Achieving privacy in verifiable computation with multiple servers – without FHE and without pre-processing*

Prabhanjan Ananth[1], Nishanth Chandran[2],
Vipul Goyal[2], Bhavana Kanukurthi[1], and Rafail Ostrovsky[3]

[1] Department of Computer Science, UCLA
prabhanjan@cs.ucla.edu, bhavanak@cs.bu.edu
[2] Microsoft Research India
{nichandr, vipul}@microsoft.com
[3] Departments of Computer Science and Mathematics, UCLA
rafail@cs.ucla.edu

Abstract. Cloud services provide a powerful resource to which weak clients may outsource their computation. While tremendously useful, they come with their own security challenges. One of the fundamental issues in cloud computation is: how does a client efficiently verify the correctness of computation performed on an untrusted server? Furthermore, how can the client be assured that the server learns nothing about its private inputs? In recent years, a number of proposals have been made for constructing verifiable computation protocols. Unfortunately, solutions that guarantee privacy of inputs (in addition to the correctness of computation) rely on the use of fully homomorphic encryption (FHE). An unfortunate consequence of this dependence on FHE, is that all hope of making verifiable computation implementable in practice hinges on the challenge of making FHE deployable in practice. This brings us to the following question: do we need fully homomorphic encryption to obtain privacy in verifiable computation protocol which achieves input privacy? Another drawback of existing protocols is that they require the client to run a pre-processing stage, in which the work done by the client is proportional to the function being outsourced and hence the outsourcing benefit is obtained only in an amortized sense. This brings us to our next question: can we build verifiable computation protocols that allow

* The first, fourth and fifth authors were supported in part by NSF grants CNS-0830803; CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; and in part by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The fifth author was also supported US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

the client to efficiently outsource even a computation that it wishes to execute just once?
In this paper, we consider a model in which the client outsources his computation to multiple (say $n \geq 2$) servers. In this model, we construct verifiable computation protocols that do not make use of FHE and that do not have a pre-processing stage. In the two-server setting, we present an extremely practical protocol based only on one-way functions. We also present a solution, based on the DDH assumption, for the multi-server model for any arbitrary n. All these protocols are secure as long as at least one server is honest. Finally, even in the n-server model, we present a solution based solely on one-way functions. This protocol tolerates up to a constant fraction of corrupted servers.

Keywords: Verifiable computation, delegatable computation, input/output privacy, garbled circuits.

1 Introduction

Recently, there have been a number of proposals for non-interactive verifiable computation protocols (also called delegation of computation) (c.f., [AIK10,GGP10,CKV10]). In this scenario, we have a computationally weak client talking to a powerful (but un-trusted) server. The client wishes to get the outcome of a desired computation (say a function $\mathcal{F}$ evaluated on an input x) with the help of the server. If the server is malicious, one could ask that the correctness of the output **and** the privacy of the input (and possibly output) of the client still be preserved. Of course, it is also imperative that the work done by the client in verifying the correctness of the output be much lesser than the work done in computing $\mathcal{F}(x)$ on his own.

Unfortunately, to the best of our knowledge, all proposed solutions that meet this security requirement, have the following two drawbacks: they rely on the assumption of fully homomorphic encryption (FHE) and they work in a pre-processing model which requires the weak client to perform work proportional to $\mathcal{F}$ during an initial pre-processing phase and only guarantees that the work done in the online phase is low. First, we see the reliance on fully homomorphic encryption as a drawback for two reasons: a) the verifiable computation protocols so obtained, are inefficient in practice since they require the client to perform FHE encryption (which, typically, is less efficient than regular encryption and have enormous public keys) as well as require the server to perform expensive computation on encrypted data; b) from a theoretical perspective, it would be interesting to base protocols for verifiable computation on weaker or (relatively more) well-studied cryptographic hardness assumptions[4] . Hence, an interesting

[4] The recent result of [BV13] constructs a *leveled* FHE scheme under the LWE assumption that matches the best-known assumption for lattice-based PKE. However, all standard FHE (i.e., non-leveled) schemes additionally require a much stronger circular security assumption.

question to ask is: *do we need fully homomorphic encryption to obtain privacy in non-interactive verifiable computation protocols?*

The second drawback of existing solutions is that they require the client to perform work proportional to $\mathcal{F}$ during an initial pre-processing phase. In addition to being a strong assumption, it is also meaningful only in settings where the client wishes to compute the same function many times. This brings us to the next question: *can we build verifiable computation protocols that allow the client to efficiently outsource computations (even ones that it wishes to execute just once)?*

In this work, we are interested in addressing the above questions. Before we do so, we first present some intuition on the challenge of avoiding FHE. First, note that in a non-interactive verifiable computation protocol, the client sends a single message to the server (that can be viewed as an "encryption" of x), and the server responds back with a single message from which the client can recover $\mathcal{F}(x)$ (hence this message must look like an encryption of $\mathcal{F}(x)$). If we require the client's computational complexity to be independent of $\mathcal{F}$, then inherently, every verifiable computation protocol seems to have an FHE scheme embedded in it[5]. In fact, even if we allow interaction between the client and the server, to the best of our knowledge we do not know verifiable computation protocols that achieve *privacy* without FHE.

1.1 Multi-server Model for Verifiable Computation

In light of the challenge in removing FHE in the single-server model, we turn to a model of verifiable computation in which a single client outsources its computation to multiple (say n) servers. Note that if, all n servers are un-trusted (and colluding), then this is equivalent to outsourcing computation to a single server, and again it seems that we require FHE to obtain secure protocols. Hence, we consider a model in which a single client, holding an input x, wishes to outsource the computation of some function $\mathcal{F}$, to a set of servers $\mathcal{S}_1, \cdots, \mathcal{S}_n$, such that client performs very little computation (independent of $\mathcal{F}$) *throughout the protocol* and yet has a guarantee that none of the servers learn x, nor can they force the client to accept any other output, other than the right value of $\mathcal{F}(x)$, even if up to $n-1$ of the servers are malicious and colluding. (Note that in the multi-server model, this is the strongest security one can achieve.)

Communication Model. Since client efficiency is our primary concern, we work in a model where the client sends and receives a single message, similar to the single-server non-interactive communication model. In particular, consider any (arbitrary) ordering of the servers. In our communication model, the client prepares a message and sends it (only) to the first server. Each intermediate server (except the first and the last server) receives a message from the previous server,

[5] This is not entirely true if the client is allowed to work in time proportional to $\mathcal{F}$ in the pre-processing stage, but it does seem like the only way we know how to obtain privacy in such protocols.

performs some computation, and, sends the outgoing message to the next server. The last server, upon doing its computation, sends the resulting message back to the client. After receiving this message, the client either accepts or rejects.

1.2 Our Results and Techniques.

We provide general positive results in the above model for any PPT computable function without relying on FHE and without requiring a pre-processing stage. Our constructions guarantee both: privacy for the input (and output) of the client, as well as the correctness of the output (in case client outputs accept). We now state the various results that we obtain in this work:

- We first consider the 2-server case. In this setting, we present a protocol that can be obtained from any non-private verifiable computation protocol[6] and any collision-resistant hash function. This protocol is secure (i.e, guarantees privacy and soundness) as long as at least one of the two servers is not corrupted.
- Furthermore, in the 2-server case, if we allow each server to send a message to the other (i.e., we add one new message from the second server to the first), we are able to achieve a highly practical protocol solely based on one-way functions. In fact, in this protocol, the client only needs to send $2\kappa(\lambda_x + \lambda_y)$ random bits, where λ_x and λ_y are the input and output lengths respectively and κ is the security parameter, (and then additionally perform a lookup), while the two servers only need to generate and evaluate a garbled circuit each (the work done by the two servers can be run in parallel further minimizing the time of the protocol execution).
- Next, we consider the n-server case. That is a client outsources the computation to n servers. Here, we construct a protocol based solely on the Decisional Diffie-Hellman (DDH) assumption that is secure as long as at least one of the n servers is not corrupted. The computational complexity of the client throughout the protocol is independent of $\mathcal{F}$, the function being outsourced; in particular it is $\mathcal{O}(\kappa \cdot \lambda_x)$. Since there is no preprocessing, there are no restrictions on which functions the servers might evaluate for the client. The function to be evaluated may even be different in different protocols executions (as long as there is a mechanism for the servers to get the function description without affecting the computational complexity of the client). The primary tool this construction relies on is the notion of rerandomizable Yao's garbled circuits due to Gentry, Halevi, and, Vaikuntanathan [GHV10] which we carefully put together along with a specific proxy re-encryption scheme [BBS98].
- Finally, we also show how to obtain a secure protocol in the n-server case that is based solely on one-way functions. This protocol is secure as long as a constant fraction of the n servers are not corrupted.

[6] That is, a verifiable computation protocol that does not necessarily guarantee any privacy

An added feature of all our protocols is that by using universal circuits, we can hide not just the input (x), but also the function ($\mathcal{F}$) being outsourced. This would be particularly useful in the case when the function description is short but the computational complexity of evaluating $\mathcal{F}$ is high. Finally, we note that our solutions do not suffer from the "rejection-bit" problem (that most earlier solutions suffer from) as we do not employ a pre-processing stage.

Remarks. We stress that we *do not* assume that the multiple malicious servers do not collude. Similar to standard secure multi-party computation protocols, all of our protocols are secure even when $n-1$ out of the n servers are malicious and colluding with each other. We remark that one could potentially reduce the communication between servers by making use of a fully homomorphic encryption (FHE) scheme; however, our goal is to not rely on FHE due to its inefficiency. Furthermore, it would seem unlikely for us to obtain an n-server protocol where server communication is independent of the function without relying on FHE as this would lead to a secure multi-party computation protocol with constant communication complexity (without relying on FHE). We stress that the lack of **any** positive results in getting privacy for outsourcing computation without FHE, makes it important to consider models such as ours.

We stress that, similar to standard MPC, in our protocol also, $n-1$ malicious servers could jointly collude and send their entire state to the honest server; this would mean that the honest server could learn the client's input. However, we do not view this as a serious limitation as this issue exists even in any MPC protocol that tolerates t corruptions. However, we stress that this problem does not arise in our setting as long as at most $n-2$ servers are corrupted. Even if $n-2$ servers send their state to one other server, it learns nothing as the protocol tolerates $n-1$ corruptions.

We finally remark that while one can obtain a private protocol for outsourcing computation in the multi-server model by simply secret sharing the clients input to the n servers and running a standard MPC protocol, our work shows that one can obtain significant efficiency gains when dealing with the specific problem of outsourcing computation (namely by leveraging the fact that the client is honest). In the 2-server setting, our protocol is even faster than standard *semi-honest* secure 2PC as we do not need to make use of oblivious transfer protocols (or zero-knowledge proofs/cut-and-choose techniques to obtain malicious security). We believe that our work can be a stepping stone towards obtaining faster protocols even in the multi-server setting.

Related work and open questions. As mentioned earlier, the works of Gennaro *et al.* [GGP10], Chung *et al.* [CKV10], Applebaum *et al.* [AIK10] were the firsts to consider non-interactive verifiable computation (with privacy) in the single server model. All the above protocols rely on fully homomorphic encryption to obtain privacy of the client's input. The works of [GKR08,KR09,GLR11,BCCT12,DFH12], and [GGPR13], all consider the problem of delegating computation, but without privacy (and obtain protocols for various classes of functions and under different assumptions). The works of

[BGV11,FG12] consider outsourcing the computation of polynomials (but not the inputs), while the work of [PRV12] considers (non-private) outsourcing of specific class of functions without FHE. Of course, one could additionally make these protocols private, by "enveloping them" under an FHE scheme; however, this is what we wish to avoid. Note that one could obtain a private verifiable computation protocol from an attribute-based encryption scheme that has the property of attribute-hiding (using the construction of [PRV12]); however, we remark that while recent work has constructed attribute-based encryption for all polynomial time functions [GVW13,GGH$^+$13], these works do not obtain attribute hiding. Furthermore, even then, using this transform, we will only get a verifiable computation protocol in the pre-processing model. Finally, the work of Goldwasser *et al.* [GKP$^+$13] shows how to construct reusable garbled circuits and from this show how to obtain a private scheme for delegating computation; however their construction makes use of a FHE scheme.

The works of Canetti *et al.* [CRR11,CRR12] were the first to consider verifiable computation in the multi-server setting. While they do not consider privacy of the client's inputs, they provide an unconditional guarantee of the client receiving the correct output (as long as at least one server is honest). The servers do not communicate in their model, however their protocol works only for a restricted class of functions (logspace uniform NC circuits). They also have a result based on computational assumptions that work for arbitrary polynomial sized circuits.

Kamara and Raykova [KR11] consider the problem of outsourcing computation in the "multi-tenant" setting, in which there any mutually untrusting tenants (clients) running computations on the same trusted server (*physical machine*). Our solutions can be extended to this setting and achieve an improvement in efficiency (e.g., the protocol in Section 4.1) compared to the solutions of [KR11].

In our work, we obtain protocols in which the client sends a single message to the first server and receives a single message from the last server, but each of the servers send and receive one message each. A very interesting open problem would be to obtain a private protocol, in which the client sends a single message to each of the servers and receives a single message from each of the servers, and can obtain the correct result from this (i.e., a model in which the servers do not communicate with each other at all).

Organization of the paper. We begin, in Section 2, by defining our security and communication model for multi-server verifiable computation. In Section 3, we give an overview of the main tools, that we use in constructing our protocols. We present our main n-server protocol based on the DDH assumption in Section 4.2. We describe an improvement of this protocol in which the client works in time independent of n in the full version [ACG$^+$14]. We refer the reader to the full version for details of our two 2-server protocols and the n-server protocol based on one-way functions (that tolerates a constant fraction of corrupt servers). We also refer the reader to the full version for more details of the constructions and for all the proofs.

2 Verifiable Computation in the Multi-server Setting

Let $\mathcal{VC}_{\mathsf{multiserv}} = (\mathcal{C}, \mathcal{S}_1, \ldots, \mathcal{S}_n)$ be a multi-server delegation scheme where $\mathcal{C}$ denotes the client and $\mathcal{S}_1, \ldots, \mathcal{S}_n$ denote the servers. The scheme basically consists of two stages - the first is the (one-time) setup stage and the second is the online stage. In the setup stage, denoted by $\mathsf{Setup}_{\mathcal{VC}_{\mathsf{multiserv}}}$, some computation is performed by the clients and the servers. The output of the setup stage consists of information public to everyone, as well as some secret information for the client as well as the servers. We stress that this stage is different from the standard pre-processing stage in literature as the work done in this stage is independent of the function $\mathcal{F}$ or the input x[7].

The second stage is the online stage when the client delegates the job of evaluating $\mathcal{F}$ on an input x to the set of servers. In this stage, the client runs in time independent of the complexity of the function $\mathcal{F}$.

A note on the setup stage. In all our constructions, the setup stage is independent of the function $\mathcal{F}$. As a result, the computational complexity of this stage is independent of the complexity of function $\mathcal{F}$. This is a much stronger condition than the proposed single-server delegation protocol [GGP10,CKV10] where the setup stage was allowed to run in time proportional to the complexity of $\mathcal{F}$. Another important advantage of the setup stage being independent of the function being delegated is that the client can execute this setup stage once (irrespective of the function being delegated) and store the secret state, which can then be reused for delegating *any* function, making our protocol efficient if the client wishes to delegate a number of different functions.

A multi-server delegation scheme should satisfy the properties of correctness, soundness and privacy. We refer the reader to the full version for the definitions.

3 Building blocks

3.1 A Variant of Garbled Circuits

Yao in his seminal paper [Yao82] introduced the notion of garbled circuits to construct a secure two-party computation protocol. For this work, as we will explain later, we will consider a variant of the garbled circuit construction, denoted by $\mathsf{YaoGarbledCkt}$, – namely, one in which the output wires are fixed. In this variant, the output wire keys are given externally to $\mathsf{YaoGarbledCkt}$ which in turn generates a garbled circuit with these fixed output wire keys. Though this violates the one-time soundness property of the garbled circuits, we will show, that this still ensures the privacy of the inputs which suffices for our construction. We formally show the proof of this claim in the full version.

[7] This requirement of having the setup stage to be independent of the function of the client makes our model significantly stronger than the ones considered in prior works.

In more detail, $\mathsf{YaoGarbledCkt}$ is a probabilistic polynomial time algorithm that takes as input a circuit $\mathcal{F}$[8], randomness R_1, R_2, and fixed output wire keys. Let the keys for the output wire be denoted by $\mathbf{w}_{\mathsf{out}} = \left\{(w^0_{\mathsf{out},1}, w^1_{\mathsf{out},1}), \ldots, (w^0_{\mathsf{out},\mu}, w^1_{\mathsf{out},\mu})\right\}$. It generates a garbled circuit according to Yao [Yao82]. R_1 is the randomness used to generate the input wire keys. R_2 is the randomness used to generate the wire keys for the rest of the circuit along with the four ciphertexts associated with every gate of the circuit. We will denote the collection of garbled gates by GC. Given the input wires corresponding to an input x, one can "evaluate" the garbled circuit and finally decode the output wires in order to obtain $\mathcal{F}(x)$.

To aid the construction we give later, we define another functionality, namely $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$, that does the following. $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$ takes as input randomness R_1, and outputs just the input wires corresponding to GC which is the output of $\mathsf{YaoGarbledCkt}(\mathcal{F}; (R_1, R_2))$. As we will see later, the client will use this algorithm to compute just the input wire keys for his input x, corresponding to the garbled circuit GC, without generating the entire garbled circuit (GC) itself. The procedure $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$ can be derived from $\mathsf{YaoGarbledCkt}$ such that the computational complexity of $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$ depends only on the size of the input to the function and not on the size of the garbled circuit itself. For more details, refer to the full version.

Re-randomizable Garbled circuits. In [GHV10], Gentry *et al.* gave an alternate construction of garbled circuits whose security was shown, based on the Decisional Diffie Hellman (DDH) assumption. The advantage of their construction was that the garbled circuits that were obtained from their approach could be rerandomized. We say that a garbled circuit produced by $\mathsf{YaoGarbledCkt}$ is rerandomizable when there exists an algorithm reRand which on input a garbled circuit produces a different garbled circuit such that no computationally bounded adversary can distinguish whether a given garbled circuit is obtained as a result of rerandomization or was computed from $\mathsf{YaoGarbledCkt}$, even when given the original garbled circuit. To explain this in more detail, we first define reRand. reRand takes as input a garbled circuit GC_1 (constructed from $\mathcal{F}$ and with fixed output wires $\mathbf{w}_{\mathsf{out}}$) and outputs another garbled circuit GC_2 (whose output wires are also fixed to $\mathbf{w}_{\mathsf{out}}$) such that the distribution of GC_1 is computationally indistinguishable from that of GC_2 even if the distinguisher is given access to $\mathcal{F}$ and the randomness used to compute GC_1. In addition to GC_1, reRand takes as input randomness (R_1, R_2) (and is denoted $\mathsf{reRand}(\mathsf{GC}_1, (R_1, R_2))$). R_1 is used to re-randomize the input wires while R_2 is used to re-randomize the rest of circuit. Note that the procedure reRand re-randomizes only the garbled circuit and not the output wires. So it does not need to take as input $\mathbf{w}_{\mathsf{out}}$. Gentry *et al.* construct re-randomizable garbled circuits whose output wires are also randomized; as mentioned earlier, we require a variant of garbled circuits whose output wires remain the same, even after re-randomizing. We will show that the

[8] We use the same symbol to denote the function as well as the circuit computing the function.

construction of Gentry *et al.* can be used even for our purposes and the security of the construction holds. For more details, we refer the reader to the full version.

We now define another functionality, namely $\mathsf{reRand}_{\mathsf{in}}$, on the lines of $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$ as follows. $\mathsf{reRand}_{\mathsf{in}}$ takes as input randomness R_1 and $\mathbf{w}_{\mathsf{GC}_1,\mathsf{in}}$, which are the input wire keys of a garbled circuit GC_1, and outputs $\mathbf{w}_{\mathsf{GC}_2,\mathsf{in}}$ which are the input wire keys corresponding to GC_2 where GC_2 is the output of $\mathsf{reRand}(\mathsf{GC}_1; (R_1, R_2))$. Like in the case of $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$, the $\mathsf{reRand}_{\mathsf{in}}$ algorithm can be easily derived from reRand such that the computational complexity of $\mathsf{reRand}_{\mathsf{in}}$ depends only on the size of the input to the function and not the size of the garbled circuit itself.

3.2 Re-encryption Scheme

Informally, a re-encryption scheme allows a third party, who possesses a re-encryption key, to transform ciphertexts encrypted under one public key pk_1 into ciphertexts of the same message under a different public key pk_2, without learning anything about the contents of the message m. Various constructions of re-encryption schemes are known; we require a re-encryption scheme that is also additively homomorphic. We show such a scheme and provide more details about it in the full version.

4 Constructions of Verifiable Computation Protocols

In this section, we shall present our protocols for verifiable computation in the multi-server model. We shall first begin by describing a protocol in the 2-server case that can be built from any non-private verifiable computation protocol coupled with any collision-resistant hash function family. We will then build our n-server protocol that is based on the Decisional Diffie-Hellman assumption. Our n-server protocol based on one-way functions (but handling only a constant fraction of corrupt servers) is given in the full version.

4.1 The Two-Server Case

We wish to construct a verifiable computation protocol that allows a client $\mathcal{C}$ to outsource the computation of $\mathcal{F}$ on input x to two servers $\mathcal{S}_1$ and $\mathcal{S}_2$ with a guarantee on both privacy and soundness when at least one server is honest. We present two protocols for this purpose.

Solution 1: The high level idea for the first protocol is as follows. $\mathcal{C}$ will pick a seed to pseudo-random function (PRF) family and send the seed to $\mathcal{S}_1$. The client will also generate the output wires of a garbled circuit for function $\mathcal{F}$ (as described in Section 3.1 using $\mathsf{YaoGarbledCkt}_{\mathsf{in}}$) and send them to $\mathcal{S}_1$. Finally, the client also picks a key to a collision-resistant hash function (call the description of this function H) and sends H to $\mathcal{S}_1$ and $\mathcal{S}_2$. Upon receiving input x, $\mathcal{C}$ picks the corresponding input wires in the garbled circuit for x and sends them to $\mathcal{S}_2$.

$\mathcal{S}_1$ generates a garbled circuit for $\mathcal{F}$ using randomness produced by the PRF seed and the output wires given by $\mathcal{C}$. $\mathcal{S}_1$ then computes a hash of this garbled circuit using H and sends the result of this hash to $\mathcal{C}$ along with a proof that the computation was performed honestly (we use the non-private verifiable computation protocol in order to do this). $\mathcal{S}_1$ will send the garbled circuit produced to $\mathcal{S}_2$. $\mathcal{S}_2$ will compute a hash of the garbled circuit received from $\mathcal{S}_1$ and send that to $\mathcal{C}$. $\mathcal{S}_2$ will also evaluate the garbled circuit using the input wires received from $\mathcal{C}$ and send the resulting output wire to $\mathcal{C}$.

The client finally checks three things: a) The non-private verifiable computation with $\mathcal{S}_1$ succeeded, b) the hash output values received from both servers were the same and c) the output wire received from $\mathcal{S}_2$ was indeed a valid output wire. If all three checks succeed, then the client decodes the output from the received output wire and learns $\mathcal{F}(x)$. For more details, we refer the reader to the full version.

Solution 2: We next present a highly practical two-server protocol based solely on one-way functions. For this protocol alone, we will have each server send a message to the other server.

The protocol works as follows. The client sends each server $\mathcal{S}_i$ (for $i \in \{1, 2\}$), a seed to a pseudo-random function K_i. Each $\mathcal{S}_i$ uses K_i and generates the garbled circuit (GC_i) for the function $\mathcal{F}$[9]. Additionally, the client also sends input wires of GC_1 (resp. GC_2), corresponding to his input, to $\mathcal{S}_2$ (resp. $\mathcal{S}_1$). Each server evaluates the garbled circuit it receives from the other server using the input wires it receives from the client and sends the output wires to the client. $\mathcal{C}$ checks the output wires it receives from both servers to make sure they are valid. If they are both valid, it decodes them to obtain the output values contained in them. If both these values are the same, it accepts the output value and rejects otherwise[10]. For more details of this protocol, we refer the reader to the full version.

While the communication between servers cannot be reduced in this protocol, we feel the practical efficiency of this protocol outweighs any overhead caused due to that extra message from $\mathcal{S}_2$ to $\mathcal{S}_1$. Indeed, the only work done by the client is to generate short randomness and finally do a look-up to obtain the output. The only work done by the servers is to generate (and evaluate) a single garbled circuit each (that can also be done in parallel by both the servers). We stress that while 2-PC protocols can be used to obtain a similar result, we need the underlying 2-PC protocol used to be secure against malicious adversaries. Such a 2-PC protocol would need to use either the cut-and-choose approach or zero-knowledge proofs, both of which are inefficient.

[9] Actually $\mathcal{C}$ needs to only give the servers the randomness for generating the input and output wires. The servers can pick their own randomness to generate the garbled circuit (consistent with these input and output wires). Security of our protocol holds even in this case – since we have that at least one server is honest, at least one garbled circuit is generated honestly. This is sufficient to guarantee security.

[10] Note that the above protocol can be modified trivially so that the client sends just one message to $\mathcal{S}_1$ and receives just one message from $\mathcal{S}_2$.

This protocol for verifiable computation is similar in spirit to a protocol by Mohassel and Franklin [MF06] to achieve efficient, malicious, 2-PC in a model where the malicious party may get some information-leakage. Our protocol, used in the context of verifiable computation, is fully secure. It, of course, avoids the use of oblivious transfer protocols and, can additionally allow the servers to run in parallel, thereby achieving better efficiency. One drawback of this solution is that its security is guaranteed only when the servers do not learn whether or not the client accepted the response. We stress that none of our other solutions suffer from this drawback.

4.2 The n-Server Case

In this section, we present our n-server verifiable computation protocol based on the DDH assumption. The high level idea behind constructing such a protocol for functionality $\mathcal{F}(x)$ works as follows: the client generates the input and output wires corresponding to GC_1 (where GC_1 is the garbled circuit for evaluating $\mathcal{F}$). $\mathcal{S}_1$ generates GC_1 (and all the wires corresponding to it). Each server $\mathcal{S}_i$ (for $1 \leq i \leq n-1$), then re-randomizes GC_i and sends it $\mathcal{S}_{i+1}$. The client re-randomizes his input wires ($n-1$ times) to obtain the wires corresponding to input x (according to the re-randomized garbled circuit GC_{n-1}). $\mathcal{S}_n$ obtains the re-randomized input wires corresponding to input x. (NIZK proofs need to be used to ensure that the re-randomizations are done correctly; likewise signature and encryption schemes need to be used to ensure that messages are sent via secure authenticated channels – we omit those details for now.[11]) $\mathcal{S}_n$ then evaluates the final garbled circuit and returns the output to the client. The client re-randomizes his output wires $n-1$ times to obtain the output wires corresponding to GC_{n-1}. Using the work of Gennaro *et al.* [GGP10], one can then show that if $\mathcal{S}_n$ returned a "correct" output wire, then he must have obtained it by evaluating the "honestly" re-randomized garbled circuit on the right input wires – therefore the protocol guarantees soundness. One can then show the privacy of this protocol from the fact that even if one of the servers does the re-randomization honestly, the re-randomized input and output wires will reveal no information to the dishonest servers (i.e., the adversary) about x and $\mathcal{F}(x)$.

Remark. Recently, the work of [BHR12] built adaptively secure garbled circuits which remain secure even if the input is chosen after seeing the garbled circuit. Such security is needed in verifiable computation protocols where the garbled circuit is generated in the pre-processing stage. In our protocol, the garbled circuits are always generated in the online stage. So standard garbled circuits as proven secure in the work of [LP09], suffice for our purposes.

[11] Alternatively, one can use techniques of cut-and-choose in order to make sure that the servers honestly create (or re-randomize) the garbled circuits; we leave the details of this construction to the full version of the paper.

While this, along with a few other ideas, forms the underlying intuition for our result, the main limitation of the above approach is that the client works proportional to n.

To this end, observe that the client works proportional to n because he needs to re-randomize both the input wires as well as the output wires. For the sake of simplicity, for now, we only discuss how to avoid the client's re-randomization of the output wires. One idea to accomplish this is to fix all the output wires (of all garbled circuits) to some specific value. However, this results in two issues. The first issue is that it is not immediately clear that this protocol guarantees privacy. However, we show that Yao's garbled circuit and it's re-randomization remains private even when using fixed output wires. We will use this to show that our protocol guarantees privacy. We refer the reader to the full version for the details.

The next, and more important, issue with this change, is that it no longer guarantees soundness. (Since the servers know the fixed output wires, $\mathcal{S}_n$ could just send a correct output wire without evaluating the garbled circuit GC_{n-1}.) We fix this by using an idea from the work of Applebaum *et al.* [AIK10]. We use a message authentication scheme $\mathsf{MAC} = (\mathsf{MACtag}, \mathsf{MACverify})$ and modify the functionality $\mathcal{F}$ to $\mathcal{G}$: instead of computing just $\mathcal{F}(x)$, $\mathcal{G}$, takes as additional inputs K_1, K_2. $\mathcal{G}(x, K_1, \mathsf{K}_2)$ executes $\mathcal{F}$ on input x to obtain y. It then computes $y \oplus K_1$ and then produces $y_{\mathsf{MAC}} = \mathsf{MACtag}(\mathsf{K}_2, y \oplus K_1)$. Now, one can show that the soundness of the protocol comes from security of the message authentication code. This is an overview of our main construction which we describe below. In this construction, the client still works proportional to n but he no longer re-randomizes the output wires.

In our full version, we describe how to avoid the client's re-randomization of the input wires, thereby making the client's running time independent of n.

Our $n-$Server Construction

Setup stage. During the Key Generation stage, each server $\mathcal{S}_i$ generates the secret key-public key $(\mathsf{sk}_i, \mathsf{pk}_i)$ pairs for an encryption scheme $(\mathsf{KeyGen}_{\mathsf{Enc}}, \mathsf{Enc}, \mathsf{Dec})$ that is CCA2 secure. Further, the client generates $(\mathsf{SK}, \mathsf{VK})$ for the signature scheme $(\mathsf{KeyGen}_{\mathsf{Sign}}, \mathsf{Sign}, \mathsf{Ver})$ that is existentially unforgeable under chosen message attack. The servers $\mathcal{S}_1, \ldots, \mathcal{S}_{n-1}$ generate $(\mathsf{SK}_1, \mathsf{VK}_1), \ldots, (\mathsf{SK}_{n-1}, \mathsf{VK}_{n-1})$ respectively for the signature scheme $(\mathsf{KeyGen}_{\mathsf{Sign}}, \mathsf{Sign}, \mathsf{Ver})$. Let $\mathsf{MAC} = (\mathsf{MACtag}, \mathsf{MACverify})$ be a message authentication scheme which is existentially unforgeable against chosen message attack. MACtag on input a MAC key K and a message m produces a message authentication code m_{MAC} for m. $\mathsf{MACverify}$ on input key K, message m and a tag m'_{MAC}, outputs 1 if m'_{MAC} is a valid message authentication code for m under the key K else it outputs 0. Let $\mathsf{Comm}(m)$ denote the commitment to a message m (that is at least computationally hiding and binding). We let $\mathsf{Open}(c)$ denote the opening of a commitment c. Further, the servers use a non-interactive zero knowledge proof (NIZK) system (in the CRS model) $(\mathsf{Prover}_{\mathsf{Rel}}, \mathsf{Verifier}_{\mathsf{Rel}})$ defined for a relation Rel in NP that satisfies

the standard notions of correctness, soundness, and zero-knowledge. The zero knowledge simulator for this proof system is denoted by $\mathsf{Sim}_{\mathsf{Rel}}$. We also use the following pseudo-random function families:

1. $\mathsf{PRF}_{\mathrm{gc}}(\cdot,\cdot)$ is used by $\mathcal{S}_1$ to output the randomness for generating all the wires of GC_1 with the exception of the input and output wires alone. Without loss of generality, assume that the output length of the PRF is sufficiently long enough to garble the circuit $\mathcal{G}$ [12] which is defined with respect to the delegated function $\mathcal{F}$ as follows. $\mathcal{G}$ on input (x, K_1, K_2) outputs $\mathsf{MACtag}(\mathsf{K}_2, \mathcal{F}(x) \bigoplus K_1)$.
2. For $2 \leq i \leq n-1$, $\mathsf{PRF}_{\mathrm{re}}(\cdot,\cdot)$ is used by $\mathcal{S}_i$ to re-randomize the entire circuit GC_{i-1} except the input wire keys. As before, assume that the output of PRF is sufficiently long enough to rerandomize the garbled circuit of $\mathcal{G}$ (which is defined above).
3. $\mathsf{PRF}_{\mathrm{in}}(\cdot,\cdot)$ is used by $\mathcal{S}_1$ to generate the keys for the input wires corresponding to GC_1. Additionally it will be used by the client to generate the keys for the input wires (without having to generate all of GC_1). Further $\mathcal{S}_i$ (for $2 \leq i \leq n-1$) uses $\mathsf{PRF}_{\mathrm{in}}(\cdot,\cdot)$ to rerandomize the input wire keys of GC_i.

We now describe our protocol $\mathcal{P}$.

1. Client on input x does the following:
   (a) $\mathcal{C}$ picks a key α_1 for $\mathsf{PRF}_{\mathrm{gc}}(\cdot,\cdot)$ and $n-2$ keys $\{\alpha_2, \ldots, \alpha_{n-1}\}$ for the pseudorandom function $\mathsf{PRF}_{\mathrm{re}}(\cdot,\cdot)$ uniformly at random. In addition he also picks keys $\beta_1, \ldots, \beta_{n-1}$ to be used by $\mathcal{S}_i$ to evaluate $\mathsf{PRF}_{\mathrm{in}}(\cdot,\cdot)$ uniformly at random.
   (b) $\mathcal{C}$ computes commitments to each of these PRF keys. Let $\mathbf{c}^\alpha = \{c_1^\alpha, \ldots, c_{n-1}^\alpha\} \stackrel{\text{def}}{=} \{\mathsf{Comm}(\alpha_1), \ldots, \mathsf{Comm}(\alpha_{n-1})\}$ and $\mathbf{c}^\beta = \{c_1^\beta, \ldots, c_{n-1}^\beta\} \stackrel{\text{def}}{=} \{\mathsf{Comm}(\beta_1), \ldots, \mathsf{Comm}(\beta_{n-1})\}$.
   (c) $\mathcal{C}$ sets $d_i^\alpha = \mathsf{Open}(c_i^\alpha)$ and $d_i^\beta = \mathsf{Open}(c_i^\beta)$ for all $1 \leq i \leq n-1$. Let $\mathbf{d}^\alpha = \{d_1^\alpha, \ldots, d_{n-1}^\alpha\}$ and $\mathbf{d}^\beta = \{d_1^\beta, \ldots, d_{n-1}^\beta\}$.
   (d) Let the client's input be $x = x_1 \cdots x_{\lambda_x}$, where each x_i is a bit. The client picks K_1 uniformly at random (where K_1 is of the same length as $\mathcal{F}(x)$) and also picks a MAC key K_2. Let K_2 be of length λ_{K_2}. λ is such that $\lambda = \lambda_x + \lambda_{\mathsf{K}_1} + \lambda_{\mathsf{K}_2}$.
   (e) $\mathcal{C}$ picks an execution id, id[13], and obtains the keys for the input wires of the garbled circuit GC_1 (to be defined later) by eval-

[12] This assumption requires the knowledge of the size of the circuit being delegated by the client before the PRF keys are generated. This in turn makes the key generation stage dependent on the function being delegated. This dependency can be eliminated as follows. Instead of using just one output of PRF to garble the circuit, use multiple PRF outputs to garble the circuit. Using sufficiently many PRF outputs the entire circuit can be garbled. For convenience sake, in our protocol description the garbling of the entire circuit is done using just one output of the PRF.

[13] This id needs to be unique for each execution. This can be achieved by the client, either by maintaining state and ensuring that ids do not repeat, or by the client picking the id at random from a sufficiently large domain (and one can then argue that except with negligible probability, the id will be unique).

uating $\mathsf{YaoGarbledCkt}_{\mathsf{in}}(\mathcal{F}, \mathsf{PRF}_{\mathrm{in}}(\beta_1, \mathsf{id}))$. Let the keys (corresponding to 0 and 1) for the input wires be denoted by $\mathbf{w}_{\mathsf{GC}_1,\mathsf{in}} = \{(w^0_{\mathsf{GC}_1,\mathsf{in},1}, w^1_{\mathsf{GC}_1,\mathsf{in},1}), \ldots, (w^0_{\mathsf{GC}_1,\mathsf{in},\lambda}, w^1_{\mathsf{GC}_1,\mathsf{in},\lambda})\}$ where $w^0_{\mathsf{GC}_1,\mathsf{in},i}$ denotes the key for the i^{th} input wire representing bit 0 while $w^1_{\mathsf{GC}_1,\mathsf{in},i}$ denotes the i^{th} wire representing the bit 1 in the garbled circuit GC_1.

(f) The client $\mathcal{C}$ then does the following. It computes $\mathsf{reRand}_{\mathsf{in}}\Big(\mathsf{reRand}_{\mathsf{in}}\big(\cdots\big(\mathsf{reRand}_{\mathsf{in}}(\mathbf{w}_{\mathsf{GC}_1,\mathsf{in}}; \mathsf{PRF}_{\mathrm{in}}(\beta_2, \mathsf{id}))\big); \cdots \big); \mathsf{PRF}_{\mathrm{in}}(\beta_{n-1}, \mathsf{id})\Big)$ to obtain $\mathbf{w}_{\mathsf{GC}_{n-1},\mathsf{in}} = ((w^0_{\mathsf{GC}_{n-1},\mathsf{in},1}, w^1_{\mathsf{GC}_{n-1},\mathsf{in},1}), \cdots, (w^0_{\mathsf{GC}_{n-1},\mathsf{in},\lambda}, w^1_{\mathsf{GC}_{n-1},\mathsf{in},\lambda}))$. Let $\mathbf{w}^X_{\mathsf{GC}_{n-1},\mathsf{in}} = (w^{X_0}_{\mathsf{GC}_{n-1},\mathsf{in},1}, \ldots, w^{X_\lambda}_{\mathsf{GC}_{n-1},\mathsf{in},\lambda})$ denote the input wire keys corresponding to the input $X = (x, K_1, \mathsf{K}_2)$ for the garbled circuit GC_{n-1}.

(g) Client $\mathcal{C}$ picks the output wire keys $w_{\mathsf{out}} = \{(w^0_{\mathsf{out},1}, w^1_{\mathsf{out},1}), \ldots, (w^0_{\mathsf{out},\mu}, w^1_{\mathsf{out},\mu})\}$. For simplicity we assume that these are chosen uniformly at random, even though we won't rely on that property in any of our proofs.

(h) Client $\mathcal{C}$ picks random strings $\mathsf{CRS}_1, \ldots, \mathsf{CRS}_{n-1}$ to be used as common reference string for the NIZK proofs.

(i) For $1 \leq i \leq n-1$, $\mathcal{C}$ sets $\mathsf{msg}_i = (\mathsf{id}, d^\alpha_i, d^\beta_i, \mathbf{c}^\alpha, \mathbf{c}^\beta, \mathsf{CRS}_1, \ldots, \mathsf{CRS}_i, \mathbf{w}_{\mathsf{out}})$. Further, $\mathcal{C}$ sets $\mathsf{msg}_n = (\mathsf{id}, \mathbf{c}^\alpha, \mathbf{c}^\beta, \mathbf{w}^X_{\mathsf{GC}_{n-1},\mathsf{in}}, \mathsf{CRS}_1, \ldots, \mathsf{CRS}_{n-1}, \mathbf{w}_{\mathsf{out}})$.

(j) Let $\sigma_i^{\mathsf{msg}_i}$ be the signature of $\mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{msg}_i)$ using signing key SK for all $1 \leq i \leq n$.

(k) $\mathcal{C}$ sends $\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{msg}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}_n}(\mathsf{msg}_n)$ along with $\sigma_1^{\mathsf{msg}_1}, \ldots, \sigma_n^{\mathsf{msg}_n}$ to $\mathcal{S}_1$.

2. Server $\mathcal{S}_1$ on input $\mathcal{F}$ and upon receiving $(\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{msg}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}_n}(\mathsf{msg}_n), \sigma_1^{\mathsf{msg}_1}, \ldots, \sigma_n^{\mathsf{msg}_n})$ from the client does the following:

(a) Compute the modified functionality $\mathcal{G}$ which does the following. $\mathcal{G}$ on input (x, K_1, K_2) executes $\mathcal{F}$ on input x to obtain y. It then computes $y \oplus K_1$ and then produces $y_{\mathsf{MAC}} = \mathsf{MACtag}(\mathsf{K}_2, y \oplus K_1)$. It outputs (y, y_{MAC}).

(b) $\mathcal{S}_1$ verifies signature $\sigma_1^{\mathsf{msg}_1}$ on the input message $\mathsf{Enc}_{pk_1}(\mathsf{msg}_1)$ by executing $\mathsf{Ver}(\mathsf{VK}, \mathsf{Enc}_{pk_1}(\mathsf{msg}_1), \sigma_1^{\mathsf{msg}_1})$. If Ver outputs reject then it aborts.

(c) $\mathcal{S}_1$ decrypts $\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{msg}_1)$ using sk_1 to obtain msg_1 which is parsed as $(\mathsf{id}, d^\alpha_1, d^\beta_1, \mathbf{c}^\alpha, \mathbf{c}^\beta, \mathsf{CRS}_1, \mathbf{w}_{\mathsf{out}})$.

(d) $\mathcal{S}_1$ evaluates $\mathsf{PRF}_{\mathrm{gc}}(\alpha_1, \mathsf{id})$ and $\mathsf{PRF}_{\mathrm{in}}(\beta_1, \mathsf{id})$ and uses the randomness output by the two PRFs to compute $\mathsf{YaoGarbledCkt}$ on input $\mathcal{G}$, to obtain GC_1. In other words,
$\mathsf{YaoGarbledCkt}(\mathcal{F}, \mathbf{w}_{\mathsf{out}}; (\mathsf{PRF}_{\mathrm{in}}(\beta_1, \mathsf{id}), \mathsf{PRF}_{\mathrm{gc}}(\alpha_1, \mathsf{id})))$ outputs GC_1 as well as the input wires corresponding to GC_1.

(e) $\mathcal{S}_1$ then computes a proof π_1 using CRS_1 as the CRS for the statement: "There exists witness d^α_1, d^β_1 such that
   i. $d^\alpha_1 = \mathsf{Open}(c^\alpha_1)$ and $d^\beta_1 = \mathsf{Open}(c^\beta_1)$;
   ii. GC_1 is the garbled circuit output by $\mathsf{YaoGarbledCkt}(\mathcal{G}; (\mathsf{PRF}_{\mathrm{in}}(\beta_1, \mathsf{id}), \mathsf{PRF}_{\mathrm{gc}}(\alpha_1, \mathsf{id})))$."

More formally, the proof is generated as follows. Consider the following relation:

$$\mathsf{Rel}_1 = \Big\{ \big((c_1^\alpha, c_1^\beta, \mathsf{GC}_1), (d_1^\alpha, d_1^\beta)\big) \ : \ d_1^\alpha = \mathsf{Open}(c_1^\alpha), d_1^\beta = \mathsf{Open}(c_1^\beta), d_1^\alpha = (\alpha_1, R_1^\alpha),\ d_1^\beta = (\beta_1, R_1^\beta), \mathsf{GC}_1 = \mathsf{YaoGarbledCkt}((\mathcal{G}, \mathbf{w}_{\mathsf{out}}); (\mathsf{PRF}_{\mathrm{in}}(\beta_1, \mathsf{id}), \mathsf{PRF}_{\mathrm{gc}}(\alpha_1, \mathsf{id})))\Big\}$$

Execute $\mathsf{Prover}_{\mathsf{Rel}_1}\big((c_1^\alpha, c_1^\beta, \mathsf{GC}_1), (d_1^\alpha, d_1^\beta)\big)$ to obtain the proof π_1.

(f) Generate signature $\sigma_{\mathcal{S}_1}$ for the message (GC_1, π_1).

(g) $\mathcal{S}_1$ lets $\pi = \{\pi_1\}$, and gives $(\mathsf{GC}_1, \pi, \mathsf{Enc}_{pk_2}(\mathsf{msg}_2), \ \ldots, \ \mathsf{Enc}_{pk_n}(\mathsf{msg}_n), \sigma_{\mathcal{S}_1}, \sigma_2^{\mathsf{msg}_2}, \ \ldots, \ \sigma_n^{\mathsf{msg}_n})$ to $\mathcal{S}_2$.

3. Server $\mathcal{S}_i$ $(2 \leq i \leq n-1)$ upon receiving $\mathcal{F}$ and $(\mathsf{GC}_1, \ldots, \mathsf{GC}_{i-1}, \pi, \mathsf{Enc}_{pk_i}(\mathsf{msg}_i), \ \ldots, \ \mathsf{Enc}_{pk_n}(\mathsf{msg}_n), \ \sigma_{\mathcal{S}_1}, \ \ldots, \ \sigma_{\mathcal{S}_{i-1}}, \sigma_i^{\mathsf{msg}_i}, \ \ldots, \ \sigma_n^{\mathsf{msg}_n})$ from $\mathcal{S}_{i-1}$ does the following:

(a) $\mathcal{S}_i$ verifies signature $\sigma_i^{\mathsf{msg}_i}$ on the input message $\mathsf{Enc}_{pk_i}(\mathsf{msg}_i)$ by executing $\mathsf{Ver}(\mathsf{VK}, \mathsf{Enc}_{pk_i}(\mathsf{msg}_i), \sigma_i^{\mathsf{msg}_i})$. If Ver outputs reject then it aborts.

(b) $\mathcal{S}_i$ parses π as $\pi_1, \ldots, \pi_{i-1}$. It then verifies signatures $\sigma_{\mathcal{S}_1}, \ldots, \sigma_{\mathcal{S}_{i-1}}$ on the messages $(\mathsf{GC}_1, \pi_1), \ldots, (\mathsf{GC}_{i-1}, \pi_{i-1})$ using the verification keys $\mathsf{VK}_1, \ldots, \mathsf{VK}_{i-1}$ respectively.

(c) $\mathcal{S}_i$ then decrypts $\mathsf{Enc}_{pk_i}(\mathsf{msg}_i)$ using secret key sk_i to obtain msg_i which is parsed as $(\mathsf{id}, d_i^\alpha, d_i^\beta, \mathbf{c}^\alpha, \mathbf{c}^\beta, \mathsf{CRS}_1, \ldots, \mathsf{CRS}_i, \mathbf{w}_{\mathsf{out}})$.

(d) $\mathcal{S}_i$ verifies all the NIZK proofs in π as follows. It first parses π as $\pi_1, \ldots, \pi_{i-1}$. It then executes $\mathsf{Verifier}_1((c_1^\alpha, c_1^\beta, \mathsf{GC}_1), \mathsf{CRS}_1, \pi_1)$ and $\mathsf{Verifier}_j((c_j^\alpha, c_j^\beta, \mathsf{GC}_j, \mathsf{GC}_{j-1}), \mathsf{CRS}_j, \pi_j)$ for all $2 \leq j \leq i-1$. $\mathcal{S}_i$ aborts if any of the verifiers $\mathsf{Verifier}_j$, for $1 \leq j \leq i-1$, aborts.

(e) $\mathcal{S}_i$ evaluates $\mathsf{PRF}_{\mathrm{re}}(\alpha_i, \mathsf{id})$ and $\mathsf{PRF}_{\mathrm{in}}(\beta_i, \mathsf{id})$ and uses the randomness output by the 2 PRFs to rerandomize the garbled circuit GC_{i-1}. More formally, it computes $\mathsf{reRand}(\mathsf{GC}_{i-1}; (\mathsf{PRF}_{\mathrm{in}}(\beta_i, \mathsf{id}), \mathsf{PRF}_{\mathrm{re}}(\alpha_i, \mathsf{id})))$ to obtain GC_i.

(f) $\mathcal{S}_i$ computes a proof π_i with respect to CRS_i for the statement:
"There exists witness d_i^α and d_i^β such that
i. $d_i^\alpha = \mathsf{Open}(c_i^\alpha)$ and $d_i^\beta = \mathsf{Open}(c_i^\beta)$;
ii. GC_i is the garbled circuit output by $\mathsf{reRand}(\mathsf{GC}_{i-1}; (\mathsf{PRF}_{\mathrm{in}}(\beta_i, \mathsf{id}), \mathsf{PRF}_{\mathrm{re}}(\alpha_i, \mathsf{id})))$."
More formally, consider the following relation:

$$\mathsf{Rel}_1 = \Big\{ \big((c_i^\alpha, c_i^\beta, \mathsf{GC}_i, \mathsf{GC}_{i-1}), (d_i^\alpha, d_i^\beta)\big) \ : \ d_i^\alpha = \mathsf{Open}(c_i^\alpha), d_i^\beta = \mathsf{Open}(c_i^\beta), d_i^\alpha = (\alpha_i, R_i^\alpha),\ d_i^\beta = (\beta_i, R_i^\beta), \mathsf{GC}_i = \mathsf{reRand}(\mathsf{GC}_{i-1}; (\mathsf{PRF}_{\mathrm{in}}(\beta_i, \mathsf{id}), \mathsf{PRF}_{\mathrm{re}}(\alpha_i, \mathsf{id})))\Big\}$$

Execute $\mathsf{Prover}_{\mathsf{Rel}_i}\big((c_i^\alpha, c_i^\beta, \mathsf{GC}_i), (d_i^\alpha, d_i^\beta)\big)$ to obtain the proof π_i.

(g) Generate signature $\sigma_{\mathcal{S}_i}$ for the message (GC_i, π_i).
(h) $\mathcal{S}_i$ lets $\pi = \pi \cup \{\pi_i\}$, and sends $(\mathsf{GC}_1, \ldots, \mathsf{GC}_i, \mathsf{Enc}_{pk_{i+1}}(\mathsf{msg}_{i+1}), \ldots, \mathsf{Enc}_{pk_n}(\mathsf{msg}_n), \sigma_{\mathcal{S}_1}, \ldots, \sigma_{\mathcal{S}_i}, \sigma_{i+1}^{\mathsf{msg}_{i+1}}, \ldots, \sigma_n^{\mathsf{msg}_n})$ to $\mathcal{S}_{i+1}$.

4. Server $\mathcal{S}_n$ does the following upon receiving $\mathcal{F}$ and $(\mathsf{GC}_1, \ldots, \mathsf{GC}_{n-1}, \pi, \mathsf{Enc}_{pk_n}(\mathsf{msg}_n), \sigma_{\mathcal{S}_1}, \ldots, \sigma_{\mathcal{S}_{n-1}}, \sigma_n^{\mathsf{msg}_n})$:
   (a) $\mathcal{S}_n$ verifies signature $\sigma_n^{\mathsf{msg}_n}$ on the input message $\mathsf{Enc}_{pk_n}(\mathsf{msg}_n)$ by executing $\mathsf{Ver}(\mathsf{VK}, \mathsf{Enc}_{pk_n}(\mathsf{msg}_n), \sigma_n^{\mathsf{msg}_n})$. If Ver outputs reject then it aborts.
   (b) $\mathcal{S}_i$ parses π as $\pi_1, \ldots, \pi_{n-1}$. It then verifies signatures $\sigma_{\mathcal{S}_1}, \ldots, \sigma_{\mathcal{S}_{n-1}}$ on the messages $(\mathsf{GC}_1, \pi_1), \ldots, (\mathsf{GC}_{n-1}, \pi_{n-1})$ using the verification keys $\mathsf{VK}_1, \ldots, \mathsf{VK}_{n-1}$ respectively.
   (c) $\mathcal{S}_n$ then decrypts $\mathsf{Enc}_{pk_n}(\mathsf{msg}_n)$ using secret key sk_i to obtain msg_i which is parsed as $(\mathsf{id}, \mathbf{c}^\alpha, \mathbf{c}^\beta, \mathbf{w}^X_{\mathsf{GC}_{n-1},\mathsf{in}}, \mathsf{CRS}_1, \ldots, \mathsf{CRS}_{n-1}, \mathbf{w}_{\mathsf{out}})$. Further, $\mathbf{w}^X_{\mathsf{GC}_{n-1},\mathsf{in}}$ is parsed as $(w^{X_1}_{\mathsf{GC}_{n-1},\mathsf{in},1}, \ldots, w^{X_\lambda}_{\mathsf{GC}_{n-1},\mathsf{in},\lambda})$.
   (d) $\mathcal{S}_n$ verifies all the NIZK proofs in π as follows. It first parses π as $\pi_1, \ldots, \pi_{n-1}$. It then executes $\mathsf{Verifier}_1((c_1^\alpha, c_1^\beta, \mathsf{GC}_1), \mathsf{CRS}, \pi_1)$ and $\mathsf{Verifier}_j((c_j^\alpha, c_j^\beta, \mathsf{GC}_j, \mathsf{GC}_{j-1}), \mathsf{CRS}, \pi_j)$ for all $2 \leq j \leq n-1$. $\mathcal{S}_n$ aborts if any of the verifiers $\mathsf{Verifier}_j$, for $1 \leq j \leq n-1$, aborts.
   (e) If $\mathcal{S}_n$ accepts all the NIZK proofs and signatures, it uses $\mathbf{w}^X_{\mathsf{GC}_{n-1}}$ to evaluate the garbled circuit GC_{n-1} to obtain the wire keys $\mathbf{w}^z_{\mathsf{out}}$. It then determines $\mathbf{z} = z_1 \cdots z_{|\mathcal{G}_{\mathsf{out}}|}$ [14] such that the set of wire keys $\{w_{\mathsf{out},1}, \ldots, w_{\mathsf{out},|\mathcal{G}_{\mathsf{out}}|}\}$ represents $\mathbf{w}^z_{\mathsf{out}}$. $\mathcal{S}_n$ sends $\mathbf{z}$ to the client $\mathcal{C}$.

5. Client on receiving $\mathbf{z}$ from $\mathcal{S}_n$ does the following:
   (a) $\mathcal{C}$ parses $\mathbf{z}$ as (y, y_{MAC}).
   (b) $\mathcal{C}$ executes $\mathsf{MACverify}_{\mathsf{K}_2}(y, y_{\mathsf{MAC}})$. If the output of $\mathsf{MACverify}$ is 0 then it outputs Reject. Else, it computes y' where $y' = y \oplus K_1$ and then it outputs Accept.

It is easy to see that correctness follows from the correctness of Yao and other underlying primitives. We defer the proof of privacy and soundness to the full version.

References

ACG+14. Prabhanjan Ananth, Nishanth Chandran, Vipul Goyal, Bhavana Kanukurthi, and Rafail Ostrovsky. Achieving privacy in verifiable computation with multiple servers – without fhe and without pre-processing. *IACR Cryptology ePrint Archive*, 2014, 2014.

AIK10. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 152–163, 2010.

BBS98. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144. Springer, 1998.

[14] $|\mathcal{G}_{\mathsf{out}}|$ denotes the length of the output of the function $\mathcal{G}$.

BCCT12. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, New York, NY, USA, 2012. ACM.

BGV11. Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, pages 134–153, 2012.

BV13. Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. Cryptology ePrint Archive, Report 2013/541, 2013. `http://eprint.iacr.org/`.

CKV10. Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010.

Cra12. Ronald Cramer, editor. *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*. Springer, 2012.

CRR11. Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 445–454. ACM, 2011.

CRR12. Ran Canetti, Ben Riva, and Guy N. Rothblum. Two protocols for delegation of computation. In *ICITS*, pages 37–61, 2012.

DFH12. Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Cramer [Cra12], pages 54–74.

FG12. Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM Conference on Computer and Communications Security*, pages 501–512, 2012.

GGH+13. Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.

GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 465–482, 2010.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Eurocrypt*, 2013.

GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 155–172, 2010.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Overcoming the worst-case curse for cryptographic constructions. In *CRYPTO*, 2013.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *STOC*, pages 113–122. ACM, 2008.

GLR11. Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

GVW13. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

KR09. Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, pages 143–159, 2009.

KR11. Seny Kama and Mariana Raykova. Secure outsourced computation in a multi-tenant cloud. Workshop on Cryptography and Security in the Clouds, 2011.

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

MF06. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

PRV12. Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Cramer [Cra12], pages 422–439.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science (FOCS), Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.