionalities. Looking ahead,

[7] Let $K = (K^1, K^2)$. Then K^1 is used for encryption and K^2 is used for authentication.

one of the (virtual) parties (that describe the 1-level functionalities in the real world) will play the role of a *designated* (virtual) party that routes messages between two higher level partner functionalities. Note that since there is only functionality at 0-level, there is no difference between the designated party and the non-designated party at 1-level (this is not the case for i-level functionalities, $i > 1$; see below). The *final* output of the 1-level functionalities is the output they receive from the 0-level functionality.

i-Level Functionality: Consider an $(i-1)$-level functionality $f_{(i-1)}$. There exist two i-level *partner* functionalities f_i^0, f_i^1 for every such $f_{(i-1)}$. The reactive functionalities f_i^0, f_i^1 are defined by the (virtual) parties that run the protocol Σ to realize $f_{(i-1)}$ (where both of them have the necessary inputs). Since $f_{(i-1)}$ is reactive functionality, it may give out several intermediate outputs before giving a final output. Only one of the i-level functionalities f_i^0, f_i^1 receives the intermediate outputs. We will refer to it as the *designated* party (say) f_i^d. Further, we say that the designated functionality f_i^d *hosts* the $(i-1)$ level functionality $f_{(i-1)}$. The final output of f_i^0, f_i^1 is the output they receive from $f_{(i-1)}$.

Note that the $(i-1)$-level functionality $f_{(i-1)}$ is essentially a (virtual) party that interacts with a partner functionality at level $(i-1)$ in an execution of Σ. Then the intermediate outputs of $f_{(i-1)}$ are simply the protocol messages to be forwarded to its partner functionality. To this end, the designated functionality f_i^d forwards (using a private channel) each intermediate output from $f_{(i-1)}$ to another i-level designated party hosting the partner functionality of $f_{(i-1)}$. Further, f_i^d waits to receive a message (in reply) before invoking $f_{(i-1)}$ again. This invocation would require an execution of Σ with the partner functionality f_i^{1-d}.

$\log(n)$-Level: As mentioned earlier, the functionalities at $\log(n)$ level are emulated by the real parties in the protocol. The input of a $\log(n)$-level functionality is the same as the input of the real party emulating it. Each pair of partner functionalities at $\log(n)$ level engage in an execution of Σ to realize a $(\log(n)-1)$ level functionality. Further these functionalities can directly send messages to each other using their private channel. The final output of these functionalities (i.e. the final output of the protocol) is the output they receive from the $(\log(n)-1)$-level functionality that they realize. Then it follows from our construction that this is simply the output of the 0-level functionality.

Now recall that two $(i-1)$-level partner functionalities communicate with each other by routing messages through the i-level designated parties hosting them. However, one or both of these designated parties may be dishonest, and may therefore modify the messages in transit. Such a situation was considered in [BCL$^+$05]. In order to solve this problem, we will make use of a covert one-time signature (OTS) scheme. During the setup phase of the protocol, each party P_i creates $n-1$ key pairs of a covert OTS scheme, one for each party $j \neq i$. For each $j \neq i$, P_i sends one public key to P_j over their private channel. Then, the new input of P_i consists of $n-1$ signatures over its original input (one with each private key generated earlier), and $n-1$ one-time public keys received from the parties $j \neq i$. Now, the 0-level functionality takes all these input values and returns the correct output only if the input public keys are consistent and all the signatures are valid. We remark that the OTS scheme of Lamport [Lam79] can be modified into a covert OTS scheme if we use one-way permutations instead of one-way functions.

Further, since there are 2^{i-1} pairs of partner functionalities at each level i, at most 2^{i-1} executions of Σ may occur concurrently at each level. Summing up, $\sum_{i=1}^{\log(n)} 2^{i-1} = n-1$ concurrent executions of Σ are possible in an execution of our protocol. Note that any of these executions (say) s can be controlled by a designated party acting as a man-in-the-middle, who can instead run separate executions of s with each of the partner parties running s. Following techniques from Barak et al [BCL$^+$05], this problem can be solved if Σ is $2(n-1)$-bounded concurrent secure. This completes the description of our protocol.

6.3 Protocol Analysis

Efficiency. We first compute the round complexity for our protocol by induction. Recall that Σ consists of t sub-protocols $\{\Pi_i\}_{i=1}^t$ if the functionality F to be realized can be queried t times. Let r be the (constant) number of rounds in Π_i. Now first consider the (virtual) parties at level 1 that run Σ to realize the 0-level

functionality. Note that this instance of Σ only consists of r rounds since it only consists of a single sub-protocol Π_1 (that corresponds to the single allowed query to the 0-level functionality). Now consider a pair of (virtual) parties (that describe a pair of partner functionalities f_i^0, f_i^1) at level $i \geq 1$ that run Σ (to realize an $(i-1)$-level functionality). Assume that this instance of Σ consists of r^i rounds. This implies that each functionality f_i^j ($j \in \{0,1\}$) must be queried r^i times. Now consider the execution of Σ between two (virtual) parties at level $(i+1)$ that realizes f_i^j. Then, this instance of Σ must consist of r^i sub-protocols $\{\Pi_k\}_{k=1}^{r^i}$, where each sub-protocol has a round complexity of r. This in turn implies that this instance of Σ consists of r^{i+1} rounds. Then by induction, the round complexity of our protocol for any pair of partner (real) parties is $r^{\log(n)} = n^{\log(r)}$, which is a polynomial (in the security parameter) if n is a polynomial.

We now compute the computational complexity for our new protocol by induction. We first observe a property of the covert computation protocol Π described in section 5 (also see appendix **??**). Let C be a circuit for the ideal functionality that Π realizes. Then there exists at least one round in Π that requires computation $\text{poly}(\kappa)|C|$ (where κ is the security parameter). Now, let C be the circuit corresponding to the 0-level functionality in our new protocol. Consider the execution of protocol Σ between the (virtual) parties at level 1 to realize the 0-level functionality. Then it follows from the aforementioned property of our covert computation protocol Π in section 5 (since Σ consists of sub-protocols Π_j that are derived from Π) that there exists a round in this execution of Σ that requires computation $\text{poly}(\kappa)|C|$. Now consider an execution of Σ between two partner functionalities at level i. Assume that there exists a round in this execution that requires computation $\text{poly}(\kappa)^i|C|$. Then the circuit size for any of these partner i-level functionality (say) f_i must be $\text{poly}(\kappa)^i|C|$. It follows then that there exists a round in the execution of Σ between the $(i+1)$-level partner functionalities realizing f_i that requires computation $\text{poly}(\kappa)^{i+1}|C|$. Then by induction, the computational overhead of our protocol is $\text{poly}(\kappa)^{\log(n)}|C|$. When n is a constant, this reduces to $\text{poly}(\kappa)|C|$. Note that in this case, the round complexity of our protocol is constant.

In a recent work, Ishai et al [IKOS08] designed a multi-party computation protocol for the semi-honest case with a constant computational overhead (i.e., $|C| + \text{poly}(\kappa)$, where C is the circuit to be evaluated). It remains an open problem to extend their work to the case of malicious parties and covert functionalities.

Proof of Security. We now give a sketch of the main arguments to prove that privacy of the inputs and covertness of the parties is maintained in our protocol.

Consider an honest party at $\log(n)$-level. Note that even if its partner functionality is cheating, the functionality emulated by them must be honest (i.e., correctly emulated). This implies that at least one functionality at $(\log(n)-1)$ level is correctly emulated. Now we consider our protocol in the $(\log(n)-1)$-level hybrid model. There are $n/2$ virtual parties at $(\log(n)-1)$ level of which at least one is honest. By induction, we note that there are two virtual parties at level 1, of which at least one is honest. Hence, we conclude that the 0 level functionality must be honest (i.e., correctly emulated)

Further note that there are two kinds of messages exchanged by the parties in our protocol. The first kind consists of the messages of the protocol Σ exchanged by the partner functionalities. From the security of our protocol in section 5, we conclude that these messages are indistinguishable from random, and therefore do not compromise the covertness of the parties. The second kind consists of the messages routed by a pair of designated functionalities between the higher level functionalities that they host. Note that these are the messages of protocol Σ run by the higher level functionalities, and are indistinguishable from random to all the parties including the designated parties.

7 Acknowledgements

We thank Yuval Ishai for useful discussions. We also thank Chris Peikert for his extremely useful comments that helped us in improving the presentation of our results.

References

[ABK07] Giuseppe Ateniese, Marina Blanton, and Jonathan Kirsch. Secret handshakes with dynamic and fuzzy matchin. In *NDSS*, 2007.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, 2004.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. *Computational Complexity*, 2008.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, 2001.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, 2005.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, 2003.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *CCC*, 2002.

[BL04] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. *SIAM J. Comput.*, 2004.

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

[CGOS07] Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *FOCS*, 2007.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[HLvA02] Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. In *CRYPTO*, 2002.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC*, 2008.

[KK07] Jonathan Katz and Chiu-Yuen Koo. Round-efficient secure computation in point-to-point networks. In *EUROCRYPT*, 2007.

[KKK08] Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of vss in point-to-point networks. In *ICALP (2)*, 2008.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, SRI, 1979.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. *ECCC*, 2004.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, 2001.

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

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

[vAH04] Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In *EUROCRYPT*, 2004.

[vAHL05] Luis von Ahn, Nicholas Hopper, and John Langford. Covert two-party computation. In *STOC*, 2005.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, 1982.

A Covert Communication Channel

In this section, we briefly describe the covert broadcast and point-to-point channels. Most of the text here is taken almost verbatim from [CGOS07].

Broadcast Channels. Messages are drawn from a set of documents denoted as D. We assume that time proceeds in discrete timesteps. Each party $P \in \{P_1, P_2, ..., P_n\}$ maintains a history h_P, which represents a list of all documents sent and received by P ordered by timestep. Let the set of well-formed histories be H. We associate to each party P, a family of probability distributions $B^P = \{B^P_h\}_{h \in H}$ on D. Communication over broadcast channel $B = (D, H, B^{P_1}, B^{P_2}, ..., B^{P_n})$ proceeds as follows: At each timestep, every party P receives messages broadcast over the channel in the previous timestep by all parties, updates h_P accordingly and draws a document $d \leftarrow B^P_{h_P}$ (Note that this may result in $\perp$). d is broadcast and h_P is updated. We further assume that all messages sent at a given timestep are received in the next one. Let $B^P_{h_P} \neq \perp$, be the distribution $B^P_{h_P}$ conditioned on not drawing $\perp$. We will consider families of broadcast channels $\{B_k\}_{k \geq 0}$ (where k is polynomially related to κ) such that the following conditions hold:

- The length of elements in D_k is polynomially-bounded in k
- For each $h \in H_k$ and for every party P, either $Pr[B^P_h = \perp] = 1$ or $Pr[B^P_h = \perp] \leq 1 - \delta$ for some constant δ.
- There exists a function $l(k) = \omega(\log k)$ such that for each $h \in H_k$, $H_\infty((B^P_h)_k \neq \perp) \geq l(k)$ (that is, there is some variability in the communications).

Party P can draw from B^P_h for any history h and the adversary can draw from B^P_h for every party P and history h. Parties running the protocol try to communicate according to using sequence of documents that appear to come from B.

Point-to-Point Channels. Each pair of parties P_i, P_j share a private channel to communicate with each other. A private channel for two parties is modeled as a broadcast channel (as defined above) for the two-party case. A message sent by P_i over its private channel with P_j is only received by P_j. We assume that a message sent at a given timestep is received in the next one. Each party P maintains a "global" history, which represents a list of all documents sent and received by P ordered by timestamp. On receiving a message over a private channel, a party updates its history accordingly.

B Covert Computation Primitives

In this section, we describe the covert computation primitives that are used in our positive results. The text in this section is taken almost verbatim from [CGOS07] (except the description of a covert encryption scheme in NC^0; see below).

B.1 Covert Commitments

We will need a perfectly binding non-interactive bitwise commitment scheme with commitments that are indistinguishable from random bits. An example of such a scheme [vAHL05] is the scheme that commits to the bit b by $com(b;(r,x)) = r||\pi(x)||(x \cdot r) \oplus b$ where π is a one-way permutation on the domain $\{0,1\}^k$, $x \cdot y$ denotes the inner-product of x and y over $GF(2)$, and $x, r \leftarrow U_k$. Now we have the follow lemma from [vAHL05]:

Lemma 2 (Covert Commitment [vAHL05]) *The non-interactive bit commitment scheme based on hard-core predicates [GL89] is indistinguishable from a message drawn at random from the uniform distribution.*

B.2 Covert 1-out-of-4 Oblivious Transfer

We present below a covert 1-out-of-4 Oblivious Transfer protocol (which is a modification of the Covert 1-out-of-2 Oblivious Transfer protocol presented in [vAHL05, NP01]). It is based on the Decisional Diffie-Hellman assumption.

Decisional Diffie-Hellman Assumption: Let p and q be two primes such that $q|p-1$. Let $\mathbb{Z}_p^*$ be the multiplicative group of integers modulo p, and let $g \in \mathbb{Z}_p^*$ have order q. Denote the advantage of an adversary A as $Adv_A(g,p,q) = |Pr_{a,b,r}[A_r(g^a, g^b, g^{ab}, g, p, q) = 1] - Pr_{a,b,c,r}[A_r(g^a, g^b, g^c, g, p, q) = 1]|$, where A_r denotes the adversary A running with random tape r, a, b, c chosen uniformly at random from $\mathbb{Z}_q$, with all multiplications being over $\mathbb{Z}_p^*$. The Decisional Diffie-Hellman Assumption states that for every PPT A, for every sequence $\{(g_k, p_k q_k)\}_k$ satisfying $|p_k| = k$ and $|q_k| = \theta(k)$, $Adv_A(g_k, p_k, q_k)$ is negligible in k (k is polynomially related to security parameter κ).
Setup for Oblivious Transfer: Let $p = rq+1$, where q is a large prime, $2^k < p < 2^{k+1}$ and $gcd(r,q) = 1$. Let g be a generator of $\mathbb{Z}_p^*$ and let $\gamma = g^r$ be a generator of the unique multiplicative subgroup of order q. Let $\hat{r}$ be the least integer such that $r\hat{r} = 1 \ mod \ q$. Assume $|m_0| = |m_1| = |m_2| = |m_3| < k/2$. $H : \{0,1\}^{2k} \times \mathbb{Z}_p \rightarrow \{0,1\}^{k/2}$ is a pairwise-independent family of hash functions. Define the randomized mapping $\phi :< \gamma > \rightarrow \mathbb{Z}_p^*$ by $\phi(h) = h^{\hat{r}} g^{\beta q}$, for a uniformly chosen $\beta \in \mathbb{Z}_r$.

Protocol Description (COT_1^4):

1. On input $\sigma \in \{0,1\}^2$, chooser C chooses uniform $a, b \in \mathbb{Z}_q$ and sets $c_\sigma = ab \ mod \ q$ setting uniformly $c_i \in \mathbb{Z}_q, \ \forall \ i \neq \sigma$. C then sets $x = \gamma^a, y = \gamma^b, z_i = \gamma^{c_i} \ \forall \ i$ and then sets $x' = \phi(x), y' = \phi(y), z_i' = \phi(z_i) \ \forall \ i$. If the most significant bits of all of $x', y', z_0', z_1', z_2', z_3'$ are 0, then C sends the least significant k bits of each to S; otherwise C starts all over again.

2. The sender S recovers x, y, z_0, z_1, z_2, z_3 by raising all received values to the power r, picks f_0, f_1, f_2 and $f_3 \in H$ and then does the following $\forall \ i$: S repeatedly chooses uniform $r_i, s_i \in \mathbb{Z}_q$ and sets $w_i = x^{s_i} \gamma^{r_i}, w_i' = \phi(w_i)$ until he finds a pair with $w_i' \leq 2^k$. He then sets $K_i = z_i^{s_i} y^{r_i}$.

3. S sends $w_i'||f_i||f_i(K_i) \oplus m_i, \ \forall \ i$ to C.

4. C recovers $K_\sigma = (w_\sigma')^{rb}$ and computes m_σ.

The proof of the following four lemmas follow from the security of the OT [NP01] and from the covertness property of 1-out-of-2 OT proved in [vAHL05].

Lemma 3 (OT Receiver [NP01, vAHL05]) *For any $\sigma, \tau \in \{0,1\}^2$, and for any PPT adversary $\mathcal{S}'$ that executes the sender's part, the views of $\mathcal{S}'$ in the case when the receiver tries to obtain M_σ and in the case when the receiver tries to obtain M_τ are computationally indistinguishable (given $\{M_0, M_1, M_2, M_3\}$).*

Lemma 4 (OT Sender [NP01, vAHL05]) *For any deterministic adversary $\mathcal{C}'$ that executes the receiver's part ($\mathcal{C}'$ is not necessarily poly-time), for any auxiliary input z that is polynomial in the security parameter κ, and for any $M_0, M_1, M_2, M_3 \in \{0,1\}^{l(\kappa)}$, there exists $\gamma \in \{0,1\}^2$ such that for every $W_1, W_2, W_3 \in \{0,1\}^{l(\kappa)}$, the view of $\mathcal{C}'(z)$ when interacting with honest sender $S(1^\kappa, M_\gamma, W_1, W_2, W_3)$, and the view of $\mathcal{C}'(z)$ when interacting with honest $S(1^\kappa, M_0, M_1, M_2, M_3)$ are statistically indistinguishable.*

Lemma 5 (Covert OT Receiver [NP01, vAHL05]) *For any honest R that executes the receiver's part in the OT protocol, and for any PPT adversary $\mathcal{S}'$ that executes the sender's part, the views of $\mathcal{S}'$ in the case when R sends messages according to the protocol and the case when R sends message drawn at random from U_κ (where U_κ denotes the uniform distribution) are indistinguishable.*

Lemma 6 (Covert OT Sender [NP01, vAHL05]) *For any honest S executing the sender's part in the OT protocol, and for any PPT adversary $\mathcal{C}'$ that executes the receiver's part, the views of $\mathcal{R}'$ in the case when S sends messages according to the protocol and the case when S sends message drawn at random from U_κ (where U_κ denotes the uniform distribution) are indistinguishable.*

B.3 Covert Yao's Garbled Circuit

Alice (holding input x) and Bob (holding input y) wish to compute a function $f(x, y)$ covertly in such a way that only Bob gets the output at the end, with no guarantee if the output is correct or not. This can be done using a covert version of Yao's garbled circuits called covert garbled circuits due to [vAHL05]. A covert garbled circuit construction [vAHL05] was obtained by a few modifications to the standard garbled circuit construction due to Yao [Yao82]. A covert garbled circuit can then be used in conjunction with covert 1-out-of-2 OT to compute $f(x, y)$. The following lemma follows from [vAHL05, LP04, Yao82]:

Lemma 7 (Garbled Circuit Security) *Consider two functions $f : \mathbb{D} \to \mathbb{R}$ and $f' : \mathbb{D} \to \mathbb{R}$, such that for all $x \in \mathbb{D}$, $f(x) = f'(x)$. Let GC represent a garbled circuit that computes f and GC' be a garbled circuit that computes f'. If $|GC| = |GC'|$ (where $|GC|$ represents the size of garbled circuit GC), then the distribution of GC and the distribution of GC' (over the random coins needed to prepare the garbled circuits) are indistinguishable.*

B.4 Covert GMW Multi-party computation

We consider the GMW ([GMW87]) protocol for multi-party computation (in the honest-but-curious model) and replace the semi-honest oblivious transfer protocol with a covert 1-out-of-4 oblivious transfer protocol. We also omit the output share broadcast phase from the GMW protocol. We call the resulting protocol Covert-GMW and describe it below in more detail:

Parties $\{P_1, P_2, ..., P_n\}$ having inputs $\{x_1, x_2, ..., x_n\}$ wish to compute $f(x_1, x_2, ..., x_n)$ covertly. They prepare a description of the circuit that outputs $f(x_1, x_2, ..., x_n)$ if $f(x_1, x_2, ..., x_n)$ is favorable, and otherwise, outputs a random value from the uniform distribution. Note that this circuit can be built using just NOT($\neg$) and AND($\wedge$) gates. In order to evaluate the circuit, the parties run the following protocol:

- P_i shares each of his input bits with all other parties in a way that exactly n parties are needed in order to gain information about the input bit. In particular, set $x_i = \bigoplus_{k=1}^{n} x_i^k$ and give P_j bit x_i^j.
- In order to compute a $\neg$ gate, one of the parties, say P_1, adds the constant 1 to his share and the other parties leave their shares as it is.
- In order to compute a $\wedge$ gate, we proceed as follows:
  - Let the two input wires be c and d that are distributed in shares $\{c_1, c_2, ..., c_n\}$ and $\{d_1, d_2, ..., d_n\}$. We wish to have a protocol at the end of which P_i holds b_i such that $\bigoplus_{k=1}^{n} b_k = b = c \cdot d$.

- Define $b_{i,j}$ so that for every $i, b_{i,i} = c_i d_i$ and for every $i \neq j, b_{i,j} + b_{j,i} = c_i d_j + c_j d_i$, and let $b_i = \sum_{j=1}^{n} b_{i,j}$.
- Note that $c \cdot d = \sum_{i=1}^{n} b_i = b$.
- P_i can compute $b_{i,i}$ on his own.
- In order to compute $b_{i,j}$ and $b_{j,i}$, P_i and P_j run a covert 1-out-of-4 oblivious transfer protocol with the following values. P_j picks $b_{j,i}$ at random and will transfer to P_i one of the following values: $b_{j,i}, b_{j,i} + d_j, b_{j,i} + c_j$ or $b_{j,i} + c_j + d_j$. P_i will choose to receive the first value, if $c_i = 0; d_i = 0$, the second value if $c_i = 0; d_i = 1$ and so on.

- Therefore, at the end of this protocol, all parties have a share of the output.

B.5 A Covert Encryption Scheme in NC^0

A covert encryption scheme is an encryption scheme with the additional property that, very informally, the distribution of a tuple consisting of an encryption key and an encryption of a random number with that key looks indistinguishable from the uniform distribution. The construction of our covert computation protocol relies on the existence of a covert encryption scheme in NC^0. Such an encryption scheme can be constructed using techniques from Applebaum et al [AIK04]. Given a function $f(x)$, Applebaum et al [AIK04] define and construct a *randomized encoding* function $\hat{f}(x, r)$. We rely on the following two key properties of their randomized encoding function. Firstly, the output distribution of $\hat{f}(x, r)$ depends only on $f(x)$ and does not further depend on x. In other words, there exists a simulator which on input $f(x)$ produces an output distribution which is indistinguishable from the distribution of $\hat{f}(x, r)$. Secondly, the construction in [AIK04] (see the Locality Construction in section 4.3) satisfies the property that if $f(x)$ is indistinguishable from the uniform distribution (where the input x is drawn from the specified input distribution), so is $\hat{f}(x, r)$. Given an encryption scheme (G, E, D) in NC^1, Applebaum et al [AIK04] show how to modify the decryption function such that the corresponding encryption function is the randomized encoding $\hat{E}(., r)$ which is computable by NC^0 circuits.

We start with a covert encryption scheme in NC^1 (such schemes can be readily constructed based on a variety of cryptographic assumptions) and similarly compile it to obtain a modified encryption scheme where the encryption function is in NC^0. It is easy to show that the modified encryption scheme would still be a covert encryption scheme through a standard hybrid argument. Assume an adversary $\mathcal{A}$ which can distinguish the distribution of a tuple consisting of an encryption key and an encryption of a random number from that key in the modified scheme from the uniform distribution with a non-negligible advantage. Then we can construct a new adversary $\mathcal{B}$ which takes as input a tuple (a, b) (which is either such a key, ciphertext pair for the original encryption scheme or a random number). The adversary $\mathcal{B}$ now computes the tuple $(a, S(b))$ where S is the simulator (guaranteed by the first property discussed above). The tuple $(a, S(b))$ is either a key, ciphertext pair for the modified encryption scheme or a random number. The adversary $\mathcal{B}$ can distinguish between these two cases by giving the tuple $(a, S(b))$ as input to the adversary $\mathcal{A}$. This contradicts the covertness of the original encryption scheme.

C Completing the Security Proof for Protocol Π

Recall that the description of the simulator algorithm $\mathcal{S}$ was given in section 5.3. In order to complete the security proof for our protocol Π, we first prove some specific security properties of our ZKSend construct and then prove that the output distributions of the real and ideal world executions are computationally indistinguishable.

C.1 Security Properties of ZKSend

In this section, we prove some security properties of the ZKSend protocol that are essential to proving the security of our constant-round covert multi-party computation protocol given in Section 5. We note that our

covert computation protocol consists of only *parallel* executions of ZKSend. However, some of the lemmas in this section are proven for the case of *bounded-concurrency* (which implies security for parallel executions). We remark that this will be useful when considering covert computation over point-to-point channels (see Section 6).

Lemma 8 (ZKSend Stand-Alone Weak Extractability) *Consider the following two security games: In the first game, honest party P_i executes $ZKSend_{ij}$ with P_j, where the input of P_i is a fixed value v. In the second game, honest party P_i executes $ZKSend_{ij}$ with P_j, where the input of P_i is a random string R (drawn from the uniform distribution). In both the games, the common input is an NP statement z. Consider a PPT adversary P_j who can distinguish between the distributions of the views in the two games with noticeable advantage ϵ. Then there exists an extractor E that extracts and outputs a witness for the statement z with probability polynomially related to ϵ.*

Proof. We first introduce some terminology to be used in the proof. Recall the description of $ZKSend_{ij}$ from section 5.1. Now consider an execution of $ZKSend_{ij}$. We define a *stage 5 prefix* to be the sequence of all the messages exchanged between P_i and P_j until the completion of stage 5 in phase I. Let $\{\gamma\}$ denote the set of all the decommitments to the shares of m_{final} (i.e., the last message of cWI-UARG). Then, we will say that a transcript of phase I is favorable if there exists $\{\gamma\}$ such that the garbled circuit $Gar[i \rightarrow j]$ on being input $\{\gamma\}$, outputs v.

Now let P_j be an adversary that can distinguish between the distributions of the views in the two games (as described above) with non-negligible advantage ϵ. Then we consider the following two cases:

Case I. For at least $\epsilon/2$ fraction of *stage 5 prefixes*, the transcript of phase I is favorable with probability at least $\epsilon/2$. We will call such *stage 5 prefixes* to be good. In this case, we show how to construct an extractor E that extracts and outputs a witness for the statement z with non-negligible probability.

Consider an execution of the cZK protocol where P proves an NP statement z to V. Let ϵ denote the probability with which V accepts. We note that there exists an extractor E_{cZK} for the cZK protocol (this follows from the existence of an extractor for WI-UARG of Barak and Goldreich [BL04]) that extracts and outputs a witness for z with probability polynomially related to ϵ. Then, informally speaking, our extractor E for $ZKSend_{ij}$ will work by running the extractor E_{cZK} and emulating a prover P to E_{cZK} that proves the same statement z that P_j proves to E in the execution of $ZKSend_{ij}$. E will simply output the same witness that E_{cZK} outputs.

As a first step, we will construct a prover P^* that interacts with an honest V in an execution of cZK such that V accepts the proof of the statement z with non-negligible probability. Informally speaking, P^* will work by emulating P_i to P_j by using the queries of V. It then uses the responses of P_j to answer the queries of V. More formally, P^* uses the following strategy.

1. On receiving any message from V, forward it to P_j.
2. For all but the last message from V, forward the response from P_j to V.
3. In order to answer the last message from V, rewind P_j in the stage 5 (challenge-response) of $ZKSend_{ij}$ to extract m_{final}. If the extraction is successful (explained below), then send m_{final} to V, else abort.

Recall that the correct response to the last message from V is the last message m_{final} of cWI-UARG. In $ZKSend_{ij}$, however, P_j never sends m_{final} in open; instead, P_j commits to random shares $\{\alpha_i^0\}_{i=1}^{\kappa}$, $\{\alpha_i^1\}_i^{\kappa}$ of m_{final}, where $\alpha_i^0 \oplus \alpha_i^1 = \beta_5$ for all i. Further in $ZKSend_{ij}$, P_j and P_i execute a challenge-response phase where P_j reveals the values $\alpha_1^{z_1}, \ldots, \alpha_\kappa^{z_\kappa}$ on receiving the challenge bits $z_1, \ldots, z_\kappa$. Now since the commitments to the shares are set at the completion of phase 3, Then P^* (emulating P_i) can rewind P_j in the challenge-response phase to obtain both α_i^0 and α_i^1 for some i with high probability.

We say that the extraction is successful if P^* receives correct values (i.e., the random shares that were committed to in stage 3) in both iterations of the challenge-response phase. Let E_{good} be the event that the *stage 5 prefix* (say) τ in the interaction of P^* with P_j is good. Let E_{fav} be the event that P^* obtains a

favorable transcript of phase I in two independent executions of the challenge-response phase with prefix τ. Then observe that V accepts the proof with the same probability that the extraction is successful (assuming that the completeness for cZK is 1). Then we have,

$$\begin{aligned}\Pr[V \text{ accepts}] &= \Pr[E_{\mathsf{good}}].\Pr[E_{\mathsf{fav}}] \\ &\geq \frac{\epsilon}{2}(\frac{\epsilon}{2})^2 \\ &\geq \frac{\epsilon^3}{8}\end{aligned}$$

which is non-negligible in κ.

Now our extractor E will simply run the extractor E_{cZK} on the prover P^*. E_{cZK} will output a witness for z with some probability polynomial in $\frac{\epsilon^3}{8}$. E simply outputs the same witness and stops.

Case II. Otherwise, for less than $\epsilon/2$ fraction of the *stage 5 prefixes*, the transcript of phase I is favorable with probability at least $\epsilon/2$. As earlier, we will call such *stage 5 prefixes* to be good.

Now fix any execution of $ZKSend_{ij}$. Let E_{good} denote the event that a *stage 5 prefix* is good. Let E_{fav} denote the event the a phase I transcript is favorable. We first derive an upper bound on E_{fav} over all possible executions of $ZKSend_{ij}$.

$$\begin{aligned}\Pr[E_{\mathsf{fav}}] &= \Pr[E_{\mathsf{good}}] \cdot \Pr[E_{\mathsf{fav}}|E_{\mathsf{good}}] + \Pr[\bar{E}_{\mathsf{good}}] \cdot \Pr[E_{\mathsf{fav}}|\bar{E}_{\mathsf{good}}] \\ &\leq \frac{\epsilon}{2} \cdot 1 + (1 - \frac{\epsilon}{2}) \cdot \frac{\epsilon}{2} \\ &\leq \epsilon - \frac{\epsilon^2}{4}\end{aligned}$$

Now, let E_1 denote the event the distinguisher P_j outputs 1 on the distribution of view in $ZKSend_{ij}$. Similarly, let E_2 be the event that P_j outputs 1 on the distribution of the view in $ZKSend_{ij}$. Then, by definition, we have:

$$\begin{aligned}\epsilon &= \Pr[E_1] - \Pr[E_2] \\ &= \Pr[E_{\mathsf{fav}}] \cdot \Pr[E_1|E_{\mathsf{fav}}] + \Pr[\bar{E}_{\mathsf{fav}}] \cdot \Pr[E_1|\bar{E}_{\mathsf{fav}}] - \Pr[E_{\mathsf{fav}}] \cdot \Pr[E_2|E_{\mathsf{fav}}] - \Pr[\bar{E}_{\mathsf{fav}}] \cdot \Pr[E_2|\bar{E}_{\mathsf{fav}}] \\ &= \Pr[E_{\mathsf{fav}}] \cdot (\Pr[E_1|E_{\mathsf{fav}}] - \Pr[E_2|E_{\mathsf{fav}}]) + \Pr[\bar{E}_{\mathsf{fav}}] \cdot (\Pr[E_1|\bar{E}_{\mathsf{fav}}] - \Pr[E_2|\bar{E}_{\mathsf{fav}}]) \\ &= (\epsilon - \frac{\epsilon^2}{4}) \cdot (\Pr[E_1|E_{\mathsf{fav}}] - \Pr[E_2|E_{\mathsf{fav}}]) + \Pr[\bar{E}_{\mathsf{fav}}] \cdot \xi \\ &\leq (\epsilon - \frac{\epsilon^2}{4}) + \xi\end{aligned}$$

Note that if ξ were negligible, then we arrive at a contradiction. We now show that ξ is negligible.

Let GC,GC' denote the garbled circuits that P_i sends to P_j in an execution of $ZKSend_{ij}$ and $ZKSend_{ij}$ respectively. Let $f : D \to R$ be the function that GC computes and let $f' : D \to R$ be the function that GC' computes. We note that if no favorable transcript for phase 1 exists, then $f(x) = f'(x)$ for all $x \in D$. Then, by lemma 7, we conclude that the distribution of GC and the distribution of GC' are indistinguishable. □

Lemma 9 (ZKSend Bounded-Concurrent Simulatability) *Consider the following two security games: In the first game, an adversary $\mathcal{A}$ acts as the sender in a polynomially-bounded number of concurrent real executions of ZKSend with honest receivers. In the second game, adversary $\mathcal{A}$ acts as the sender in a polynomially-bounded number of simulated concurrent executions of ZKSend with a simulator. Then no such PPT adversary $\mathcal{A}$ can distinguish between the views in the two games with advantage greater than ϵ, where ϵ is negligible in the security parameter.*

Proof. Let us first recall the structure of ZKSend. Note that an execution of ZKSend is identical to cZK except that instead of simply sending the last message m_{final} of cWI-UARG, the receiver (prover) sends

covert commitments to random shares of m_{final}, followed by a challenge-response phase and a garbled circuit evaluation. That is, prior to sending the covert commitments to the shares of m_{final}, the communication between P_j and P_i in $ZKSend_{ij}$ is identical to that between P and V in cZK.

Now consider the following scenario. Consider a polynomially bounded number (say k) of concurrent executions of ZKSend where an adversary $\mathcal{A}$ is acting as the sender in all the executions, while the simulator $\mathcal{S}$ is acting as the receiver. At the same time, messages of an outer protocol (more specifically, our covert multi-party computation protocol) may be being exchanged between $\mathcal{A}$ and $\mathcal{S}$, including messages of other executions of ZKSend. $\mathcal{S}$ is internally running the prover algorithm of cZK in order to generate the receiver messages in each of the k ZKSend executions. Note that in case the prover algorithm is using the simulator strategy for cZK, it will require the code of the adversary (controlling the sender) along with short descriptions of all the messages of $\mathcal{S}$ contained in slot 1 or slot 2 (this includes the messages sent by $\mathcal{S}$ in other executions of ZKSend, in particular the long challenge strings r_1, r_2) in order to compute the outgoing messages. We observe that $\mathcal{S}$ already has all this information to input to the prover algorithm. In this manner, $\mathcal{S}$ is able to generate the receiver messages until stage $4(a)$ in all the k executions. In order to generate the receiver messages from stage 4(b) onwards in any execution, $\mathcal{S}$ first prepares the last message m_{final} of cWI-UARG in cZK using the prover algorithm for that execution. $\mathcal{S}$ now executes the rest of ZKSend honestly with $\mathcal{A}$ by using m_{final}. In particular, $\mathcal{S}$ first computes random shares of m_{final} and sends covert commitments to these shares to $\mathcal{A}$. $\mathcal{S}$ then executes a challenge-response protocol with $\mathcal{A}$ and reveals the shares to m_{final} honestly as chosen by $\mathcal{A}$. Finally, $\mathcal{S}$ evaluates the garbled circuit (also executes OT protocols with $\mathcal{A}$) and obtains some output.

Now consider the following two cases. In the first case, the prover algorithm of cZK uses the correct witness to generate the outgoing messages in each of the k executions. Note that this experiment is the same as the first game (see lemma statement). In the second case, the prover algorithm uses the simulator strategy for cZK in order to generate the outgoing messages. Note that this experiment is the same as the second game. Now, if $\mathcal{A}$ can distinguish between the views in the two games with non-negligible advantage ϵ, then $\mathcal{S}$ can distinguish with the same advantage ϵ between the views of the two cases where the prover algorithm of cZK uses the correct witnesses, and when it uses the simulator strategy for cZK. From the zero-knowledge property of cZK, it follows that this is a contradiction. □

Lemma 10 (ZKSend Bounded-Concurrent Simulation Soundness) *Consider a polynomially-bounded number of concurrent executions of ZKSend where the simulator $\mathcal{S}$ might be simulating any number of these executions when acting as the receiver. Let s be the identifier of one of these ZKSend executions where $\mathcal{S}$ is acting as the sender. Further, in this execution, the NP statement that $\mathcal{A}$ is supposed to prove is false. Now consider the following two games: In the first game, $\mathcal{S}$ uses some fixed input value v in the execution s. In the second game, $\mathcal{S}$ uses a random value R (drawn from the uniform distribution) in execution s. Then no PPT adversary $\mathcal{A}$ can distinguish between the distributions of the views in the two games, except with negligible advantage.*

Proof. Informally speaking, we will prove this lemma in the same way as the proof of simulation soundness for cZK in [Pas04]. Assuming that the lemma statement is false, we will rely on the ZKSend Stand-Alone Weak Extractability lemma (Lemma 8) to derive a contradiction. We only give a sketch of the main ideas in the proof.

Let $ZKSend_{ij}$ be the session s where $\mathcal{A}$ is controlling the receiver P_j and $\mathcal{S}$ is acting as the sender P_i. The NP statement that P_j is supposed to prove is false. Now, suppose that $\mathcal{A}$ can distinguish between the distributions of the views in the two games with noticeable advantage. Then we show how to construct a cheating receiver P_j^* for a single instance of $ZKSend_{ij}$ (in the stand-alone setting) such that P_j^* is able to distinguish with noticeable advantage between the distributions of the views when the honest sender uses a fixed input value v and when he uses a random value R (even when the NP statement that P_j^* is supposed to prove is false). This will contradict the ZKSend Stand-Alone Weak Extractability lemma (Lemma 8).

Consider the following experiment. P_j^* works in the same way as $\mathcal{S}$ except that in session s, P_j^* will forward the messages of $\mathcal{A}$ to an external honest sender E. The replies of E are then forwarded by P_j^* back

to $\mathcal{A}$ on behalf of $\mathcal{S}$. Let us assume that P_j^* is able to simulate all the ZKSend executions (where $\mathcal{S}$ is acting as the receiver) correctly even when talking to the external sender E. Now, if E used a fixed input v, then this experiment is the same as the first game (see lemma statement). Otherwise, if E used a random value R as its input, then this experiment is the same as the second game. Now if $\mathcal{A}$ can distinguish between the views of the two games with noticeable advantage, then P_j^* can distinguish with the same noticeable advantage between the views of two stand-alone executions of ZKSend, such that the receiver E uses a fixed input input v and a random value R in the first and second executions respectively. It follows from the *ZKSend Weak Extractability* lemma (Lemma 8) that this is a contradiction. Now all that remains to show is that P_j^* (i.e., the simulator $\mathcal{S}$) can correctly simulate all the instances of ZKSend where it is acting as the receiver.

As in [Pas04], we initially run into a problem with the simulation because the code of the external sender E is not available to the simulator. This means that the straightforward simulation of the concurrent ZKSend sessions (where $\mathcal{S}$ is acting as the receiver) cannot be done as it is since it explicitly requires a 'short description' of the sender messages sent by the simulator (that would include the messages from the external sender E). This problem is resolved by the two-slot simulation technique of Pass. For the sake of completeness, we recall the argument from [Pas04] below.

Let r_1^E, r_2^E denote the (long) challenges sent by E in session s. Note that except these challenges, the simulator does have a description of all the messages sent to $\mathcal{A}$ that is shorter than $\ell(\kappa)$ (recall that $\mathcal{S}$ uses a pseudo-random generator to produce the long challenges in all sessions other than s where it is acting as the sender). Then, all we need to show is that it is sufficient to have a short description of the messages sent in one of the slots of s. Consider any session $s' \neq s$ where $\mathcal{S}$ is acting as the receiver. Now two cases are possible:

- In the first case, both r_1^E and r_2^E are contained in the same slot of s'. In this case, the other slot is free, and can therefore be used to perform the simulation.
- Otherwise, r_1^E and r_2^E are contained in different slots of s'. However, by construction, it follows that either the first or the second challenge in s' is at least $\ell(\kappa)$ bits longer than the corresponding challenge (sent by E) in s. Thus there exists a slot in s' such that even if we include r_i^E in the description, we still have enough bits left to describe the other messages of the simulator, which implies that the simulation can be performed.

Lemma 11 (ZKSend Covertness) *In protocol $ZKSend_{ij}$, when v is random, no PPT machine P_j can distinguish between the messages sent by an honest sender P_i and messages drawn at random from the uniform distribution, with probability $> \epsilon$, where ϵ is negligible in the security parameter κ. Similarly, no PPT machine P_i can can distinguish between the messages sent by an honest receiver P_j and messages drawn at random from the non-participating distribution, with probability $> \epsilon$, where ϵ is negligible in the security parameter κ. In other words, covertness of P_i and P_j are both preserved.*

Proof. Let us consider the two cases when P_i and P_j are malicious respectively.
P_i is malicious. Let us first analyze the messages sent by P_j in an execution of $ZKSend_{ij}$. In phase I, P_j only sends covert commitments and additionally reveals some shares to the last message of cWI-UARG. Further, in Phase II, P_j sends some bits to select values in the covert OT protocol. We will show that P_j's covertness is preserved with the help of a hybrid argument.

In the first hybrid, P_j simply sends a message drawn at random from the uniform distribution for all of his messages in $ZKSend_{ij}$. In the second hybrid, P_j sends covert commitments (using the commitment scheme *Com*) to random strings in phase I and executes the OT protocol of the garbled circuit (in phase II) on random input values. Note that by the covertness property of the commitment scheme *Com* (from Lemma 2), and the covertness preserving property of the OT scheme with respect to the receiver (from Lemma 5), the view of P_i in the second hybrid is indistinguishable from the view in the first hybrid. In the third hybrid, P_j sends covert commitments to the correct values until stage $4(a)$ of $ZKSend_{ij}$ and follows the rest of the protocol as in the second hybrid. Since the commitment scheme *Com* is computationally

hiding, the view of P_i in the third hybrid is indistinguishable from the view in the second hybrid. In the fourth hybrid, P_j runs a mental experiment, where he computes random shares for m_{final} (the last message of cWI-UARG). It follows the protocol exactly as in the third hybrid except that in stage $5(b)$, he reveals the correct shares of m_{final} as selected by P_i. Since hybrids three and four are identical, the views of P_i are identical in both hybrids. In hybrid $five$, P_j sends covert commitments to the random shares of m_{final} in stage $4(b)$ and follows the rest of the protocol as in the fourth hybrid. Since the commitment scheme Com is computationally hiding, the view of P_i in the fifth hybrid is indistinguishable from the view in the fourth hybrid. Finally, in hybrid six, P_j executes the OT protocol on correct input values (and follows the rest of the protocol as in the previous hybrid). By the security property of the OT protocol with respect to the receiver (from Lemma 3), the views of P_i in hybrid five and six are indistinguishable.

Overall, from the indistinguishability of the views of P_i in the first and the last hybrid, we conclude that the view of any PPT adversary P_i remains indistinguishable when it interacts with a P_j who is following the protocol as compared to the case when P_j sends messages drawn at random from the non-participating distribution. Hence, covertness of P_j is preserved.

P_j is malicious. Let us first analyze the messages sent by P_i in an execution of $ZKSend_{ij}$. In phase I, P_i sends the description of a hash function (which is essentially a random string) followed by random strings in stage 2,3,4 and randomly chosen query bits in stage 5. Further, in phase II, P_i sends a garbled circuit that checks the final message of cWI-UARG from P_j. We will show that P_j's covertness is preserved with the help of a hybrid argument. Since P_i only sends random strings in phase I of the protocol, we will only focus on the messages in phase II.

In the first hybrid, P_i simply sends a message drawn at random from the uniform distribution instead of a garbled circuit (including the OT protocol). In the second hybrid, P_i sends a random garbled circuit with random commitments but runs the OT protocol on correct values. Since the garbled circuit used in cZK preserves covertness, and from the covertness preserving property of the OT scheme with respect to the sender (from Lemma 6), it follows that the view of P_j in the second hybrid is indistinguishable from the view in the first hybrid. Finally, in the third hybrid, P_i sends the right garbled circuit and follows the rest of the protocol as in hybrid two. By the security property of garbled circuits (Lemma 7), the view of P_j in the third hybrid is indistinguishable from the view in the second hybrid.

Overall, from the indistinguishability of the views of P_j in the first and the last hybrid, we conclude that the view of any PPT adversary P_j remains indistinguishable when it interacts with a P_i who is following the protocol as compared to the case when P_i sends messages drawn at random from the non-participating distribution. Hence, covertness of P_i is preserved.

Combing the results from the two cases of either P_i or P_j being malicious, we conclude that the $ZKSend_{ij}$ protocol preserves covertness of both P_i and P_j when v is random and hence the claim. Note that if a simulated execution of $ZKSend_{ij}$ is run, covertness is still preserved. □

C.2 Indistinguishability of the Outputs

We now use a hybrid argument to prove the indistinguishability of the output of a real execution of our covert multi-party computation protocol Π from the output of an ideal world execution with the simulator $\mathcal{S}$. We construct a series of hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_{10}$, where $\mathcal{H}_0$ represents the real world execution and $\mathcal{H}_{10}$ represents the ideal world execution. We now explain the hybrids in detail.

Hybrid $\mathcal{H}_0$:

Description. This experiment is exactly the same as the protocol execution in the real world. All inputs and party participation data are available to $\mathcal{S}$. All participating parties take part in the protocol, while non-participating parties output random messages throughout the protocol.

Hybrid $\mathcal{H}_1$:

Description. This experiment is the same as $\mathcal{H}_0$, except that in each execution of $ZKSend_{ij}$ in the Output Exchange phase where the receiver is an honest party $j \in H$, $\mathcal{S}$ simulates $ZKSend_{ij}$.

Proof of output indistinguishability from $\mathcal{H}_0$. It follows from the *ZKSend Bounded-Concurrent Simulatability* lemma (Lemma 9) that the views of the adversary and the output distributions of the honest parties in $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. We remark that in this proof, we only require lemma 9 to hold for the case where the ZKSend executions are performed in *parallel*.

Hybrid $\mathcal{H}_2$:

Description. This experiment is the same as $\mathcal{H}_1$, except that in each execution of $ZKSend_{ij}$ in the Input Commitment phase where the sender is an honest party $i \in H$, $\mathcal{S}$ uses a random value as its input instead of IR_i (the decommitments to $Com(X_i)$ and $Com(t_i)$, where X_i is the input of P_i and t_i is the randomness that P_i uses in the Garbled Circuit Generation phase).

Proof of output indistinguishability from $\mathcal{H}_1$. It follows from the *ZKSend Stand-Alone Weak Extractability* lemma (Lemma 8) that the views of the adversary and the output distributions of the honest parties in $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable, otherwise we can create an inverter for the one-way permutation scheme f_{OWP} used in our construction.

Hybrid $\mathcal{H}_3$:

Description. This experiment is the same as $\mathcal{H}_2$, except that in each execution of $ZKSend_{ij}$ in the Input Commitment phase where the receiver is an honest party $j \in H$, $\mathcal{S}$ simulates $ZKSend_{ij}$.

Note that if a party $i \in M$ behaves honestly during the simulation of the ZKSend executions in the Input Commitment phase, then except with negligible probability, $\mathcal{S}$ is able to extract IR_i (the decommitments to $Com(X_i)$ and $Com(t_i)$). That is, $\mathcal{S}$ obtains the input and randomness (to be used in the Garbled Circuit Generation phase) of $i \in M$.

Proof of output indistinguishability from $\mathcal{H}_2$. It follows from the *ZKSend Bounded-Concurrent Simulatability* lemma (Lemma 9) that the views of the adversary and the output distributions of the honest parties in $\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable.

Hybrid $\mathcal{H}_4$:

Description. This experiment is the same as $\mathcal{H}_3$, except that all commitments made by the honest participating parties in the Input Commitment phase are changed to commitments to random values.

Proof of output indistinguishability from $\mathcal{H}_3$. Note that decommitments to the commitments made in the Input Commitment phase are no longer used anywhere else in the protocol (since all executions of ZKSend where an honest party is the receiver are being simulated by $\mathcal{S}$). From the computational hiding property of the commitment scheme *Com* used in our protocol, it follows that the views of the adversary in $\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable. Also, the output distributions of the honest parties in $\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable.

Hybrid $\mathcal{H}_5$:

Description. This experiment is the same as $\mathcal{H}_4$, except the following. Consider the executions of the ZKSend protocol in the Output Exchange phase. Then, for each pair of parties i, j, where $i \in H$ is the sender with value GC_i^j, $\mathcal{S}$ instead executes $ZKSend_{ij}$ with random value r, if either of the following conditions is true:

1. $\mathcal{S}$ failed to extract IR_i (i.e., the opening of the commitments to the input and randomness) of some malicious party $i \in M$ during the simulation of the *ExtractEnable* protocol.

2. Some party $i \in M$ cheated in the Garbled Circuit Generation phase.

If neither of these conditions is true, then this experiment is identical to $\mathcal{H}_5$.

Proof of output indistinguishability from $\mathcal{H}_4$. Note that if either of the above mentioned conditions is true, then except with negligible probability, there exists at least one party $j \in M$ that was dishonest in the protocol execution prior to the Output Exchange phase. In this case, the NP statement that asserts P_j's honest behavior in an execution of $ZKSend_{ij}$ will be false. Then, it follows from the *ZKSend Bounded-Concurrent Simulation Soundess* lemma (Lemma 10) that the views of the adversary and the output distributions of the honest parties in $\mathcal{H}_4$ and $\mathcal{H}_5$ are indistinguishable.

Hybrid $\mathcal{H}_6$:

Description. This hybrid is exactly the same as $\mathcal{H}_5$ except that if the extraction of IR_i of all malicious parties was successful during the simulation of *ExtractEnable*, and if all parties were honest in the Garbled Circuit Generation phase, then $\mathcal{S}$ sends the extracted inputs of all the malicious parties to the trusted party in the ideal world. Let $x_1, \ldots, x_n$ denote all the inputs received by the trusted party (including the inputs of the honest parties in the ideal world). Then the Ideal functionality returns the value $f(x_1, \ldots, x_n)$ if $g(x_1, \ldots, x_n) = 1$, otherwise it returns $\perp$.

Now $\mathcal{S}$ constructs a simulated garbled circuit (as explained earlier in the description of the simulator). If the Ideal functionality returned $f(x_1, \ldots, x_n)$, then the output of this garbled circuit is the value $f(x_1, \ldots, x_n), K_1, \ldots, K_n$ (where $K_1, \ldots, K_n$) are the keys chosen by all parties in the input commitment phase). For all $i \in M$, $\mathcal{S}$ uses K_i from IR_i. Otherwise, if the ideal functionality returned $\perp$, then the output of this garbled circuit is a random string R. Let GC be the simulated garbled circuit. $\mathcal{S}$ then changes the values of garbled circuit shares GC_i of honest parties in such a way that $(\bigoplus_{i \in H} GC_i) \oplus (\bigoplus_{i \in M} GC_i) = GC$. Note that, since $\mathcal{S}$ knows the garbled circuit shares GC_i of malicious parties, it picks shares at random for honest parties such that the above conditions hold.

Proof of output indistinguishability from $\mathcal{H}_5$. It follows from the *Garbled Circuit Security* lemma (Lemma 7) that the views of the adversary and the outputs of the honest parties in $\mathcal{H}_5$ and $\mathcal{H}_6$ are indistinguishable.

Hybrid $\mathcal{H}_7$:

Description. This hybrid is exactly the same as $\mathcal{H}_6$ except the following. Consider the execution of the covert-GMW protocol in the Garbled Circuit Generation phase. Then, in all executions of the covert 1-out-of-4 OT protocol where an honest party $i \in H$ is the sender, P_i runs the OT protocol on random input values (instead of the correct values as per the covert-GMW protocol). On the other hand, in all executions of covert 1-out-of-4 protocol where an honest party $i \in H$ is the receiver, P_i chooses to receive one of the four values uniformly at random.

Proof of output indistinguishability from $\mathcal{H}_6$. Let us first consider an execution of the covert 1-out-of-4 OT protocol where an honest party $i \in H$ is the receiver. In this case, P_i chooses to receive one of the four values uniformly at random. Note that this randomness is used no where else in the protocol (neither before this OT protocol, nor afterwards). Then, it follows from the *OT Receiver* lemma (Lemma 3) that the view of any PPT adversary in this case is indistinguishable from its view when P_i chooses one of the four values as per the covert-GMW protocol.

Now consider the case where an honest party $i \in H$ is the sender in an execution of the covert 1-out-of-4 OT protocol. More specifically, consider an execution of the OT protocol between an honest party P_i and some P_j for an AND gate of the GMW protocol. P_i, P_j hold shares c_i, c_j (respectively) of the first input bit

and shares d_i, d_j (respectively) of the second input bit to the AND gate. At the end of the OT protocol, the receiver P_j must hold $b_{j,i}$ such that $b_{i,j} + b_{j,i} = c_i d_j + c_j d_i$ (where $b_{i,j}$ is selected at random by sender P_i). The randomness used by the sender to select $b_{i,j}$, is never used anywhere else in the protocol (neither before this OT protocol, nor afterwards). Although this single bit determines the four values from which the receiver must choose one, each of the four values are individually random. Also, as mentioned earlier, P_i chooses these four values uniformly at random (instead of the correct values as per the covert-GMW protocol). Then, it follows from the *OT Sender* lemma (Lemma 4) that the view of even an infinitely powerful adversary is indistinguishable from its view when P_i selects the four values as per the covert-GMW protocol.

Note that the above arguments hold for *all* OT protocol executions in the covert-GMW protocol. Then, by a standard hybrid argument, it follows that the views of the adversary in $\mathcal{H}_6$ and $\mathcal{H}_7$ are indistinguishable. Also, the output distributions of honest parties in $\mathcal{H}_6$ and in $\mathcal{H}_7$ are indistinguishable.

Note that at this point, $\mathcal{S}$ no longer makes use of inputs of honest participating parties in the protocol, but still makes use of party participation data. We shall show in the following hybrids, how to remove the dependence on party participation data.

Hybrid $\mathcal{H}_8$:

Description. This hybrid is exactly the same as $\mathcal{H}_7$ except that non-participating parties send commitments to random values in the Input Commitment phase and take part in OT protocol executions in the covert-GMW protocol (in the Garbled Circuit Generation phase) with random values (instead of simply sending random values throughout the protoocol).

Proof of indistinguishability from $\mathcal{H}_7$. First, let us only consider the case where the non-participating parties send covert commitments in the Input Commitment phase. Then, by a standard hybrid argument, it follows from the *Covert Commit* lemma (Lemma 2) that the view of any PPT adversary in this case must be indistinguishable from its view when the non-participating parties simply send random strings in the Input Commitment Phase. Now, consider the case where the non-participating parties participate as a sender in the OT protocol executions with random input values. Then, by a standard hybrid argument, it follows from the *Covert OT Sender* lemma (Lemma 6) that the view of any PPT adversarial receiver in this case must be indistinguishable from its view when the non-participating parties simply send random strings during the OT protocol. Now consider the opposite case where the non-participating parties participate as a receiver in the OT protocol executions and select values uniformly at random. Then, by a standard hybrid argument, it follows from the *Covert OT Receiver* lemma (Lemma 5) that the view of any PPT adversarial sender in this case must be indistinguishable from its view when the non-participating parties simply send random strings during the OT protocol executions.

Combining all the above arguments, it follows that the view of the adversary in $\mathcal{H}_7$ is indistinguishable from its view in $\mathcal{H}_8$. Also, the outputs of all the honest parties are identical in $\mathcal{H}_7$ and $\mathcal{H}_8$.

Hybrid $\mathcal{H}_9$:

Description. This hybrid is exactly the same as $\mathcal{H}_8$ except that non-participating parties participate in executions of ZKSend in the Input Commitment phase as well as the Output Exchange phase. When a non-participating party P_i is a sender in an execution of $ZKSend_{ij}$, it uses a random value as its input. On the other hand, when a non-participating party P_j is a receiver in an execution of $ZKSend_{ij}$, it uses $\mathcal{S}_{\text{cZK}}$ to simulate $ZKSend_{ij}$ (by using the same strategy as explained earlier in the description of $\mathcal{S}$).

Proof of indistinguishability from $\mathcal{H}_8$. The indistinguishability of the adversary's view and the outputs of the honest parties in $\mathcal{H}_8$ and $\mathcal{H}_9$ follows immediately from the *ZKSend Covertness* lemma (Lemma 11) (which holds even for simulated executions of $ZKSend_{ij}$).

Hybrid $\mathcal{H}_{10}$:

Description. This hybrid is exactly the same as $\mathcal{H}_9$ except that if the output received from the trusted party was $\perp$, then $\mathcal{S}$ sets the garbled circuit shares of each non-participating party to be a random value and broadcasts a covert commitment to this random value (instead of a random string) at the end of the Garbled Circuit Generation phase. The non-participating parties further participate in the Output Exchange phase with this random value as their garbled circuit share. Note that $\mathcal{H}_{10}$ is in fact identical to the ideal world execution with $\mathcal{S}$.

Proof of indistinguishability from $\mathcal{H}_9$. First note that if the output received from the trusted party was $f(x_1, x_2, ..., x_n)$, then all parties participated in the protocol and hence there are no non-participating parties. In this case, $\mathcal{H}_{10}$ is identical to $\mathcal{H}_7$. On the other hand, if the output received from the trusted party was $\perp$, then possibly some parties did not participate in the protocol execution. In this case, the garbled circuit shares of all non-participating parties are set to random values and $\mathcal{S}$ broadcasts covert commitments to these random values at the end of the Garbled Circuit Generation phase. The non-participating parties then participate in the Output Exchange phase using these random values as their garbled circuit shares. Note that this is similar to hybrid $\mathcal{H}_6$, where all shares of participating parties are set to random values when the output received from the trusted party is $\perp$. Therefore, it follows that the views of the adversary and the output distributions of the honest parties in $\mathcal{H}_9$ and $\mathcal{H}_{10}$ are indistinguishable. □

D Short Description of Our Results

In this section, we give a (relatively) short and largely informal description of all our results. This mainly contains text from the proceedings version of the paper. This is intended for a reader who only wishes to focus on the key ideas in our results.

D.1 Impossibility of Constant Round Covert Computation with Black Box Simulation

In this section, we show the existence of a PPT computable covert two-party functionality for which there does not exist any constant-round covert computation protocol with respect to a black-box simulator. Our impossibility result rules out any expected polynomial time simulator which uses the adversarial algorithm as an oracle.

Let us first consider any two-party covert functionality F and assume that there exists a *constant*-round covert computation protocol Π that securely realizes F with respect to a black-box simulator. We first construct a real world adversary for Π and derive a lower bound on the probability p_{fail} with which every black-box simulator for Π gets full participation from the adversary in the "main thread" (where main thread is defined as the execution thread output by the simulator in its view), but fails to get any participation in all other threads of execution. In other words p_{fail} is the probability with which the simulator is essentially "straight-line". As we will show later in the proof, p_{fail} is noticeable (in the security parameter) if the functionality F is such that an adversary in a real world execution of Π can distinguish whether or not an honest party is participating in the protocol only with negligible probability. This is indeed true for several covert functionalities. In the second part of the proof, we give an example of such a functionality and finally derive a contradiction based on the fact (established in the first part) that the black-box simulator for any constant-round protocol for this functionality is "straight-line" with noticeable probability.

Here we only give a sketch of the main ideas used to derive the lower bound on the probability p_{fail} (as described above). We refer the reader to section 3 for complete details.

Let F be any covert functionality for two parties (P_1, P_2). Let Π be any constant-round covert computation protocol that securely realizes F with respect to a black-box simulator. Without loss of generality, we assume that P_2 sends the first message in Π. Let $\{\mathcal{T}_k\}_k$ be a family of q-wise independent predicates, where $t \in \mathcal{T}_k$ maps $\{0,1\}^{\leq \text{poly}(\kappa)}$ to $\{0,1\}$ such that on any randomly chosen valid input β, t outputs 1 with

probability $1/q^2$.[8] Here q is a parameter polynomial in the security parameter (to be determined later). Suppose that the adversary chooses a predicate t from the q-wise independent family $\mathcal{T}_k$. Now consider an execution between an adversary and the other party (which may be emulated by the simulator). We will say that a query β made by the other party (resp. simulator) to the adversary is favorable if t outputs 1 on β *as well as* on every prior query in that execution (resp. execution thread).

Adversary P_1^*. We first describe our adversary P_1^*. We assume that P_1^* has the required input and random tape as would an honest party. Then P_1^* works as follows. P_1^* first chooses a predicate t from the q-wise independent family $\mathcal{T}_k$. Now consider any round in the execution of Π. If the query from P_2 is favorable, then P_1^* sends an honest reply (as it would if it were participating honestly in the protocol); otherwise it sends a message drawn from the uniform distribution[9].

Since the number of rounds in Π is a constant (say) c, it follows that P_1^* participates in Π with a noticeable probability ($= (1/q^2)^c$). Let $\mathcal{S}$ be a black-box simulator for protocol Π. We now use a hybrid argument to derive a lower bound on the probability p_{fail}, defined as follows. Consider the view output by $\mathcal{S}$ at the end of its interaction with P_1^*. As mentioned earlier, we will refer to the thread of execution in this view as the "main thread". Then, informally speaking, p_{fail} is the probability that $\mathcal{S}$ finds full participation from P_1^* in the main thread (i.e., the predicate t chosen by P_1^* returns 1 on every query from $\mathcal{S}$ in the main thread), but fails to find any participation from P_1^* in any other execution thread (i.e., the predicate t chosen by P_1^* returns 0 on every query from $\mathcal{S}$ in any thread other than the main thread).

We now construct a series of hybrids $\mathcal{H}_i$, $i \in [0, 3]$, where each hybrid represents the interaction between an adversarial P_1 (referred to as $\mathcal{A}$) with a strategy we define and the simulator $\mathcal{S}$. The adversary $\mathcal{A}$ in $\mathcal{H}_3$ is identical to P_1^*. In each hybrid, we define and analyze the winning probability of $\mathcal{A}$ such that p_{fail} is the probability with which $\mathcal{A}$ wins in $\mathcal{H}_3$.

Hybrid $\mathcal{H}_0$. In this experiment, $\mathcal{A}$ simply sends a message drawn from the uniform distribution on receiving any query from $\mathcal{S}$. Let q be the median of the number of queries that $\mathcal{S}$ makes. That is, with probability $1/2$ (where probability is taken over all the coins of $\mathcal{S}$ and $\mathcal{A}$), $\mathcal{S}$ makes at most q queries. Note that $q = \text{poly}(\kappa)$. We say that $\mathcal{A}$ wins in this experiment if every query from $\mathcal{S}$ in the main thread (defined by the view output by $\mathcal{S}$) is favorable, but every other query is *not* favorable. Looking ahead, if $\mathcal{A}$ were to reply honestly on each favorable query (as it does in $\mathcal{H}_3$), then the winning probability of $\mathcal{A}$ would be identical to p_{fail}.

We analyze the winning probability of $\mathcal{A}$ over the choices of t for a fixed execution defined by a random tape of $\mathcal{S}$ and a random tape that $\mathcal{A}$ uses to draw messages from the uniform distribution. Consider the event that every query from $\mathcal{S}$ in the main thread is favorable (i.e., t outputs 1 on every simulator query in the main thread). Now consider the event that every query from $\mathcal{S}$ that is not in the main thread is not favorable (i.e., t outputs 0 on any query not in the main thread). We observe that these two events are independent conditioned on $\mathcal{S}$ making at most q queries (since in this case t essentially behaves as a random function and the queries made by $\mathcal{S}$ and its view in general are independent of the choice of t). Then, by a simple probability analysis, we show that $\Pr[\mathcal{A} \text{ wins}] \geq \frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q})$, which is noticeable in the security parameter.

Hybrid $\mathcal{H}_1$. Same as $\mathcal{H}_0$ except the following. On receiving any favorable query β from $\mathcal{S}$, $\mathcal{A}$ checks if it had earlier received a favorable query $\beta' \neq \beta$ in a "different" thread (in the sequel, we will refer to this check as the *stopping condition*). If the check succeeds, then $\mathcal{A}$ stops the experiment and we say that $\mathcal{S}$ wins. Otherwise, the winning criterion for $\mathcal{A}$ in this experiment is defined exactly as in $\mathcal{H}_0$.

Intuitively, the only difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that in $\mathcal{H}_1$, whenever $\mathcal{S}$ is successful in making a favorable query in more than one execution thread, $\mathcal{A}$ stops the experiment and declares $\mathcal{S}$ to be the winner. However, in this case, $\mathcal{S}$ would have won in $\mathcal{H}_0$ as well. Then, it can be shown that the winning probability of $\mathcal{A}$ in $\mathcal{H}_1$ is identical to that in $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. Same as $\mathcal{H}_1$, except that on receiving any favorable query from $\mathcal{S}$, if the *stopping condition* is false, $\mathcal{A}$ sends an honest reply (as it would if it were participating honestly) to $\mathcal{S}$ (instead of sending a

[8] Such a family can be easily constructed from a family of q-wise independent hash functions.

[9] Note that the check for a query being favorable implicitly ensures that P_1^* sends an honest reply only if it had not already stopped participating in the protocol in an earlier round.

message drawn from the uniform distribution). However, if the *stopping condition* is true, it continues to stop the experiment as in $\mathcal{H}_1$. The winning criterion for $\mathcal{A}$ in this experiment is defined exactly as in $\mathcal{H}_1$.

The key property of the interaction between $\mathcal{A}$ and $\mathcal{S}$ in this experiment is that $\mathcal{A}$ participates honestly only in at most a single thread of execution (since $\mathcal{A}$ stops experiment just before this property stops being true). Now, in order to bound the winning probability of $\mathcal{A}$ from below, we consider an experiment where $\mathcal{A}$ "exposes" such a thread of its interaction (if it exists) to an external party P_1. More specifically, $\mathcal{A}$ forwards each favorable query in this thread to P_1 and then forwards back its response to $\mathcal{S}$. Now, note that if all the replies of P_1 were drawn from the uniform distribution, then this experiment is identical $\mathcal{H}_1$, otherwise if P_1 replied honestly to each query, then it is identical to $\mathcal{H}_2$. Let ϵ be the probability with which a PPT machine can distinguish between these two cases. Then, the winning probability of $\mathcal{A}$ in $\mathcal{H}_2$ must be at least $\frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q}) - \epsilon$.

Looking ahead, the probability ϵ would be negligible for several functionalities in keeping with the covertness property of the protocol Π. Hence, the winning probability of $\mathcal{A}$ in this experiment would still be noticeable.

Hybrid $\mathcal{H}_3$. Same as $\mathcal{H}_2$ except that on receiving any favorable query from $\mathcal{S}$, $\mathcal{A}$ sends an honest reply (as it would if it were participating honestly) even if the *stopping condition* is true (as opposed to stopping the experiment). We observe that by definition, $\mathcal{A}$ in this experiment is identical to P_1^*. The winning criterion for $\mathcal{A}$ in this experiment is defined exactly as in $\mathcal{H}_2$.

Then, the winning probability of $\mathcal{A}$ in $\mathcal{H}_3$ is identical to that in $\mathcal{H}_2$; intuitively, this is because the only difference between $\mathcal{H}_2$ and $\mathcal{H}_3$ is that for some choices of the random tapes of $\mathcal{S}$ and $\mathcal{A}$, $\mathcal{A}$ might stop the experiment in $\mathcal{H}_2$ (and hence lose); however, note that for all these random tapes, $\mathcal{A}$ will lose in $\mathcal{H}_3$ as well. Hence, we have that $\mathsf{p}_{fail} = \Pr[\mathcal{A} \text{ wins}] \geq \frac{1}{2q^{2c}} \cdot (1 - \frac{1}{q}) - \epsilon$.

Completing the proof. In the second part of the proof, we give an example of a functionality for which the probability ϵ is negligible. Hence the probability with which the simulator is "essentially" straight line (that is, p_{fail}) is noticeable. Finally we show that for our functionality, this contradicts the security of the protocol Π.

D.2 Constant-Round Covert Multi-party Computation

At a high level, our constant-round covert computation protocol can be seen as the result of a two step process: (a) First, construct a *constant*-round semi-honest covert computation protocol adopting techniques from the work of Beaver et al [BMR90]. (b) Next, the semi-honest protocol is "compiled" with a gadget known as *zero knowledge proofs to garbled circuits* in order to guarantee security against malicious adversaries. Here we adopt some techniques from Chandran et al [CGOS07] to our setting. In the subsection below, we first discuss the notion of zero-knowledge proof to garbled circuit as introduced by Chandran et al [CGOS07], and then give a constant-round construction for the same with some additional security properties (that are necessary when using this gadget in the constant-round setting). Later, we will use our construction of constant-round zero knowledge proof to garbled circuit in presenting our constant-round covert computation protocol.

D.2.1 Zero Knowledge Proofs to Garbled Circuits

Zero Knowledge proofs have been established as a basic building block for constructing multi-party computation protocols secure against active adversaries. However, in the setting of covert computation, this technique does *not* work because if one party "verifies" that another party is executing the protocol honestly, then covertness is immediately compromised. To this end, Chandran et al [CGOS07] introduced the notion of zero knowledge proofs to garbled circuits, where a party gives a proof of its honest behavior to a *garbled circuit* prepared by another party. More specifically, consider two parties (sender, receiver) who share a common input (x, L). The sender wishes to give the receiver a private value v, only if $x \in L$ and the receiver has a valid witness (for $x \in L$). Chandran et al [CGOS07] gave a protocol for this setting based on Blum's 3-round (public-coin) ZK proof for Graph Hamiltonicity. In their protocol, the parties first exchange

the first two messages of Blum's protocol. Then the sender (verifier) prepares and sends a garbled circuit to the receiver (prover); this garbled circuit takes as input the last prover message and outputs v if the verification is successful, else it outputs a random value. Since Blum's protocol is a zero knowledge *proof* with soundness 1/2, if the theorem is false, there does not exist (with probability 1/2) a "correct" last prover message. [CGOS07] also show how to improve the soundness of this basic protocol.

As implied by the results in the previous section, non black-box techniques are necessary to construct such a gadget (henceforth referred to as ZKSend) in the constant-round setting. To this end, we use the non black-box simulation technique of Pass [Pas04] (which in turn builds on the work of Barak [Bar01]). Fortunately, in the zero knowledge protocol of Pass [Pas04], except for the last message, the prover only sends commitments and the verifier only sends random strings. Then, using the same idea as above, we can modify Pass' protocol such that the receiver (prover) sends the last message to a garbled circuit prepared by the sender (verifier), and receives an output value depending upon whether or not the verification was successful.

However, this naive attempt fails since Pass' protocol is an *argument* system. In particular, even if the receiver (prover) was dishonest, a satisfying last message might exist which, very informally speaking, allows the dishonest receiver (prover) to get the sender's input value out of the garbled circuit (even though the receiver does not have such a message explicitly). Then, to be able to reduce the security of ZKSend in such a case to the soundness of Pass' protocol, it seems that the garbled circuit evaluation sub-protocol would need to be able to support extraction of the inputs. However, as implied by the impossibility result in the previous section, very informally speaking, such a sub-protocol cannot work in a constant number of rounds (using a black-box extractor).

To solve the above problem, we observe that such an extraction of the input will not be required by the simulator of our final covert computation protocol constructed using ZKSend as a building block. Instead, such an extraction would only be required to prove a separate lemma that reduces the following security property of ZKSend to the soundness of Pass' protocol: assuming that the statement is false, the view of a cheating receiver (prover) must be indistinguishable across the two cases where an honest sender (verifier) uses a fixed input value in the ZKSend execution in the first case and a random value as its input in the ZKSend execution in the second case. (Looking ahead, such a lemma would be used in the hybrid experiments to prove the indistinguishability of the simulated view from the view in the real protocol execution.) Hence, the extraction procedure is only required to work with a noticeable probability (as opposed to overwhelming probability). This is because of the following. Say we can extract the input to the garbled circuit (which is the last prover message in Pass' protocol) with a noticeable probability. Then we can use that message to violate the soundness of the protocol of Pass with a noticeable probability.

Using these ideas, we now describe our protocol $ZKSend_{ij}$, where P_i is the sender (verifier) and P_j is the receiver (prover). P_i and P_j share a common input (x_j, L_j). P_i additionally has a private input v. Let $\ell(\kappa)$ be the length parameter and $\mathcal{H}_\kappa$ be a family of collision resistant hash functions. Let cWI-UARG be the 5-round witness indistinguishable universal argument (WI-UARG) of Barak and Goldreich [BG02] instantiated with the commitment scheme Com. The protocol $ZKSend_{ij}$ proceeds in the following steps.

Stage 1 **(Setup):** P_i sends $h \stackrel{\mathrm{R}}{\leftarrow} \mathcal{H}_\kappa$ to P_j.

Stage 2 **(Slot 1):** P_j sends a covert commitment $c_1 = Com(0^\kappa)$ to P_i, who responds by sending the first challenge $r_1 \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^{j\ell(\kappa)}$.

Stage 3 **(Slot 2):** P_j sends a covert commitment $c_2 = Com(0^\kappa)$ to P_i, who responds by sending the second challenge $r_2 \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^{(n+1-j)\ell(\kappa)}$.

Stage 4 **(Main Proof Body):**

1. P_i and P_j exchange (only) the first 4 messages of a 5-round cWI-UARG where P_j proves the following statement: either $x_j \in L_j$ or there exists a decommitment to c_1 (resp. c_2) to a program Π such that Π takes as input c_1 (resp. c_2) and a string y of length $j\ell(\kappa) - \kappa$ (resp. $(n+1-j)\ell(\kappa) - \kappa$), and outputs r_1 (resp. r_2). Let m_{final} denote the final message of this cWI-UARG.

2. Let $k = \omega(\log(\kappa))$. P_j chooses k pairs of random shares of m_{final} and sends covert commitments to these shares to P_i.

Stage 5 **(Challenge-Response):** Now, P_i and P_j engage in a 2-round challenge-response protocol, where P_i randomly selects one share from each pair of random shares of m_{final} and P_j reveals the shares selected by P_i. Note that P_j does not send openings to the commitments, but simply the shares selected by P_i.

Stage 6 **(Garbled Circuit):** Finally, P_i sends a covert garbled circuit, $Gar[i \to j]$, with the following description:
1. It takes as input, decommitments to the the random shares of m_{final}. This input is provided by P_j. It then checks whether the decommitments to the shares are *correct*, each pair of shares are shares of the same string, and that correct shares were revealed in stage 5.
2. If either of the above checks fail, then it simply outputs a uniformly chosen random number. Otherwise, it runs the final step of the honest verifier algorithm V for cZK on m_{final}. If the output is accept, then it outputs v, else it outputs a uniformly chosen random number.
3. P_j executes covert 1-out-of-2 OT with P_i as necessary in order to evaluate the garbled circuit $Gar[i \to j]$ and obtain an output.

We prove the following four security properties of our ZKSend construct.

1. Consider a ZKSend execution between a sender (verifier) and a receiver (prover) who share a common input (x, L). The sender's input in the protocol is either a fixed value v or a random value. Then, we show that unless the receiver (prover) "knows a witness for $x \in L$", it cannot distinguish between the two cases where the sender (verifier) is using v in the first case and a random value as its input in the second case. As noted earlier, we prove this security property by reducing it to the soundness of Pass' protocol.

2. Next, we show that an adversarial sender (verifier) who is running a polynomially-bounded number of concurrent executions of ZKSend cannot distinguish whether it is "interacting" with honest receivers (provers) or the simulator. Our simulator relies on the simulator of Pass' protocol to "simulate" the ZKSend executions; as such, we prove this security property by reducing it to the zero knowledge property of Pass' protocol. We stress that even though our covert computation protocol (see Section 5) consists of only *parallel* executions of ZKSend, we consider the more general setting of *bounded-concurrency* as it proves to be useful in the construction of a covert computation protocol over *point-to-point* channels.

3. Now consider a polynomially-bounded number of concurrent executions of ZKSend between an adversary and the simulator, where the adversary is acting as the sender (verifier) in some "left" executions (which are being "simulated" by the simulator) and as the receiver (prover) in a "right" execution with a false theorem as the common input. Then, we show that any such adversary cannot distinguish whether the simulator uses a fixed input value v or a random value as its input in the "right" execution. We stress that we cannot reduce this security property into the simulation soundess of Pass' protocol for the following reasons. Such a proof would require us to construct an adversary who proves a false theorem in Pass' protocol, which in turn, requires rewinding our adversary in the "right" execution (to extract the last message of Pass' protocol). However in this case, very informally speaking, the simulator who is "simulating" the "left" executions, may also get rewound and hence no longer work. Solving this problem requires going into the details of the simulation soundness proof technique of Pass to prove this security property *directly*. In particular, we show that the two slot simulation technique of Pass is powerful enough to handle this scenario as well.

4. Finally, we show that our ZKSend protocol preserves the covertness of both the sender and the receiver. In particular, we show that messages of a sender (resp. receiver) are indistinguishable from random to a receiver (resp. sender).

D.2.2 Our Protocol

Let $P_1, \ldots, P_n$ be n parties that hold inputs $x_1, \ldots, x_n$ respectively. Let $F = (f, g)$ be a covert functionality that they wish to compute. F outputs $f(x_1, \ldots, x_n)$ if the function output is favorable to all parties, else it outputs $\perp$. Here $g(\cdot)$ is the function that determines whether the function output is favorable

$(g(x_1, \ldots, x_n) = 1)$ or not $(g(x_1, \ldots, x_n) = 0)$. We now give the construction of a constant-round covert multi-party computation protocol that securely realizes F.

Overview. At a high level, our protocol consists of three main phases: (a) *Input Commitment phase*: In this phase, all parties commit to their inputs and randomness (note that we do not require coin flipping in our protocol) and run an "extract enable" phase with each other. Intuitively, the purpose of this phase is to enable the simulator to extract the input and randomness of malicious parties during the simulation. (b) *Garbled Circuit Generation phase*: In this phase, the parties run the covert version of the GMW protocol to jointly construct a covert garbled circuit that evaluates the appropriate function (that the parties wish to compute). Each party only obtains an individual share of the garbled circuit as the output of covert-GMW protocol. (c) *Output Exchange phase*: In this phase, parties exchange their garbled circuit shares with each other if and only if all the parties behaved honestly till the end of the previous phase. By incorporating some validity checks in this phase, we are able to ensure output correctness (without compromising covertness).

Intuitively, the garbled circuit generation phase can be viewed as a semi-honest covert computation protocol. Here we adopt techniques from [BMR90] to ensure that the protocol is constant-round. Then, by adding the other two phases (adopting techniques from Chandran et al [CGOS07]), we essentially "compile" the semi-honest protocol (with zero-knowledge proofs to garbled circuits, as will be evident from the description of each phase) to obtain security against active adversaries. In each phase, the key challenge is to ensure that the number of rounds are only a constant. For lack of space, below we only highlight the main ideas in our protocol.

Input Commitment phase. In the Input Commitment phase, all parties commit to their inputs and randomness and run an "extract enable" sub-phase. Intuitively, the purpose of the ExtractEnable sub-phase is to allow the simulator to extract the input and randomness (for the garbled circuit generation phase) of the malicious parties. We note that Chandran et al [CGOS07] used a challenge-response phase of linear round-complexity to enable black-box extraction. However, our impossibility result from the previous section rules out such an approach in the constant-round setting. To this end, we use our construction of zero-knowledge proof to garbled circuit (as described in the last subsection) to enable extraction, as described below.

The main idea of the ExtractEnable sub-phase is as follows. Let the value we would like to enable the simulator to extract be V_i for party P_i. Each party P_i executes the following sub-protocol with each party P_j, in *parallel*.

1) P_i picks a random string r and computes $y = f_{owp}(r)$, where f_{owp} is a one-way permutation. It sends y to P_j.

2) P_j and P_i now engage in an execution of $ZKSend_{ij}$, where P_i is the sender with input V_i and P_j is the receiver, on the following NP statement as the common input: "$\exists r$ such that $y = f_{owp}(r)$" (for a specific witness relation such that any witness for this NP statement contains such an r). Note that an honest P_j will not have a witness for this NP statement and is instructed to simply send random strings during the execution of $ZKSend_{ij}$.

Note that the malicious parties may deviate from the protocol specification (for example, by sending an incorrect garbled circuit inside ZKSend), which may result in the simulator failing to extract. Looking ahead, the later stages will force the output of *all* the parties to be $\perp$ in this case.

Garbled Circuit Generation phase. In this phase, the parties run an instance of the covert-GMW protocol (see section 4) to jointly construct a covert garbled circuit that evaluates the appropriate function, such that each party only obtains a single share of this garbled circuit. The key challenge in this phase is to use only a constant number of rounds.

In many works, constant round multi-party computation protocols have been designed using techniques from Beaver et al [BMR90]. The basic idea is to have the parties run a secure computation protocol (like GMW) to jointly generate a garbled circuit (which they can evaluate on their own later on). To keep the protocol constant round, the encryption scheme used in [BMR90] to generate the gate tables is the simple XOR function. Further, to avoid blowing up the size of the wire keys exponentially, the parties run a preprocessing phase locally in which (among other things) they expand their wire keys using a pseudorandom

generator (PRG). Unfortunately, such a preprocessing phase fails in the setting of covert computation. This is because the PRG evaluations done locally by a party during garbled circuit generation will have to be performed again (locally) while evaluating the resulting garbled circuit. Thus, the fact that the computation done in the preprocessing phase "conforms" to the computation done while evaluating the received garbled circuit leaks the fact that all parties are participating in the protocol (even if the output is not favorable).

To solve this problem, we eliminate the preprocessing phase and move the required cryptographic operations (involved in generating the gate tables) into the circuit of the underlying secure computation protocol. However this presents the following problem. If we use GMW (or a similar protocol like covert-GMW) as the underlying secure computation protocol, the number of rounds required will be linear in the depth of the circuit (which generates a gate table) being evaluated. Thus, any cryptographic operations done by this circuit should be in NC^0 for our protocol to be constant rounds.

Towards that end, to still keep the secure computation protocol constant round, we construct an encryption scheme in NC^0 using techniques from Applebaum et al [AIK04]. We would require that the encryption keys and the ciphertexts produced by such an encryption scheme are indistinguishable from the uniform distribution. We construct such an encryption scheme based on standard assumptions.[10][11] Forgetting our goal of obtaining a covert computation protocol, consider the protocol in which we use such an encryption scheme in gate tables and use standard GMW for gate table generation. We believe this protocol is of independent interest since it also gives a arguably cleaner alternative to the Beaver et al [BMR90] protocol (which in turn has been used widely in the study of round complexity of secure computation).

Then, in this phase, the parties engage in an execution of the covert-GMW protocol that takes as input the shares of the wire keys (and other relevant information). The encryption scheme used in the construction of gate tables is the covert encryption scheme in NC^0. At the end of protocol, each party P_i will hold only a share GC_i of the garbled circuit GC and other necessary information required to evaluate GC (but not GC itself).

Discussion about Covert-GMW. The covert-GMW protocol [CGOS07] is similar to the semi-honest GMW protocol, except that it uses a specific covert secure 1-out-of-4 OT rather than a semi-honest 1-out-of-4 OT, and does not consist of the output broadcast phase. The natural question is, what guarantees could it possibly provide when the parties might deviate from the protocol executions arbitrarily?

Intuitively, the malicious parties (even if they deviate from the protocol arbitrary) do not learn any information (in a computational sense) in covert GMW protocol because of the following. The covert-GMW protocol consists of only two kinds of message: one where parties broadcast $(n-1)$ random shares of their input to other parties, and the second where parties engage in a covert 1-out-of-4 OT protocol with others. The first type of message is simply a random string, therefore it does not reveal any information about GC. In the second type of messages, a party gives away only one of the 4 bits (when acting as a sender in a covert 1-out-of-4 OT protocol). However, all the four bits are *individually* indistinguishable from random. This is because while preparing these four bits, a single bit of randomness is used [GMW87] (making the four bits *individually* random).

Output Exchange phase. In the previous two stages, no information about the output (or participation) was revealed to any party (even if they deviate arbitrarily from the protocol). In other words, the messages exchanged between the parties in the previous stages were "not valuable" for deducing any potentially unknown information. In this phase, the parties actually exchange valuable messages having information so as to be able to get the function output at the end. Specifically, the parties exchange their garbled circuit shares *conditioned* upon the fact that *every* party was honest till the end of the Garbled Circuit Generation phase. This is done using our ZKSend construct as described below.

If any party deviated from the protocol during the Input commitment or Garbled Circuit Generation

[10] Applebaum et al [AIK04] show that there do not exist any encryption schemes such that the decryption function is in NC^0. However, fortunately, for our purpose, we only require the encryption function to be in NC^0.

[11] The option of simply moving the pre-processing phase of [BMR90] to the secure computation protocol circuit would require that circuit to evaluate a PRG. We however, note that a PRG with appropriate stretch exists in NC^0 based only on non-standard assumptions [AIK08].

phase, it could be potentially dangerous for P_i to give out its garbled circuit share (since then the garbled circuit is not guaranteed to be correctly constructed). Hence, P_i breaks its share GC_i further into n sub-shares $\{GC_i^j\}_{j=1}^n$ and transfers these sub-shares in *parallel* to other parties P_j using $ZKSend_{ij}$. In particular, in an execution of $ZKSend_{ij}$, the input of P_i is GC_i^j while P_j is supposed to prove an NP statement which asserts that P_j was "honest" in the protocol up to the end of the Garbled Circuit Generation phase.

If any of the $n-1$ parties was dishonest previously, it is guaranteed that at least one of those n sub-shares will be "lost" (since ZKSend will output a randomly selected value instead of the right sub-share to a dishonest party); in this case, GC is "lost" as well.

After all the executions of ZKSend are completed, P_i obtains $\{GC_1^i, \ldots, GC_n^i\}$. It then broadcasts $GC^i = \bigoplus_{j=1}^n GC_j^i$. Upon receiving GC^i for all i, all parties compute $GC = \bigoplus_{i=1}^n GC^i$.

This completes the description of the main ideas in our protocol. We note that apart from the above, our construction uses other (new and old) ideas to ensure correctness of the output, simulatability of the protocol etc. We refer the reader to section 5 for more details on the protocol.

D.3 Covert Computation over Point-to-Point Channels

The protocol in the previous section required the parties to have a common communication channel (for example, when the parties are exchanging e-mails over a mailing list). In this section, we consider the setting where the parties only have point to point communication channels (for example, when the parties only have private e-mail conversations with each other). This setting is quite different from the previous one since, in general, the parties may not have any innocent reason to send the same message to multiple other parties. See appendix A for more details on this setting.

There exists a rich body of literature on designing secure computation protocols over point to point channels (see [KK07],[KKK08] and references therein). However, to our knowledge, a common theme in all these works is a party sending the same message to multiple (or all) other parties over the pairwise private channels. Unfortunately, such techniques are inherently bound to fail in our scenario. The key challenge in our setting is to design a protocol where the messages exchanged between a pair of parties look indistinguishable from random even given the messages exchanged between all other pairs of parties (till the point when it is clear that all the parties are participating and that the output is favorable).

The basic idea of our construction is as follows. As part of the protocol specifications, the n parties are grouped into $n/2$ pairs. Each pair of parties run a covert two-party computation protocol to emulate a *virtual party*. This leads to a total of $n/2$ virtual parties. These virtual parties are further grouped into $n/4$ pairs and each pair of virtual parties run a covert two-party computation protocol to emulate another virtual party. By applying this idea recursively, there would eventually be a single virtual party which has all the required inputs and thus computes the output. Very informally, even if a single (real) party behaves honestly in the protocol, the final single virtual party would be "honest" as well.

In order to realize the above ideas, we first outline the necessary changes to our protocol in the previous section in order to construct a covert computation protocol for *reactive* functionalities in the *bounded concurrent* setting. Security in the bounded concurrent setting is important since there would be multiple uncoordinated executions of the covert two-party computation protocol in our final construction. We then define a hierarchy of reactive functionalities where the functionality at level 0 has all the required inputs and can compute the output while each functionality at level $\log(n)$ defines the algorithm of a real party.

We show how these virtual parties can communicate with each other through parties at the level just "below". The communication channel of these virtual parties can, in general, be controlled by a man in the middle. We employ the techniques from Barak et al [BCL$^+$05] to solve this problem. Finally, we show that the round complexity of our protocol is polynomial in n, and that the computational complexity of our protocol is polynomial in the security parameter if the number of parties n is a constant.