Covert Multi-party Computation

Nishanth Chandran* Vipul Goyal* Rafail Ostrovsky† Amit Sahai‡
University of California, Los Angeles
{nishanth,vipul,rafail,sahai}@cs.ucla.edu

Abstract

In STOC'05, Ahn, Hopper and Langford introduced the notion of covert computation. A covert computation protocol is one in which parties can run a protocol without knowing if other parties are also participating in the protocol 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. Ahn et al. constructed a protocol for covert two-party computation in the random oracle model.

*In this paper, we offer a construction for covert **multi-party** computation. Our construction is in the standard model and does not require random oracles. In order to achieve this goal, we introduce a number of new techniques. Central to our work is the development of "zero-knowledge proofs to garbled circuits," which we believe could be of independent interest. Along the way, we also develop a definition of covert computation as per the Ideal/Real model simulation paradigm.*

1. Introduction

The notion of secure multi-party computation was introduced by Yao [16] and Goldreich, Micali and Wigderson [9]. In multi-party computation, parties $\{P_1, ..., P_n\}$ holding inputs $\{x_1, x_2, ..., x_n\}$ wish to compute $f(x_1, x_2, ..., x_n)$ in such a way that at the end of the protocol, all parties obtain $f(x_1, x_2, ..., x_n)$ and party P_i cannot obtain any information other than what can be determined given just $f(x_1, x_2, ..., x_n)$ and x_i.

*Research partially done while visiting IPAM. Supported in part by grants listed below.

†Research partially done while visiting IPAM. This research was supported in part by IBM Faculty Award, Xerox Innovation Group Award, NSF Cybertrust grant no. 0430254, and U.C. MICRO grant.

‡Research partially done while visiting IPAM. This research was supported in part by NSF ITR and Cybertrust programs (including grants 0627781, 0456717, and 0205594), a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, and an Alfred P. Sloan Foundation Research Fellowship.

Now, consider the following situation: $\{P_1, P_2, ..., P_n\}$ are members of an organization. The organization also has a leader who is a harsh dictator! If all the members of the organization unite in their fight against the leader, then they can overthrow the leader and elect a new one. The combined strength of all members is needed in order to overthrow the leader. However, if it becomes known that some subset of the members wish to rise against the authority (but not all of them), then this could prove to be disastrous for the members opposing the leader. The members somehow wish to run a protocol such that at the end of the protocol, if all members wish to oppose the leader, the output is 1, otherwise the output is 0. This can be done using standard multi-party computation. But for this to work, one of the members P_i needs to come forward and ask the question "shall we run a protocol to see if all of us are interested in overthrowing the leader?". This already reveals information about P_i's intent. Is it possible for the parties to run a secure multi-party protocol in such a way that the execution of the protocol is not known until and unless the output is favorable to all parties? In this paper, we answer this question in the affirmative. Specifically, we construct a protocol which allows a group of n parties to perform *Covert Multi-party Computation*.

Ahn, Hopper and Langford introduced this fascinating notion of covert computation [14], and gave a protocol for covert two-party computation in the random oracle model. Informally, a covert computation protocol has the following properties: (1) No outsider can tell if the parties are running a protocol or just communicating normally; (2) No party has a significant advantage over other parties in determining if the parties are participating in the protocol or not; (3) Even if one party does not take part in the protocol, or the output of the computation is "unfavorable", then the execution of the protocol is indistinguishable from "ordinary looking" conversations. Ahn et al [14] discuss several applications of covert two-party computation such as dating, cheating in card games and so on. Many of them also apply to multi-party computation,. For illustration, we discuss a few applications below:

Joint buying over of a company: There exist a set of n companies who may be potentially interested in jointly buying another company C. They individually do not have the resources to buy the company. If a company reveals its interest in buying C, this could significantly increase the stock price of C (thus making it more expensive). To prevent such a situation, the n companies can run covert computation to determine if all of them are interested in buying C (and pooling resources for it) and if they jointly possess the resources to do so. Only if this is the case will the joint intention of all the companies to buy C be revealed to each other.

Detecting co-spies in Military Camps: Imagine a set of n spies who have infiltrated into an enemy camp. It would be very useful if they could collectively obtain information from the enemy. On the other hand, suggesting that they may be a spy to a normal member of the military camp could prove to be disastrous. A set of people can run covert multi-party computation with their signed credentials as inputs to determine if all of them are spies. The result is revealed only if they are all allies (in other words, covert computation can be used to perform *handshakes* among the members of a secret community).

Tracing a hacker: There exists a set of companies whose data sets have possibly been hacked into by an external party. No company wishes to admit that it is under attack. Yet, they all know that if they join forces, they can prevent further damage and possibly trace the attacker. The companies can run covert computation to determine if all of them have been attacked by the hacker.

Aside from the fact that covert multi-party computation has these interesting applications, the various technical challenges which it comes with makes covert computation interesting theoretically.

There are a few natural questions that arise while considering the notion of covert computation. These issues were discussed in [14], 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 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.)

Does MPC and Steganography lead to Covert MPC? At first glance, it may seem that by combining multi-party computation and steganography, we can immediately obtain covert computation protocols. Again, as argued by [14], 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 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 Our Contributions

Our main contribution in this paper is the first construction of covert *multi-party* computation. Our construction is in the standard model. Along the way, we also introduce a tool that we call a "zero-knowledge proof to garbled circuit," which we believe could be of independent interest.

We also develop a clean new definition of covert computation which is based on the Ideal/Real model simulation paradigm. This definition adds various desirable properties beyond those guaranteed by the model of [14], as we discuss in Section 2. Techniques similar to what we develop here may be necessary even for covert two-party computation satisfying an Ideal/Real model simulation based definition.

In many applications of covert computation, a notion of fairness is central to the covertness that we desire: We want that at the end of the protocol, either all parties learn about the participation of other parties, or none do[1]. However, analyzing fairness and covertness together is complex. One

[1]This is exactly analogous to the notion of fairness for output delivery. Indeed, we will later show that techniques developed for output delivery can be composed with our techniques for covertness to achieve "fair covertness".

of our technical contributions is to give a meaningful definition of covert multi-party computation that does not guarantee fairness. We provide a relatively clean construction (incorporating all our main ideas) that achieves this definition. We then construct a covert version of "Timed-commitment" [2, 12] (which is a standard tool to add fairness in secure computation protocols). Finally, we show that by following known techniques for achieving fairness [12, 6], we can modify our construction (using covert timed-commitments) to achieve the end goal of fair covert multi-party computation.

New Techniques. We now describe the main technical challenges that we encounter and the techniques used to overcome them.

1. Enforcing honest behavior in covert computation is a non-trivial task. This is because if one party "verifies" that another party is executing the protocol honestly, then covertness is immediately compromised. To enforce honest behavior, our main tool is a novel technique which we call "zero-knowledge to garbled circuits".

2. Unfortunately, however, zero-knowledge 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. We first modify the GMW multi-party computation protocol [9] (to incorporate correctness and other required properties) and use zero-knowledge to garbled circuit in a specific manner at the end. We then do a direct analysis of the resulting protocol to prove that it indeed satisfies the notion of covert multi-party computation.

3. We also construct a covert version of "timed-commitments" [2, 12] (which is a standard tool to add fairness in secure computation protocols), using techniques similar to those used to construct "dense cryptosystems" [13]. Covert timed-commitments can then be directly employed in covert computation protocols to achieve fairness (similar to how timed-commitments are employed in regular secure computation protocols [12, 6]).

1.2 Covert Two-Party vs Multi-Party Computation

In this section, we informally describe some of the ideas of the covert two-party computation protocol of Ahn et al [14], and argue why we cannot translate their results and techniques to achieve our result. The starting point of [14] is an observation that the problem of covert two-party computation reduces to designing a protocol in which the messages sent by each party are indistinguishable from random to the other one. They first consider a simple two party computation scenario where only the first party has to receive the output and there is no requirement of output correctness. This problem can be solved by using Yao's garbled circuits [16] and 1-out-of-2 oblivious transfers (OT). Ahn et al [14] modify Yao's garbled circuits to create a covert version by using an encryption scheme where a ciphertext is indistinguishable from a random number (indeed, a garbled circuit is simply a number of encrypted secret keys). They also construct a covert 1-out-of-2 OT by modifying an OT protocol of Naor and Pinkas [11]. This already gives a weak form of covert two-party computation where only one of the parties receive the output (and there is no correctness guarantee). One could use the protocol twice (once in each direction) where both parties should get the output.

A novel idea used by [14] to obtain their final result is to use covert garbled circuits to *replace* zero-knowledge proofs: We have already argued that zero-knowledge proofs destroy covertness (since they are verifiable). Ahn et al get around this problem as follows: Instead of sending a message directly, each party will send a garbled circuit. This circuit takes as input all the secrets of the other party, checks the honesty of the conversation so far, and if it is correct, outputs the next message. If the recipient has cheated, then the garbled circuit outputs a random message.

Unfortunately, this idea does not extend to our situation. The main reason is that a dishonestly prepared garbled circuit would learn all the secrets of some party A, and could output a message to A that causes A's later communication to destroy A's privacy. In the two-party case, as soon as A receives such a dishonestly prepared garbled circuit from B, B stops getting any further messages from A (since B behaved dishonestly). However in the multi-party setting, another party C could still get further messages from A (thus potentially destroying A's privacy). Furthermore, this does lead to other undesirable situations even in the two party case [14]; in Appendix A, we describe a situation involving the protocol of [14] in which one of the parties could force the other party's output of the protocol to be, for example, the other party's input itself. To overcome this problem, we develop a method of providing zero-knowledge proofs to garbled circuits, which guarantees that even dishonestly prepared garbled circuits do not learn any new information.

2 Preliminaries and Model

Network Model. We consider a system of n parties that 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 [14], we envision that the protocol messages are hidden by the parties in "ordinary" or "innocent-looking" messages (using steganographic techniques). Ordinary communication patterns and messages are formally defined in a manner similar to the channels used by [10] and [15]. The only difference between the channels in [14] and the channels we use, is that our channel is a *broadcast channel* shared by all n participants in the protocol.

Similar to [14], 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 above lemma follows from the proof for the bidirectional channel present in [14]. We refer the reader to [14] for further details.

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.

2.1 Covert Computation Model

We first consider a model of covert computation which does not include the notion of fairness (we discuss the issue of fairness later on). Informally speaking, our model is as follows. As before, 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 are 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 $f : (\{0,1\}^*)^n \rightarrow \{0,1\}^*$, 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 $g : (\{0,1\}^*)^n \rightarrow \{0,1\}$ 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 as discussed in [7]. 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. Let $\mathcal{M} \subset [n]$ denote the set of players malicious by the adversary. Then 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 (jumping 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 party P_i for all $i \in \mathcal{M}$.

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 $\vec{PD}$ (indicating 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* , 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 expected* PPT *adversary $\mathcal{S}$ corrupting at most t players in the ideal model, such that for all $\vec{PD} \in (\{0,1\})^n, \vec{x} \in (\{0,1\}^*)^n$, we have:*

$$\left\{\text{IDEAL}_{f,g,\mathcal{S}}(\vec{PD}, \vec{x})\right\} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}}(\vec{PD}, \vec{x})\right\}$$

Fair Covert Multi-Party Computation: We now consider the issue of fairness (of getting output or learning about participation). Similar to standard multi-party computation [6], we note that achieving full fairness (which could be modeled by requiring that the adversary has to instruct the trusted party either to abort or to send the result to every party *before* getting the result) of output is impossible [4] and this could be easily extended to participation data. However, a weaker notion (similar to [12, 6]) of fairness is indeed achievable. In this notion, we divide the protocol in two phases namely: computation phase and output revealing phase. In the computation phase, every party has negligible advantage in predicting the output and party participation. In the output revealing phase, if a subset of parties together have advantage ϵ in predicting the output and party participation, then all honest parties have roughly the same advantage (e.g., polynomial in ϵ) in predicting the output and party participation. We call a protocol satisfying

this notion of fairness as *fair covert multi-party computation*. We remark here that the properties of covertness and fairness are tied to each other. Suppose, at a given stage, a party has an ϵ advantage towards guessing the value of the output, then it has at least an ϵ advantage towards guessing party participation as well. We give a construction of fair covert multi-party computation in the full version of this paper [3].

Comparing our model to the model of Ahn et al [14]: The approach taken in [14] to define covert computation was different. They consider various desirable properties of covert computation (like strong internal covertness, strong fairness and final covertness) and give separate definitions for each of them. We first note that it is not clear to us if one can satisfy the property of fairness and final covertness simultaneously. To the best of our interpretation, final covertness informally means that even after the protocol has terminated and the output was favorable, the fact that any party participated remains hidden unless the output itself is distinguishable from random. Indeed it seems that their construction does not achieve these two properties together. It is very easy to repair the construction in such a way that the property of final covertness is lost, but we do not see how to repair it while still guaranteeing final covertness. We note that our definition does not require final covertness. More discussion on this issue is included in Appendix A.

Also, a desirable property that is not included in the model of [14] is that of *correctness of output*. In their protocol, it is possible for a malicious party to misbehave in a way which allows it to "force" any value as the function output on the honest party, even without having an input which would lead to such a function output (although the malicious party would not be able to itself learn any information in this case)[2]. We note that correctness is a desirable property in covert computation (e.g., if the adversary can trick the honest party into thinking that the function output is favorable, the honest party may decide to behave in a way that could be fatal to it). This property is addressed by the Ideal/Real model simulation-based definition that we propose.

3 Summary of Covert Computation Primitives

In this section, we list the primitives required to build covert multi-party computation. These primitives are "covert" variants of the standard ones. In other words, all the messages sent by a party in these primitives look indistinguishable from random messages. Below, we give a brief description of each of these primitives. A more detailed description can be found in the full version of this paper [3].

1. **Covert Commitments [8]:** The commitment scheme based on the Goldreich-Levin hard-core predicate is a covert bit commitment that is perfectly binding, since the commitments are indistinguishable from random.

2. **Covert 1-out-of-2 and 1-out-of-4 Oblivious Transfer [14, 11]:** Covert 1-out-of-2 Oblivious Transfer is presented in [14], while a slight modification of the protocol gives us covert 1-out-of-4 Oblivious Transfer.

3. **Covert Yao's Garbled Circuit [14, 16]:** This is a version of garbled circuit [16] in which the garbled circuit *program* (which is just a series of encrypted gate tables [16, 1]) can be evaluated in a "non-verifiable" manner. Covert garbled circuits can be used (in conjunction with covert 1-out-of-2 oblivious transfers) as a building block in covert two-party computation [14].

4. **Covert Semi-honest GMW multi-party computation protocol [9]:** This is just the standard (semi-honest) GMW multi-party computation protocol [9] where (a) we use *covert* 1-out-of-4 oblivious transfer, (b) we exclude the final share broadcast phase. That is, in covert GMW multi-party computation protocol, the parties initially hold their inputs and 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).

Notation Let a and b be two strings. $a \circ b$ denotes the concatenation of the two strings a and b. We write $Com(x)$ to denote a perfectly binding covert commitment to a string or bit x and $Open(Com(x))$ denotes the opening to $Com(x)$ (which x along with the randomness used in computing $Com(x)$). $\mathcal{P} = [n]$ denotes the set of all parties, $\mathcal{M} \subset [n]$ denotes the set of parties malicious by the adversary and $\mathcal{H} = \mathcal{P} - \mathcal{M}$ denotes the set of honest parties in the protocol.

4 Zero-knowledge Proofs to Garbled Circuits

In this section, we describe a tool for the construction of covert multi-party computation. Suppose P_i wishes to give P_j a value v, only if P_j can prove an NP statement L_j. If the proof is incorrect, then P_j must receive a random value. We also desire covertness, that is, none of the parties should know whether the other participated or not (and in

[2] In particular, the malicious party could send a garbled circuit (as per their protocol) which takes secrets of the honest party as input and outputs any value (possibly even dependent on the honest party's input)

particular, P_i should not know if P_j gave a correct proof to the statement or not).

One way to achieve this is to have P_i send a covert garbled circuit to P_j which takes as input the statement L_j (input by P_i) and the witness w_j (input by P_j). The garbled circuit checks if the witness w_j is a valid witness that proves L_j. If this is the case, the circuit outputs the required value v to P_j (and a random value otherwise). Now, if we were to use this construction as a subprotocol in a bigger protocol, then the following is a possible concern (as discussed before). If P_j feeds in the witness w_j to the covert garbled circuit (that could have been maliciously prepared by P_i), the output could be w_j itself or some function of it. Hence P_j could potentially lose its covertness or security (of input) if P_j uses this output to compute an outgoing message (later in the protocol). In our construction, we require a stronger property from this kind of functionality, namely, the output of the functionality should be independent of the secrets of the receiver (and picked by the sender from a distribution known in advance). To summarize, we need a protocol to achieve the following: (a) P_i has to give a value v to P_j only if P_j can prove an NP statement L_j (and P_j should receive a random number otherwise), (b) The output v to P_j is from a distribution "known"[3] to P_j, (c) The covertness of both P_i and P_j is preserved. We now proceed to present our construction and prove the above properties as 3 lemmas later on.

Let the value that P_i wishes to give P_j be denoted by v. Let the NP statement that P_j wishes to prove to P_i be L_j. Let P_j have a witness w_j that proves the NP statement L_j. Without loss of generality, we can assume that the statement L_j is "A given graph G has a Hamiltonian cycle". Hence w_j is a witness to the Hamiltonian cycle of the graph G. The high level idea behind the protocol is as follows: P_j initially picks a random permutation π of graph G and sends (covert) commitments of the adjacency matrix $H = \pi(G)$ of the graph. P_j will also commit to the permutation π. P_i will flip a bit b at random and send this to P_j. If $b = 0$, P_i will prepare a covert garbled circuit that takes as input: the opening to the commitment of the adjacency matrix of H and the opening to the commitment of the permutation π. This garbled circuit will check if the openings are valid and if $H = \pi(G)$. If these checks succeed, then the output of the garbled circuit is v. Otherwise, the output is a random value R. If $b = 1$, P_i prepares a covert garbled circuit that takes as input, the opening to the commitment to the edges in H forming a Hamiltonian cycle. This garbled circuit will check if the opening is valid and if the opened edges form a Hamiltonian cycle. If these checks succeed, then the output of the garbled circuit is v. Otherwise, the output is a random value R. P_i will then send the prepared covert garbled circuit to P_j. If $b = 0$, P_j's input to the garbled circuit is the opening to all the commitments and if $b = 1$, P_j's input to the garbled circuit is the opening to the edges of the Hamiltonian cycle. In this way, P_j will get the value v only if he gives a valid proof. Note that in this protocol, we achieve soundness only with probability half. To amplify the soundness so that a cheating P_j will be caught with overwhelming probability, P_i will break v into q shares (where q is determined by the security parameter) - $\{v_1, v_2, ..., v_q\}$ and run the above protocol q times, giving outputs $v_1, v_2, ..., v_q$ one by one in each round to P_j. If the statement L_j is true, P_j receives $v = \bigoplus_{t=1}^{n} v_t$. Otherwise, he will receive a random value. We describe the zero-knowledge proof to a garbled circuit protocol in more detail in the full version [3].

We call this protocol $ZKSend(i, j, v, L_j)$ (since it transfers a value v covertly from P_i to P_j only if P_j can prove in zero-knowledge the validity of statement L_j). We remark here that the above protocol achieves covert version of *conditional* oblivious transfer (conditional OT was introduced by di Crescenzo, Ostrovsky and Rajagopalan [5]), where a value is sent from a sender to a receiver conditioned upon the fact that a given NP statement is true. The sender does not know which of two values the receiver gets (i.e., if the NP statement was true or not). The receiver can choose one of two values, by either giving a valid proof to the NP statement or not.

We prove the following lemmas about this construction in the full version of this paper [3].

Lemma 2 (ZKSend Security) *Consider the following two security games: In the first game, honest party P_i executes $ZKSend(i, j, v, L_j)$ with P_j, where v is a fixed value. In the second game, honest party P_i executes $ZKSend(i, j, R, L_j)$ with P_j where R is drawn at random from the uniform distribution. In both games, the statement L_j is false. Then, no PPT adversary P_j can distinguish between the distributions of the views in the two games except with negligible probability.*

Lemma 3 (ZKSend Covertness) *In protocol $ZKSend(i, j, v, L_j)$, when v is random and if P_i is honest, no PPT machine P_j can distinguish between the messages sent by P_i and messages drawn at random from the uniform distribution, with probability greater than ϵ, where ϵ is negligible in the security parameter κ. Similarly, if P_j is honest, no PPT machine P_i can can distinguish between the messages sent by P_j and messages drawn at random from the uniform 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.*

[3] More formally, this means that there exists a simulator which can extract v from P_i even without having a witness to L_j

Lemma 4 (ZKSend Simulatability) *No PPT adversary P_i running an execution of $ZKSend(i, j, v, L_j)$, can distinguish between the view in a real execution of $ZKSend(i, j, v, L_j)$ and the view in a simulated execution of $ZKSend(i, j, v, L_j)$, with advantage $> \epsilon$ (where ϵ is negligible in κ).*

5 Covert Multi-party Computation without fairness

Parties $\{P_1, P_2, ..., P_n\}$ holding inputs $\{x_1, x_2, ..., x_n\}$ wish to compute a function $f(x_1, x_2, ..., x_n)$ covertly if and only if the function output is favorable to all parties. $g(x_1, x_2, ..., x_n)$ is the function that determines if the function output is favorable ($g(x_1, x_2, ..., x_n) = 1$) or not ($g(x_1, x_2, ..., x_n) = 0$). We first give an outline of the protocol and then discuss it in detail. The protocol consists of three main phases: the Input Commitment phase, the GMW phase and the Output Exchange phase. In the Input Commitment phase, all Parties commit to their inputs and randomness (note that we do not require coin flipping in our protocol) and run a challenge-response protocol with each other (which would enable the simulator to extract the input and randomness later on). In the GMW phase, the parties run a Covert-GMW protocol to compute a commitment to the function value and the opening to the commitment (if the function output is favorable to all parties). If the function value is not favorable, then the output of the Covert-GMW protocol is just a randomly chosen number. At the end of the GMW phase, the parties hold shares of the output (which is either a commitment along with its opening, or a random number). In the Output Exchange phase, parties exchange these shares with each other if and only if all the parties were honest till the GMW phase. We now discuss the protocol in more detail.

5.1 Input Commitment phase

1. *InputRandomCommit :* Let $X_i = x_i \circ K_i \circ r_i$, where x_i is the input of party P_i and r_i is a random tape that P_i will contribute to the GMW circuit (which requires randomness to compute its output). K_i is a random key chosen by P_i. This key will be used by P_i to check the correctness of the function output. This key will guarantee that the function output was indeed computed using the GMW circuit (and was not in some malicious way forced as the output by other parties). We shall explain later how this check exactly works. P_i additionally generates random tapes u_i, s_i and t_i. These random tapes will be used in different stages of the protocol. P_i now commits to X_i, u_i, s_i and t_i. Here u_i is the randomness that P_i will use in the GMW phase (that is, to break his input into shares and to execute the oblivious transfer protocols), s_i is the randomness that P_i will use in subprotocol *ExtractEnable* and t_i is the randomness that P_i will use in the *CommitmentExchange* subprotocol of the Output Exchange phase.

2. *ExtractEnable :* In this stage, parties execute a challenge-response protocol with each other. Intuitively, the protocol execution is such that it enables the simulator to extract (X_i, u_i) for all malicious parties P_i during simulation (e.g., by rewinding the execution). To execute this stage, a party P_i uses the random tape s_i wherever required (e.g., to compute commitments and generates challenges).

   Let $IR_i = Open(Com(X_i)) \circ Open(Com(u_i))$. Every pair of parties (P_i, P_j) will run the following l times ($1 \leq t \leq l$, where l is determined by the security parameter):

   (a) P_i picks a pair $(IR_{ij}^t[0], IR_{ij}^t[1])$ at random such that $IR_{ij}^t[0] \oplus IR_{ij}^t[1] = IR_i$. It then commits to this pair by sending $(Com(IR_{ij}^t[0]), Com(IR_{ij}^t[1]))$.

   (b) P_j now picks a bit q_{ij}^t at random and sends it to P_i.

   (c) P_i responds with $IR_{ij}^t[q_{ij}^t]$. Note that P_i does not send an opening to the commitment but only sends $IR_{ij}^t[q_{ij}^t]$. This is because sending an opening will compromise P_i's covertness.

   We note here that the *ExtractEnable* protocol will help the simulator extract the inputs and some specific randomness (i.e., X_i and u_i) of all malicious parties during protocol simulation (by rewinding the protocol execution) *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.

5.2 GMW phase

In this phase all parties execute a Covert-GMW protocol (see Section 3). In this protocol, the parties jointly evaluate a circuit for the following function (which takes as input

$\{X_1, X_2, ..., X_n\}$ where $X_i = x_i \circ K_i \circ r_i$):

$$F(X_1, ..., X_n) = \begin{cases} (R_1, R_2) & \text{if } g(x_1, x_2, ..., x_n) = 0 \\ (Com(f(x_1, ..., x_n), K_1, ..., K_n), & \\ Open(Com(f(x_1, ..., x_n), K_1, ..., K_n))) & \\ \text{otherwise} & \end{cases}$$

Where R_1 and R_2 are randomly selected values of appropriate length. The randomness used by this function is $r_1 \oplus r_2 \oplus ..... \oplus r_n$. The randomness used by P_i to execute the Covert-GMW protocol (namely to compute shares of its input and to execute all the oblivious transfers) comes from u_i. Let $F(X_1, X_2, ..., X_n) = (F[1], F[2])$ where $F[1]$ is the commitment $Com(f(x_1, x_2, ..., x_n), K_1, K_2, ..., K_n)$ while $F[2]$ is its opening $Open(Com(f(x_1, x_2, ..., x_n), K_1, K_2, ..., K_n))$. At the end of the GMW phase, every party P_i will hold shares of the output, i.e., P_i holds $(F_i[1], F_i[2])$ such that $\bigoplus_{t=1}^{n} F_t[1] = F[1]$ and $\bigoplus_{t=1}^{n} F_t[2] = F[2]$.

The Covert-GMW protocol is similar to the semi-honest GMW protocol (except for the fact that it uses our specific covert secure 1-out-of-4 OT based on [11] rather than a semi-honest 1-out-of-4 OT). What guarantees could it possibly provide when the parties might deviate from the protocol executions arbitrarily? Interestingly, we show that the above protocol (when used in conjunction with our fully secure OT rather than a semi-honest one) maintains privacy of the inputs (if the parties do not broadcast their output shares). More precisely, the view of an adversary (controlling up to $n-1$ parties) in the above protocol can be simulated until the output broadcast phase.

Intuitively, the malicious parties, even if they deviate from the protocol arbitrary, do not learn any information (in a computational sense) in this phase because of the following. The GMW protocol (not considering output share broadcast) consists of only two kind of message: one where parties broadcast $(n-1)$ random shares of their input to other parties, the second where parties engage in a 1-out-of-4 OT protocol with others. The first type of message obviously does not reveal any information about the output (since its just a randomly selected number). In the second type of messages, a party gives away only one of the 4 bits (when acting as a sender of information in a covert 1-out-of-4 OT protocol). However, all the four bits are *individually* indistinguishable from a random. This is because while preparing these four bits, a single bit of randomness is used [9] (making the four bits *individually* random).

5.3 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.

This phase consists of two subprotocols: *CommitmentExchange* and *OpeningExchange*. Informally, in the *CommitmentExchange* phase, the parties exchange their shares (held at the end of GMW phase) of the commitment *conditioned* upon the fact that *every party* was honest till the end of GMW phase. In the *OpeningExchange* phase, the parties exchange their shares of the commitment opening conditioned upon the fact that every party was honest till the end of *CommitmentExchange* phase. These two subprotocols heavily rely on the various properties of our zero-knowledge to garbled circuit (i.e., $ZKSend(i, j, v, L_j)$) construction.

1. *CommitmentExchange :* The randomness that a party P_i uses in this subprotocol comes from t_i. In this subprotocol, parties exchange shares of $F[1]$. Every party P_i breaks his share of $F[1]$ (namely $F_i[1]$) further into n shares: $\{F_i^1[1], F_i^2[1], ..., F_i^n[1]\}$. P_i then runs $ZKSend(i, j, F_i^j[1], L_j)$ with every $P_j, j \neq i$. L_j is an NP statements which asserts that the party P_j was honest in the protocol up to the end of the GMW phase. More specifically, L_j is the following statement: There exists $X_j (= x_j \circ K_j \circ r_j), u_j$ and s_j such that,

   - There exist strings for the commitments $Com(X_j), Com(u_j)$ and $Com(s_j)$ (broadcast in the Input Commitment phase) that open the commitments to X_j, u_j and s_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) GMW phase using randomness u_j.

   Upon having $\{F_1^i[1], F_2^i[1], ..., F_n^i[1]\}$ after executing $ZKSend$ with other parties to get $(n-1)$ shares (the share $F_i^i[1]$ is already held by P_i), the party P_i broadcasts $F^i[1] = \bigoplus_{t=1}^{n} F_t^i[1]$. Upon receiving $F^i[1]$ for all i, all parties compute $F[1] = \bigoplus_{i=1}^{n} F^i[1]$.

   This protocol is used to do the following. A party P_i holds a share of the output (at the end of the GMW phase). Only if the other $n-1$ parties were honest, P_i would like to give its share out. However, if any party deviated from the prescribed protocol during the Input commitment or GMW phase, it could be potentially dangerous for P_i to give out its output share (since the

output of the GMW circuit then is not guaranteed to be the function value only). Hence, P_i breaks its share further into n subshares and transfers these subshares one by one to other parties using $ZKSend$. If any of the 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 happened in previous stages (and the output share is not a share to the right function output), the GMW output is essentially "lost".

2. *OpeningExchange :* In this subprotocol, parties exchange shares of $F[2]$. Every party P_i breaks his share of $F[2]$ (namely $F_i[2]$) into n further shares $\{F_i^1[2], F_i^2[2], ..., F_i^n[2]\}$. P_i, then runs $ZKSend(i, j, F_i^j[2], L_j)$ with every $P_j, j \neq i$. L_j is an NP statement which asserts that the party P_j was honest in the protocol up to the end of *CommitmentExchange* in the Output Exchange phase. Similar to the *CommitmentExchange* subprotocol, upon receiving $\{F_1^i[2], F_2^i[2], ..., F_n^i[2]\}$, P_i broadcasts $F^i[2] = \bigoplus_{t=1}^{n} F_t^i[2]$. Finally, all parties compute $F[2]$.

   All parties now check if $F[2]$ is a valid opening to the commitment $F[1]$. Every party P_i further checks if K_i is contained in $F[2]$. If these checks succeed, then the output of P_i is $f(x_1, x_2, ..., x_n)$. Otherwise if any of the checks fail, the output of P_i is $\perp$.

   The only goal of this subprotocol is to ensure correctness of the output (rather than privacy of inputs or covertness etc). If all the parties were honest till *CommitmentExchange*, they already got a commitment to the right function output (and thus an incorrect function output cannot be "forced" on a party). Otherwise, a party P_i ensures that the share of its commitment opening is "lost" (and thus the secret value K_i cannot figure in the final broadcast).

A full proof of security (as per our Ideal/Real model simulation based definition) of the above construction can be found in the full version of this paper [3].

Acknowledgements: We thank Yuval Ishai and Manoj Prabhakaran for several helpful discussions.

References

[1] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[2] D. Boneh and M. Naor. Timed commitments. In *CRYPTO '00: Advances in Cryptology, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24*, pages 236–254, 2000.

[3] N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai. Covert multi-party computation. Cryptology ePrint Archive, 2007. http://eprint.iacr.org/.

[4] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.

[5] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT*, pages 74–89, 1999.

[6] J. A. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *TCC '06: Proceedings of the Third Theory of Cryptography Conference*, pages 404–428, 2006.

[7] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[8] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, 15-17 May, Seattle, Washington, USA*, pages 25–32, 1989.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the 19th annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[10] N. J. Hopper, J. Langford, and L. von Ahn. Provably secure steganography. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 77–92, London, UK, 2002. Springer-Verlag.

[11] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[12] B. Pinkas. Fair secure two-party computation. In *EUROCRYPT '03: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, Proceedings*, pages 87–105, 2003.

[13] A. D. Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *FOCS '92: 33rd Annual Symposium on Foundations of Computer Science, 24-27 October, Pittsburgh, Pennsylvania, USA*, pages 427–436, 1992.

[14] L. von Ahn, N. Hopper, and J. Langford. Covert two-party computation. In *STOC '05: Proceedings of the 37th annual ACM symposium on Theory of computing*, pages 513–522, New York, NY, USA, 2005. ACM Press.

[15] L. von Ahn and N. J. Hopper. Public-key steganography. In *EUROCRYPT '04: Proceedings of the 23rd Annual Eurocrypt Conference on Advances in Cryptology*, pages 323–341. Springer-Verlag, 2004.

[16] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *FOCS '82: Proceedings of 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

Appendix

A Covert Two-Party Computation Protocol of [14]

In this subsection, we recall the covert two-party computation protocol (in the malicious adversary model) from [14]. We then point out a scenario in which their construction does not satisfy the property of fairness. It is very easy to repair the construction to achieve fairness, but such that the property of final covertness is lost. However, it is not clear to us if the properties of fairness and final covertness can be satisfied simultaneously. The protocol of Ahn et al [14] proceeds in three stages and is as follows (here G is modeled as a random oracle):

1. Commit Stage: Alice picks $k + 2$ strings, $r_0, s_0[0], s_0[1], ..., s_0[k]$, each k bits in length, where k is related to the security parameter κ. Alice commits to these strings using the covert commitment scheme. The strings $s_0[0], ..., s_0[k]$ are state values used by Alice to check if Bob was honest. Bob does likewise with values $r_1, s_1[0], ..., s_1[k]$.

2. Compute Stage: We make use of a pseudorandom generator $G : \{0,1\}^k \leftarrow \{0,1\}^l$.

   - Alice and Bob run two serial rounds of Covert-YAO as described earlier.
   - If neither party cheats and if the output is favorable, then Alice knows $f(x,y) \oplus G(r_1)$ and Bob's first state value $s_1[0]$. Bob knows $f(x,y) \oplus G(r_0)$ and Alice's first state value $s_0[0]$.

3. Reveal Stage: This consists of k rounds where each round consists of two runs of the Covert-YAO protocol.

   - At the end of round i, if nobody cheats, Alice learns the i^{th} bit of Bob's string r_1, labeled $r_1[i]$ and also the next state value, $s_1[i]$ of Bob. Similarly, Bob learns $r_0[i]$ and $s_0[i]$, the next state value of Alice (see [14] for a full description).
   - After k rounds in which neither party cheats, Alice knows r_1 and can compute $f(x,y)$ and likewise Bob can compute the output.

Fairness scenario. Now, we consider the following situation. Let $w = w_1 w_2 .... w_n$ and $w' = w_1 w_2 .... \bar{w_n}$ be two strings such that $G(w) \neq G(w')$. In other words, w and w' differ only in the last bit, but their outputs when fed into the random oracle G are complete different. Now, Alice sets $r_0 = w$ and takes part in the covert two-party computation protocol with Bob. At the last round of the reveal stage, Alice, after learning Bob's string r_1 completely, quits the protocol without giving Bob, the value of w_n. Now, although Alice gets the output completely, Bob has ambiguity between two equally likely values of the output. Furthermore, in the event that $f(x,y) \in \{0,1\}$, the output is completely hidden from Bob. In this way, the protocol is unfair. To correct this, Alice and Bob could initially compute $H(r_0)$ and $H(r_1)$ respectively (where H is modeled as a different random oracle) and output these values as commitments to r_0 and r_1. Now, if some party quits the protocol after receiving n bits, the other party has at least $n-1$ bits and after guessing the last bit, the party can verify the value by computing $H(r)$. In short, a verifiable value needs to be exchanged between Alice and Bob (so that if one party aborts early, the verifiable value allows the other party to "guess and check" the remaining bits). However clearly such a verifiable value destroys the property of final covertness (i.e., if every party participated and the function output was favorable, then every party knows that the computation happened).

Output Correctness Scenario. We now consider another scenario in which it is possible for the malicious party (say Alice) to misbehave in a way which allows it to "force" any value as the function output on the honest party (say Bob), without having an input which would lead to such a function output. (We stress that in the protocol of [14], however, privacy is guaranteed: Alice would not be able to itself learn any information in this case). To do this, Alice starts deviating from the protocol from the very beginning and prepares an incorrect garbled circuit in the compute stage itself. The garbled circuit takes as input the secrets (including input) of Bob (as per the protocol) and chooses a string s possibly depending upon Bob's inputs. The garbled circuit now outputs $s \oplus G(r_0)$ to Bob (as opposed to $f(x,y) \oplus G(r_0)$). In the reveal stage, Alice's garbled circuits simply reveal bits of r_0 one by one as per the protocol specifications. Since Alice deviates from the protocol, she would be unable to get anything except uniformly chosen messages from the garbled circuits prepared by Bob. However, Alice is still free to prepare her garbled circuits maliciously in a manner described above. Hence, the output of Bob is forced to be the string s.