Refereed Delegation of Computation*

Ran Canetti Ben Riva
Tel Aviv University
{canetti, benriva}@tau.ac.il

Guy N. Rothblum
Princeton University
rothblum@alum.mit.edu

Abstract

We consider a weak client that wishes to learn and verify the result of an expensive computation. When the client uses only a single untrusted server, current techniques suffer from disadvantages such as computational inefficiency for the client or the server, limited functionality, or high round complexity. We demonstrate relatively efficient and general solutions where the client delegates the computation to *several* servers, and is guaranteed to determine the correct answer as long as even a *single* server is honest. We call such protocols Refereed Delegation of Computation (RDoC) and show:

1. A 1-round (2-messages) unconditionally-sound RDoC for any function computable in log-space uniform $\mathcal{NC}$, assuming the existence of private communication channels.
2. A potentially practical computationally-sound RDoC for any efficiently computable function, with logarithmically many rounds, based on any collision-resistant hash family.

In both protocols the servers incur only a polynomial overhead relative to simply computing the function and the client is at most quasi-linear in the input length. These protocols adapt techniques from the works of Feige and Kilian [STOC 1997] and Goldwasser, Kalai and Rothblum [STOC 2008].

KEYWORDS: Refereed Games, Verifiable Computation, Delegation of Computation.

*Research supported by the Check Point Institute for Information Security.

1 Introduction

An emerging paradigm in modern computing is *pay-per-use* Cloud Computing. As companies and users reduce their computing assets and turn to weaker computing devices, an increasing number and variety of computations are being performed remotely by untrusted parties that may be error-prone or even malicious.

This shift motivates exploring methods for delegating computations reliably: a weak client delegates his computation to a powerful server. After the server returns the result of the computation, the client should be able to verify the correctness of that result using considerably *less* resources than required to actually *perform* the computation from scratch.

Much attention was given to the case where the delegator (the client) interacts only with a single untrusted worker (server), most notably the concepts of Computationally-Sound proofs [Mic00] and delegated computation [GKR08]. See Appendix A for a more complete list. However, these works either incur high (although polynomial) overhead on top of the original computation (e.g. the use of PCPs in [Mic00]), are limited in their applicability or require a large number of communication rounds (e.g. [GKR08]). Furthermore, to some extent, the use of PCPs seems inherent [RV10].

We consider a natural extension of this model, where the client interacts with *several* servers. Indeed, this relaxation allows some very simple solutions: If one is willing to interact with three or more servers and assume that a majority of the servers are honest, then it suffices to simply ask each server for the answer and take the majority answer. Even if we assume there is only one honest server, the client can still detect inconsistency between the answers. Then, in case of inconsistency, he has to compute the function by himself. But can one do better? In particular, can we get efficiency improvements over the single server case, with access to *two* (or more) servers, only one of which is honest, and for a client that can *not* compute the function by himself?

That is, we are interested in the following model. The client asks for the value of $f(x)$ from several servers. In case they make contradictory claims about $f(x)$, they "play" against each other in a protocol where the weak client can efficiently determine the true claim as long as there is at least *one* honest server. As for the efficiency, we require that the computational requirements from an honest server are not much more than those required to compute the function in the first place, and that the client's running time would be much smaller than required to compute the function. We call this model Refereed Delegation of Computation (RDoC).

Since the client actually acts as a referee, in the rest of this work we use the terms referee and servers. (For protocols with a single server, as in the *interactive-proof* model, we stay with the terms client and server.) Note that a server is a player in the protocol, who may send arbitrary messages, and an *honest* server is the algorithm that a given server is *supposed* to run in the protocol (and dishonest means that the server does deviate from its algorithm).

A closely related model to ours is the *Refereed Games (RG)* model of Feige and Kilian [FK97] where they focus on two unbounded competing servers and polynomial time referee. However, we are faced with the additional challenges of building protocols with efficient honest servers, with super-efficient client and for any number of servers. Indeed, our model can be considered also as *refereed games with efficient servers* and super-efficient clients.

1.1 Our Results

For the description here we restrict attention to the case when there are exactly two servers, one honest and one malicious (but the referee/client does not know which is honest). We later show how to extend our protocols for more than two servers.

We show two new protocols for polynomial-time computations. Both of these protocols have honest servers that are polynomial in the time to compute the function in question, and referees that are *quasi-linear* in the input size. More specifically, we show two protocols:

Protocol I. A 1-round (2-message) unconditionally-sound RDoC for any function computable in $\mathcal{L}$-uniform $\mathcal{NC}$. To the best of our knowledge, all previous single-round protocols for reliably delegating computation in the single server model [Mic00, GKR08, GGP10, CKV10] require cryptographic assumptions and provide only computational soundness.

We note that a 1-round unconditionally sound interactive proof for delegation of computation can be achieved in the multi-prover model (MIP). However, in the MIP model, soundness is guaranteed only if no two malicious provers/servers can communicate or coordinate their strategies during the protocol. We believe that this assumption is less realistic for cloud computing. Moreover, the definition of MIP allows cases where even a single malicious prover/server can cause the verifier/client to reject the proof of valid statements.

Our protocol adapts techniques from the work of Feige and Kilian [FK97], who construct a refereed game (or RDoC) but with honest servers that are inefficient even for log-space computations, along with ideas and techniques from the work of Goldwasser, Kalai and Rothblum [GKR08], and some new techniques.

At high level, our protocol follows the structure of the [GKR08] interactive proof. We view the computation as a circuit. The servers make claims about the output layer of the circuit, and we use a (very efficient) sum-check protocol to reduce a claim about a high layer in the circuit, which we call an input claim, into a claim about a lower layer (closer to the circuit's input layer), we call this an output claim. The guarantee is that if the input claim is false, then w.h.p. over the referee's coins the output claim will also be false. They use this sub-protocol to reduce the claim about the circuit's output layer into a claim about the circuit's input layer, and complete the protocol by noting that claims about the input layer can be verified by the referee in quasi-linear time.

However, the [GKR08] protocol is highly interactive: First, each sum-check sub-protocol requires a logarithmic number of rounds. Second, the claim for each layer in the circuit depends on the coins chosen by the referee in the sum-check for the layer above it, so all of these sum check protocols must be run sequentially from the top circuit output layer to the bottom circuit input layer. To eliminate the first source of interaction, we use a variant of the one-round refereed game for the sum check test from [FK97]. This still leaves us with a significant technical obstacle: How can we collapse all of sum-check protocols from the different layers into just one round of interaction? This is challenging, because in order to run the [FK97] protocol, both servers need to know the claim being debated, but this claim depends on the referee's (non-public) coins in the sum check for the layer above. Revealing all of those coins to both servers ahead of time would compromise soundness.

We overcome this obstacle (and additional lower-level ones) using techniques tailored to our setting. In a nutshell, the claim for each layer is the value of a low-degree multi-variate polynomial (say p) on a certain secret point (say z) that is known only to the referee. The referee sends to each server a different low-degree parametric curve passing through the point z (but also through many others), and asks for the (low-degree) polynomial q describing p restricted to that server's curve. Essentially, soundness follows because each server (on its own) cannot tell which of the points on its curve is the one that the referee will be checking. If the server cheats and sends $q' \neq q$, then (since q and q' are low degree polynomials over a larger field) with high probability the server must be cheating on the point z that the referee is checking on.

Protocol II. A potentially practical and *full-information* computationally-sound RDoC protocol for any efficiently computable function, with logarithmically many rounds, based on any collision-resistant hash function family. Here, by full information we mean that the servers can see the full internal state of the referee and the communication between the referee and the servers is public. The honest servers' work grows only *quasi-linearly* with the complexity of the computation. This protocol is highly generic and can work with any reasonable computation model. Specifically, we describe it with Turing Machines (TM) but it can be adapted for real-world models, which can potentially lead to very efficient implementations. Previously,

Feige and Kilian [FK97] gave a *private information* but unconditionally sound protocol with similar parameters. We note that it follows from their results that it is unlikely that an information-theoretically sound full-information protocol with similar performance can be obtained (in particular, this is impossible unless all of $\mathcal{P}$ can be computed in poly-logarithmic space).

Our protocol builds directly on the protocol of Feige and Kilian. This new protocol seems to be qualitatively more practical than known techniques for delegating computation in the single-prover setting. In particular, all known protocols rely either on arithmetization and PCP techniques [Mic00, GKR08], or provide only amortized performance advantages and rely on fully homomorphic encryption [GGP10, CKV10]. Moreover, all known protocols work with the (arguably less practical) circuit representation of the computation.

At high level, in this protocol the referee searches (using binary search) for inconsistencies between the intermediate states of the two servers' computations. On finding an inconsistency, the referee can detect the cheater by performing only a *single step* of the delegated TM. The collision-resistant hash functions are used to allow the servers to "commit" to the (large) intermediate internal states of the computation using small commitments.

In addition, in a setting where messages between referees and servers are digitally signed, the protocol guarantees that if one server cheats, the referee detects the cheating and obtains a publicly verifiable proof of this fact. This is a strong guarantee: we view the servers as *rational* self-interested parties (say cloud computing service providers). An honest server can convince even third parties that all of the cheating servers are cheaters. Assuming that pointing out cheaters is rewarded and cheating is penalized, playing honestly becomes (always) a dominant strategy for rational servers.

1.2 Organization

Section 2 defines the model of refereed delegation of computation, shows that parallel repetition of RDoC protocols reduces the error in much the same way as it reduces the soundness error in plain interactive proofs, and describes how to extend RDoC with two servers to any number of servers. Section 3 reviews the techniques of [FK97] and [GKR08] and presents the construction of one-round RDoC for any $\mathcal{L}$-uniform $\mathcal{NC}$ computation. Section 4 shows the construction of computationally sound RDoC for any polynomial time computation and Appendix G presents several possible extensions of this construction. Appendix A reviews prior work.

2 Refereed Delegation of Computation

A refereed delegation of computation for a function f is a protocol between a referee R and N servers $P_1, P_2, \ldots, P_N$. All parties may use local randomness. The referee and the servers receive an input x. The servers claim different results for the computation of $f(x)$ and the referee should be able to determine the correct answer with high probability. We assume that at least one of the servers is honest.

Definition 1 (Refereed Delegation of Computation)**.** *Let $(P_1, P_2, \ldots, P_N, R)$ be an ε-RDoC with N servers for a function f if the following holds:*

- *For any input x, if server P_i is honest then for any $P_1^*, \ldots, P_{i-1}^*, P_{i+1}^*, \ldots, P_N^*$ the output of R is $f(x)$ w.p. at least $1 - \varepsilon$.*
- *The complexity of the referee is at most quasi-linear in $|x|$ and the complexity of the (honest) servers is polynomial in the complexity of evaluating f.*

If soundness holds only for polynomially bounded (in $|x|$) servers then we say that it is a *computationally sound RDoC*. Furthermore, if the referee starts by sending all its local random choices to *all* servers, and if all the communication between the referee and the servers is public, we call it a *full-information* RDoC.

For completeness of the description, we briefly review the model of Refereed Games [FK97]. A refereed game (RG) for a language L is a protocol between a referee R and two competing *unbounded* servers P_1 and P_2. All three parties may use local randomness. The referee and the servers receive $x \in \{0,1\}^*$. Without loss of generality we can assume P_1 claims that $x \in L$ and P_2 claims that $x \notin L$, and the referee should be able to determine the correct answer with probability at least 2/3.

2.1 Parallel Repetition for RDoC

We have the following "parallel repetition" theorem for RDoC for boolean functions.

Theorem 2 (Parallel Repetition for RDoC)**.** *Let $(P_1, P_2, \ldots, P_N, R)$ be a ε-RDoC for a boolean function f, and let $(P_1^k, P_2^k, \ldots, P_N^k, R^k)$ be a RDoC obtained by running $(P_1, P_2, \ldots, P_N, R)$ k times in parallel and in which R^k accepts if and only if R accepted in the majority of the executions. Then, $(P_1^k, P_2^k, \ldots, P_N^k, R^k)$ is a RDoC with error probability $\varepsilon^{poly(k)}$.*

Proof (sketch). We use the fact that parallel repetition reduces the error probability of any interactive proof system, and we build an interactive proof system (P, V) for the language $L = \{\ x \mid f(x) = 1\ \}$ from our RDoC $(P_1, P_2, \ldots, P_N, R)$. Without loss of generality, we assume $x \in L$ and P_1 is an honest server. We view the referee R and the honest server P_1 as the verifier V, and the other servers as the prover P. Similarly, we view P_1^k and R^k as the verifier V^k in the parallel repetition version of (P, V). Since $(P_1, P_2, \ldots, P_N, R)$ is a RDoC, the soundness of (P, V) is bounded by ε. Now, if we assume there are malicious servers $P_2^k, \ldots, P_N^k$ that convince the referee in $(P_1^k, P_2^k, \ldots, P_N^k, R^k)$ with probability p, it means there is a prover P^k that can convince V^k with probability p. However, since the parallel repetition of interactive proofs reduced the error probability to $\varepsilon^{poly(l)}$, p is negligible. □

A similar lemma can be proved for computationally sound RDoC with three rounds using the results of Bellare *et al.* [BIN97] and Canetti *et al.* [CHS05].

2.2 From Two Servers to N Servers

In the next sections we show protocols for RDoC with two servers. Here we show how, given a RDoC with two servers and negligible error probability, one can construct a RDoC with N servers and negligible error probability, where we only need to assume that at least *one* of them is honest. The idea is to execute the RDoC with two servers between each pair of servers. By the soundness of the RDoC with two servers, with high probability there exists an honest server P_i that convinces the referee in *all* of his "games". The referee outputs the claimed result of P_i.

This solution can be executed in parallel for all pairs, and therefore keeps the number of rounds the same. However, it requires $\frac{N \cdot (N-1)}{2}$ different executions of the protocol.

3 One-round RDoC for Any $\mathcal{L}$-uniform $\mathcal{NC}$ Computation

We start by describing the two protocols we base our construction upon. We outline the intuition behind the one-round refereed game for the sum-check task of [FK97]. (In Appendix C we present the detailed protocol.). And, we present the protocol of [GKR08].

3.1 Preliminaries

For completeness, in Appendix B we review the definition of a low degree extension of a function.

3.1.1 The Protocol of [FK97]

We present a variant of the one-round refereed game from [FK97] for the sum-check task. In this task we have a finite field F, a subset of F denoted by H, a fixed number k and a multivariate polynomial $f : F^k \to F$ of degree $\leq d$ in each variable.[1] The referee can evaluate f by himself in polynomial time in

[1] The [FK97] protocol considers $f : \{0,1\}^k \to F$. We extend it to $f : F^k \to F$.

the size of f. Server 1 claims that

$$\sum_{x_1,x_2,\ldots,x_k \in H} f(x_1, x_2, \ldots, x_k) = N_0$$

for some value N_0 and Server 2 claims otherwise (denote this value by N_0').

Lund *et al.* [LFKN92] show an interactive proof with one sever for the sum-check task. Their protocol requires k rounds. In the first round, the server sends to the client the univariate polynomial $g_1(x) = \sum_{x_2,\ldots,x_k \in H} f(x, x_2, \ldots, x_k)$ and the client checks if $\sum_{x \in H} g_1(x) = N_0$. Then, the client chooses a random element $c_1 \in F$ and sends it to the server. The protocol continues to the next rounds, where in round i (for $i \in [2..k]$) the server sends to the client $g_i(x) = \sum_{x_i,\ldots,x_k \in H} f(c_1, \ldots, c_{i-1}, x, x_{i+1} \ldots, x_k)$ and client checks if $\sum_{x \in H} g_i(x) = g_{i-1}(c_{i-1})$. Then, the client chooses another random element $c_i \in F$ and sends it to the server. In the last round, the client does not send c_k to the server. Instead, he computes $f(c_1, \ldots, c_k)$ by himself, and checks whether it equals to $g_k(c_k)$. Note that the correctness of the protocol requires that the server cannot guess the c_i-s in advance as they are randomly chosen by the client. Actually, this is why the protocol requires k rounds. If the client would have send all the c_i-s in one round, the server could easily cheat.

In order to reduce the number of rounds, the protocol of [FK97] uses information from both servers. Intuitively, instead of asking the server for a fixed prefix along the rounds (i.e., $c_1, c_2, \ldots c_{i-1}$ is the prefix for round i), for each $i \in [1..k]$ the referee asks on many random prefixes of length i. This allows the referee to send all those prefixes in a single round. However now, since the prefixes are not fixed, the referee cannot efficiently do the *consistency check* between $g_i(x)$ and $g_{i-1}(x)$ (i.e., checking that $\sum_{x \in H} g_i(x) = g_{i-1}(c_{i-1})$). So, the referee uses the second server for that. The consistency check is done by asking both servers for the polynomials g_i-s for random prefixes, such that for each length i there is one prefix that both servers receive from the referee. If both servers answer the same for that specific prefix, then by the assumption that one of the servers is honest, this answer is correct.

The full protocol is presented in Appendix C.

3.1.2 The Protocol of [GKR08]

In [GKR08] (and in more detail in [Rot09]) a protocol for delegation of computation is presented for any language in $\mathcal{L}$-uniform $\mathcal{NC}$, i.e., for any language that can be computed by circuits of poly-size and poly-logarithmic depth where there is a log-space Turing Machine that generates those circuits. The protocol is between a server and a client, where both know the input x and the said TM. We denote by k the length of x. The protocol is constructed in three steps:

1. They construct a protocol (which they call the *bare-bones protocol*) for delegation of computation of circuits of size $s(k)$ and depth $d(k)$, where the server running time is $poly(s(k))$, the client running time is $k \cdot poly(d(k), log(s(k)))$ and the number of rounds is $poly(d(k), log(s(k)))$. This protocol assumes there is an oracle that answers whether a given set of wires is connected through an addition or a multiplication gate in the circuit. (Actually, the oracle answers according to the low degree extensions of those predicates.)

2. They show that for any language L in $\mathcal{NL}$ (i.e., L has a non-deterministic log-space Turing Machine), it is possible to compute the oracle answers in time $poly(log(s(k)))$.

3. They show how to use the previous two steps to get delegation of computation for any language in $\mathcal{L}$-uniform $\mathcal{NC}$. The idea is to use the server as the oracle in the protocol for step 1, and, since for

$\mathcal{L}$-uniform $\mathcal{NC}$ there is a log-space circuit generator, we can use the protocol of step 1 to verify each one the server's answers regarding the circuit specification.

We note that it is important that the verification of the $\mathcal{NC}$ circuit, given the claimed existence of gates, be done before the verification of those claimed gates. Otherwise, the information that the server learns from the verification of the circuit specification answers can be used to cheat in the verification of the $\mathcal{NC}$ circuit.

The protocol can be extended to work with *non-uniform* $\mathcal{NC}$ languages by adding a *pre-processing* stage where the client runs in polynomial time (in the size of the circuit), independently of the input x. We note that the same technique is also applicable to our protocol. I.e., our protocol can work for non-uniform $\mathcal{NC}$ languages by adding the same pre-processing stage.

We describe the construction of [GKR08] in more detail. Given an arithmetic circuit $C : \{0,1\}^k \rightarrow \{0,1\}$ of fan-in 2 gates, size S, depth d (we assume that $d = \omega(log(S))$) and input length k, the server chooses the following parameters: 1) An extension field H of $GF[2]$ such that $max(d, log(S)) \leq |H| \leq poly(d, log(S))$, 2) An integer m such that $S \leq |H|^m \leq poly(S)$, 3) An extension field F of H such that $|F| \leq poly(|H|)$. (The size of F will influence the soundness of the protocol.) 4) An integer δ such that $|H| - 1 \leq \delta \leq |F|$. This integer depends on the circuit structure, and for $\mathcal{L}$-uniform $\mathcal{NC}$ circuits we set δ to be exactly $|H| - 1$.

Using standard techniques, we can transform the arithmetic circuit C to a new arithmetic circuit $C' : F^k \rightarrow F^S$ over the field F with the following properties: 1) C' is of size $poly(S)$ and depth d, with fan-in 2 gates, 2) Each layer, except for the input layer, is of size S (simply by adding dummy gates), 3) For every $(x_1, \ldots, x_k) \in \{0,1\}^k$, $C'(x_1, \ldots, x_k) = (C(x_1, \ldots, x_k), 0, \ldots, 0)$.

Let Υ be the predicate describing the circuit. That is, $\Upsilon(i, b, w_1, w_2, w_3)$ returns 1 if in the i-th layer there is a gate that connects wires w_2 and w_3 to wire w_1, and this gate is a b-gate where $b \in \{add, mult\}$. Let $\widetilde{\Upsilon}$ be the low degree extension of Υ with respect to H, F and m, of degree δ in each variable. The circuit specification oracle will answer according to $\widetilde{\Upsilon}$.

We denote the output layer as the 0 layer and the other layers according to their distance from the output layer. The input layer is the d-th layer. For $0 \leq i \leq d$ we associate a vector $v_i = (v_{i,0}, \ldots, v_{i,S-1}) \in F^S$ with the values of all gates of the i-th layer in the computation of $C'(x_1, \ldots, x_k)$. v_0 is the circuit result $(C(x_1, \ldots, x_k), 0, \ldots, 0)$ and v_d is the circuit input $(x_1, \ldots, x_k)$. Let $\widetilde{V}_i : F^m \rightarrow F$ be the low degree extension of the vector v_i with respect to H, F and m. This polynomial is of degree $\leq |H| - 1$ in each of its variables, and given v_i can be computed in time $\leq poly(|F|^m) = poly(S)$. Since v_d is of length k, $\widetilde{V}_d$ can be computed in time $\leq k \cdot poly(|H|, m)$.

Now, the protocol is as follows. The server claims that $C'(x_1, \ldots, x_k) = (0, \ldots, 0)$. An interactive protocol is executed between the server and the client. In each step the server reduces the correctness of the computation of layer i to the correctness of the computation of layer $i+1$. Concretely, for layer i, the server claims that $\widetilde{V}_i(u_i) = r_i$ for some randomly chosen u_i that the client picked and sent to the server. Then, the server reduces the correctness of this claim to the correctness of $\widetilde{V}_{i+1}(u_{i+1}) = r_{i+1}$ for some randomly chosen u_{i+1} that the client picked. This process continues until they reach the input layer and then the client verifies the correctness of this layer by himself (as $\widetilde{V}_d$ is small and known to the client).

We now describe in detail the reduction between the layers. Let $f_u^{(i)} : (F^m)^3 \rightarrow F$ be the function defined by

$$f_u^{(i)}(p, w, w') = \widetilde{\beta}(u, p) \cdot [\ \widetilde{\Upsilon}(i+1, 0, p, w, w')(\widetilde{V}_{i+1}(w) + \widetilde{V}_{i+1}(w'))\ +\ \widetilde{\Upsilon}(i+1, 1, p, w, w')(\widetilde{V}_{i+1}(w) \cdot \widetilde{V}_{i+1}(w'))\ ]$$

where $\widetilde{\beta}(u, p)$ is a $|H| - 1$ degree polynomial that depends only on F, H and m, and, can be computed in time $poly(|H|, m)$ (see [Rot09]). $f_u^{(i)}$ is a $3m$-variate polynomial of size $\leq poly(S)$ and degree $\leq 2\delta$. Given

an oracle access to $\widetilde{\Upsilon}$ and the values of $\widetilde{V}_{i+1}(w)$ and $\widetilde{V}_{i+1}(w')$, the function $f_u^{(i)}(p,w,w')$ can be evaluated in time $poly(|H|,m)$.

Given a claim for layer i that $\widetilde{V}_i(u_i) = r_i$ for some randomly chosen u_i that the client picked and sent to the server, it can be shown that $\widetilde{V}_i(u_i) = \sum_{p,w,w' \in H^m} f_{u_i}^{(i)}(p,w,w')$. Thus, proving that $\widetilde{V}_i(u_i) = r_i$ is equivalent to proving that $r_i = \sum_{p,w,w' \in H^m} f_{u_i}^{(i)}(p,w,w')$.

This part is done by a standard sum-check interactive protocol between the two servers. For each layer of the circuit, the client and the server execute a sum-check interactive protocol that consists of $3m$ rounds. The last step of the sum-check requires a computation of $f_{u_i}^{(i)}(p,w,w')$ by the client. In order to do that, the server sends a low degree polynomial $\widetilde{V}_{i+1}(\gamma(t))$ where $\gamma(t)$ is the 1-degree curve that passes through w and w'. Using this polynomial, the client computes $\widetilde{V}_{i+1}(w)$ and $\widetilde{V}_{i+1}(w')$ and uses that to compute $f_{u_i}^{(i)}(p,w,w')$. Then, the client picks a random point t' on the curve $\gamma(t)$ and continues to the correctness proof of the claimed value of $\widetilde{V}_{i+1}(\gamma(t'))$.

Complexity. The overall running time of the server is $poly(|F|^m) = poly(S)$, the running time of the client is $poly(|F|,m) + k \cdot poly(|H|,m) = k \cdot poly(d, log(S))$ and the communication complexity is $poly(|F|,m) = poly(d, log(S))$.

3.2 Our Protocol Given a Circuit Specification Oracle

The intuition behind our protocol is as follows. We assume the referee has an oracle access to $\widetilde{\Upsilon}$. We use the idea of [GKR08] to check the entire computation by checking the sum-checks between each two consecutive layers. We use the protocol of [FK97] to run each sum-check in a single round of communication. Ideally, we would like to execute all the sum-checks in parallel, in a single round. But, we cannot do that directly since the security of the [GKR08] protocol requires that the referee sends the random u_i to the server only *after* the execution of the $(i-1)$-th sum-check. Still, in order to use the protocol of [FK97], the servers have to know u_i since this value determines the function for the sum-check test. Thus, we change the "linking" between the layers.

For simplicity, we now describe the protocol as a sequential protocol with several rounds. However, since we want a one-round protocol, all servers actually execute all rounds of the this protocol *together*, in a single round. The referee chooses his messages for all rounds together and sends them to the servers in one message. Then, the servers answer all rounds together. Last, the referee reads all answers, starting from the input layer towards the output layer, and checks the servers' answers until he finds who is the honest server. (In our protocol the direction of the "linking" reductions is different than in [GKR08].) We denote this protocol by (P_1, P_2, R).

Given an input x, for each layer i the referee chooses two random parametric curves $\gamma_i(t), \varphi_i(t)$ that intersect at point z_i ($\gamma_i(z_i)$ corresponds to the point u_i of [GKR08]). The referee sends $\gamma_i(t), \varphi_i(t)$ to P_1 and P_2, respectively, and asks the servers for the polynomials $\widetilde{V}_i(\gamma_i(t))$ and $\widetilde{V}_i(\varphi_i(t))$. Next, he checks whether those polynomials agree on z_i. If they agree, then he assumes both answers are correct and continues to checking the next layer, $i-1$. Otherwise, he executes a one-round sum-check protocol *a la.* [FK97] to determine the correct value of $\widetilde{V}_i(\gamma_i(z_i))$. (Actually, we use a variation of the protocol of [FK97] which we describe in Appendix C.) But, as we mentioned before, we do not want to explicitly give the servers the value of z_i. Instead, the servers answer the sum-check challenges for *all* the points on $\gamma_i(t)$ and $\varphi_i(t)$, including the value at z_i. Then, the referee uses the referee of the one-round sum-check protocol of [FK97] to determine who is the honest server.

A subtle issue here is how the referee checks the correctness of the sum-checks without being able to compute $f_{z_i}^{(i)}$ by himself. The protocol of [FK97] assumes the referee can compute $f_{z_i}^{(i)}$ by himself for any point, but here, $f_{z_i}^{(i)}$ itself is too complex for the referee to compute. Specifically, in the protocol of [FK97] the referee needs to compute $f_{z_i}^{(i)}$ on three points: two points that are known only to P_1 and one point that is known only to P_2. In order to solve this problem we again use the *point-on-a-line* technique to

get those values "implicitly" from the servers themselves. When the referee believes that the answers on $\widetilde{V}_{i+1}(\gamma_{i+1}(t))$ and $\widetilde{V}_{i+1}(\varphi_{i+1}(t))$ for layer $i+1$ are correct, he takes few random points on those polynomials and uses that to compute the values of $f_{z_i}^{(i)}$ on those three points. The solution requires increasing by one the degrees of the polynomials of the protocol of [FK97] in order to keep the added points secret (see Appendix C).

The detailed protocol (P_1, P_2, R) is presented in Figures 1 and 2. Since some of the polynomials conceal secret intersection points, when the referee sends some polynomial to the servers, we require that he sends the canonical representation of that polynomial.

The referee's running time is $poly(|F|, m, d, |H|) + k \cdot poly(|H|, m) = k \cdot poly(d, log(S))$, the servers running time is $poly(S, |F|, m, d) = poly(S)$ and the communication complexity is $poly(|F|, m, d) = poly(d, log(S))$.

Theorem 3. *Let L be a language in $\mathcal{NC}$ and let C_L be the circuit that decides on L. For any input x and for any constant error probability ε, given a circuit specification oracle for C_L, the protocol $(P_1(x), P_2(x), R(x))$ is ε-RDoC with two servers for the circuit C_L.*

The proof is given in Appendix D.

3.3 Removing the Circuit Specification Oracle

For any language $L \in \mathcal{L}$-uniform $\mathcal{NC}$ there exists a circuit C_L of poly-size and polylogarithmic-depth that computes L. Furthermore, the polynomials $\widetilde{\Upsilon}$ of C_L can be computed by a log-space TM, which means that $\widetilde{\Upsilon}$ can be computed by an $\mathcal{NL}$ circuit, $C_{spec(L)}$. As shown in [GKR08], the circuit specification function $\widetilde{\Upsilon}$ of circuits in $\mathcal{NL}$ can be computed in poly-logarithmic time. This means that the referee can compute $\widetilde{\Upsilon}$ of $C_{spec(L)}$ by himself, and execute the protocol from Section 3.2 without an oracle assistance.

Recall the idea of [GKR08] for extending the *bare-bones* protocol to $\mathcal{L}$-uniform $\mathcal{NC}$ circuits. In order to verify the computation of the circuit C_L, the client runs the bare-bones protocol for verifying C_L, and asks the server for the required values of the circuit specification function $\widetilde{\Upsilon}$ (i.e. the server acts as the oracle). Then, the client checks each of those claimed values by executing the bare-bones protocol for the circuit C_{specL} (for which he can compute the oracle answers by himself).

Now, if we try to follow this idea for extending the protocol from Section 3.2 to work with $\mathcal{L}$-uniform $\mathcal{NC}$ circuits, and try to run in parallel the protocol also for verification of $C_{spec(L)}$, we get contradicting requirements. On the one hand, for verification of $C_{spec(L)}$, both servers have to know p_j, w_j, w'_j for $j = 0, 1, 2$ as those are the *inputs* for the specification circuit (and the protocol assumes those inputs are known to both servers), but on the other hand, for verification of C_L, the soundness of the protocol requires that those values will not be known to both servers.

In order to tackle this problem, we use a similar idea to the one used in the previous protocol. The referee asks the servers to answer on many points, without revealing the actual p_j, w_j, w'_j. Note that each one of p_j, w_j, w'_j is at least "implicitly" known to both servers. E.g., p_0, w_0, w'_0 is implicitly known to P_2 from $D_{3k}(t)$ (and is explicitly known to P_1 by A_{3m}). Also, we can explicitly send the values of p_0 and p_2 to P_1, and the values of p_1 to P_2, without ruining the soundness of the previous protocol.

Using those two observations, we construct a protocol (P'_1, P'_2, R') for any language in $\mathcal{L}$-uniform $\mathcal{NC}$. For verification of the output of C_L, the referee executes the protocol from Section 3.2 with two modifications: 1) The referee sends to P'_1 also the values of p_0, p_2 and to P'_2 also the values of p_1 for all layers, and, 2) P'_1 sends the (claimed) values of $\widetilde{\Upsilon}(i, b, p_0, w_0, w'_0)$ and $\widetilde{\Upsilon}(i, b, p_2, w_2, w'_2)$ for all layers, and P'_2 sends the (claimed) values of $\widetilde{\Upsilon}(i, b, p_1, w_1, w'_1)$ for all layers.

For each of the answers $\widetilde{\Upsilon}(i, b, p_j, w_j, w'_j)$, the referee executes the protocol from Section 3.2 for verification of those claimed values using the circuit $C_{spec(L)}$ (for which he can compute the circuit specification by himself). As we mentioned before, neither of p_j, w_j, w'_j (for $j = 0, 1, 2$) is explicitly known to both

Publicly known parameters

H, F, m, d, S, k, δ as in Section 3.1.2.

Initialization

For $i = 1, \ldots d$, R randomly picks $z_i \in F$, a random degree-4 parametric curve $\gamma_i(t) \in F[t]^m$, and a random degree-2 parametric curve $\varphi_i(t) \in F[t]^m$ going through $(z_i, \gamma_i(z_i))$.
He also sets $\gamma_0 = \varphi_0 \equiv 0$ and $z_0 = 0$, and computes $M_d(t) = \widetilde{V}_d(\gamma_d(t))$ and $Q_d(t) = \widetilde{V}_d(\varphi_d(t))$.

For $i = d, \ldots, 1$

R's computations :

R sets $w_0 = \gamma_i(0), w_0' = \gamma_i(1), w_1 = \varphi_i(0), w_1' = \varphi_i(1), w_2 = \gamma_i(2), w_2' = \gamma_i(3)$ and randomly chooses $p_0, p_1, p_2 \in F$.

For $1 \le j \le 3m$, R chooses random vectors $A_j, B_j \in F^j$ and random elements $a_j, b_j \in F$. R sets A_{3m} to $p_0 \circ w_0, \circ w_0'$ and chooses a random $r \in F$.

For $1 \le j \le 3m-1$ let $C_j(t) \in F[t]^j$ be the unique degree-$|H|$ parametric curve going through

$$(0, A_{j-1} \circ 0), \ldots, (|H|-1, A_{j-1} \circ (|H|-1)), (|H|, B_j)$$

and let $C_{3m}(t) \in F[t]^{3m}$ be the unique degree-$(|H|+1)$ parametric curve going through

$$(0, A_{3m-1} \circ 0), \ldots, (|H|-1, A_{3m-1} \circ (|H|-1)), (|H|, B_{3m}), (r, p_1 \circ w_1 \circ w_1').$$

For $1 \le j \le 3m-1$, let $D_j(t) \in F[t]^j$ be the unique degree-1 parametric curve going through

$$(a_j, C_j(a_j)), (b_j, A_j)$$

and let $D_{3m}(t) \in F[t]^{3m}$ be the unique degree-2 parametric curve going through

$$(r, p_2 \circ w_2 \circ w_2'), (a_{3m}, C_{3m}(a_{3m})), (b_{3m}, A_{3m}).$$

We define

$$\Phi_{q,j}(x_1, \ldots, x_j) = \sum_{x_{j+1}, \ldots, x_{3m} \in H} f_q^{(i-1)}(x_1, \ldots, x_j, x_{j+1}, \ldots, x_k).$$

R sends to $\mathbf{P}_1$:

$C_j(t), A_j$, for $1 \le j \le 3m$, and the curve $\gamma_{i-1}(t)$.

$\mathbf{P}_1$ sends to R :

For all $q \in F$ define $N_{j,q} = \Phi_{\gamma_{i-1}(q),j}(A_j)$ and $F_{j,q}(t) = \Phi_{\gamma_{i-1}(q),j}(C_j(t))$.

P_1 sends $N_{j,q}, F_{j,q}(t)$ for $1 \le j \le 3m$ and all $q \in F$, and, $M_{i-1}(t)$ where $M_{i-1}(t) = \widetilde{V}_{i-1}(\gamma_{i-1}(t))$.

R sends to $\mathbf{P}_2$:

$D_j(t)$ for $1 \le j \le 3m$, and the curve $\varphi_{i-1}(t)$.

$\mathbf{P}_2$ sends to R :

For all $q \in F$ define $G_{j,q}(t) = \Phi_{\varphi_{i-1}(q),j}(t))$.

P_2 sends $G_{j,q}(t)$ for $1 \le j \le 3m$ and all $q \in F$, and, $Q_{i-1}(t)$ where $Q_{i-1}(t) = \widetilde{V}_{i-1}(\varphi_{i-1}(t))$.

Figure 1: Protocol I: initialization and interactive phase

servers. So instead we ask one of the servers to answer on many points instead of the specific p_j, w_j, w_j'. Specifically, for verification of $\widetilde{\Upsilon}(i, b, p_0, w_0, w_0')$ of layer i of C_L, we execute the protocol from Section

Checking layer i for $i = d, \ldots, 1$

P_1 is declared as the cheater if $M_{i-1}(t)$ has degree bigger than $4 \cdot m \cdot (|H| - 1)$. P_2 is declared as the cheater if $Q_{i-1}(t)$ has degree bigger than $2 \cdot m \cdot (|H| - 1)$.

If $M_{i-1}(z_{i-1}) = Q_{i-1}(z_{i-1})$ the referee continues to the proof of layer $i - 1$. Otherwise, he continues as follows.

Given $M_i(t)$, R computes:

- $f_{z_{i-1}}^{(i-1)}(A_{i,3m})$ and $f_{z_{i-1}}^{(i-1)}(p_2 \circ w_2 \circ w_2')$ using $M_i(0), M_i(1)$, and $M_i(2), M_i(3)$ (and the oracle).
- $f_{z_{i-1}}^{(i-1)}(p_1 \circ w_1 \circ w_1')$ using $Q_i(0), Q_i(1)$ (and the oracle).

Now, R verifies the sum-check of $M_{i-1}(z_{i-1}) = \sum_{p,w,w' \in H^m} f_{z_{i-1}}^{(i-1)}(p, w, w')$ using the referee from [FK97].

Concretely:

- The referee sets $N_{0,z_{i-1}} = M_{i-1}(z_{i-1})$.
- P_1 is declared as the cheater if: 1) $N_{3m,z_{i-1}} \neq f_{z_{i-1}}^{(i-1)}(A_{3m})$, or 2) $F_{3m,z_{i-1}}(r) \neq f_{z_{i-1}}^{(i-1)}(p_1 \circ w_1 \circ w_1')$, or 3) For some j, $F_{j,z_{i-1}}(t)$ has degree greater than $deg(C_j) \cdot j \cdot 2\delta$ or $\sum_{m=0}^{|H|-1} F_{j,z_{i-1}}(m) \neq N_{j-1,z_{i-1}}$.
- P_2 is declared as the cheater if for some j, $G_{j,z_{i-1}}(t)$ has degree greater than $deg(D_j) \cdot j \cdot 2\delta$, or, $G_{3m,z_{i-1}}(r) \neq f_{z_{i-1}}^{(i-1)}(p_2 \circ w_2 \circ w_2')$.
- Let's denote by j the smallest number such that $G_{j,z_{i-1}}(b_j) = N_{j,z_{i-1}}$. If $G_{j,z_{i-1}}(a_j) = F_{j,z_{i-1}}(a_j)$ then P_1 is declared as the honest, otherwise P_2 is declared as the honest.

Outputting the result

If P_1 was declared as the honest server or P_2 was declared as the cheater, R outputs $M_0(0)$, otherwise he outputs $Q_0(0)$. (Recall that $M_0(0)$ is the claimed result of P_1 and $Q_0(0)$ is the claimed result of P_2.)

Figure 2: Protocol I: verification of answers

3.2, where P_1' plays the role of P_1 and knows p_0, w_0, w_0', and P_2' plays the role of P_2 for *all the points* on the curve $D_{3m}(t)$ as the possible inputs for $C_{spec(L)}$. (There are at most $|F|$ points on this curve). This means that P_2' does not know the specific p_0, w_0, w_0'. However, since $D_{3m}(t)$ passes through p_0, w_0, w_0', one of those answers will be the needed P_2''s answer for the input p_0, w_0, w_0'. Similarly, the same roles go for p_2, w_2, w_2' (which is also included in $D_{3m}(t)$). For p_1, w_1, w_1' for all layers of C_L, P_2' plays the role of P_1 in the protocol of Section 3.2 and P_1' plays the role of P_2 for *all the points* on the curve $C_{3m}(t)$ (which includes p_1, w_1, w_1').

When the referee receives the messages from both servers (for verification of $C_{spec(L)}$ and of C_L), he checks if they agree on all the values of $\widetilde{\Upsilon}(i, b, p_j, w_j, w_j')$. If they disagree on some of the values, then the referee checks one of those disagreements using the referee R from Section 3.2 and outputs according to its answer. If the servers agree on all the values of $\widetilde{\Upsilon}$, then by the assumption that one of them is honest, those values are correct. Then, the referee verifies the computation of the circuit C_L given the values of $\widetilde{\Upsilon}$ he got before. He runs the checking phase of the referee from Section 3.2 and outputs according to its answer.

The overhead of this solution is only polynomial in all parameters. For each layer we have three invocations of the protocol form Section 3.2 where one of the servers executes the protocol for $|F|$ points. Summing over all layers, the total overhead is $\leq 3 \cdot d \cdot |F| \cdot depth(C_{spec(L)})$ which is still poly-logarithmic in the size of the input.

Theorem 4. *Let L be a language in $\mathcal{L}$-uniform $\mathcal{NC}$. For any input x and for any constant error probability*

ε, the protocol $(P'_1(x), P'_2(x), R'(x))$ is ε-RDoC with two servers for the circuit C_L.

The proof is given in Appendix E.

4 Computationally Sound RDoC for Any Polynomial Time Computation

We base our protocol on the work of Feige and Kilian [FK97] where they present a refereed game with polynomial number of rounds and private communication channels (therefore not full information) for any *EXPTIME* language. Their protocol uses *arithmetization* for consistency checks and then takes advantage of the *locality* property of a single Turing Machine step (each Turing Machine transition uses only $O(1)$ *local* information: the current state, the current head position and the current character). In their protocol for languages in *EXPTIME*, the referee is polynomial in the length of the input x.

Their construction can be directly scaled-down for languages in P, yielding a protocol where the servers are polynomial in the input size and the referee is quasi-linear. Correctness remains unconditional. However, the protocol requires private communication channels between the referee and the two servers, and it has a relatively high (but constant) error probability.

We modify their scaled-down protocol (for $\mathcal{P}$ languages) by replacing the use of arithmetization with Merkle Hash Trees (see Appendix F for a brief description of Merkle Hash Trees). Although it gives only computational soundness, it greatly improves the efficiency of the protocol. Furthermore, using Merkle Hash Tree gives us a negligible error probability for even one execution of the protocol. Last, our protocol is full-information and in particular does not require private communication channels. In a setting where messages between the players are digitally signed, the referee can obtain a publicly verifiable proof that a server is cheating.

4.1 Reduced Turing Machine Configuration

We denote by $MH_{root}(st)$ the value of the root of the Merkle Hash Tree for the string st, and by $MH_{proof}(st, i)$ the proof of consistency for character i in the Merkle Hash Tree for the string st. (This proof also includes $MH_{root}(st)$.) Also, we denote by *VerifyMHProof*(i, st_i, prf) the function that given a claimed $prf = MH_{proof}(st, i)$, outputs True if prf is a valid proof of consistency for character st_i, and False otherwise.

We use the following property of Merkle Hash Trees. If one has the proof of consistency for the i-th character of st, $MH_{proof}(st, i)$, he can choose any character c and efficiently compute $MH_{root}(st')$ where st' equals to st except that the i-th character is c. He does that by computing $h_i = H(c)$ and by using $MH_{proof}(st, i)$ to compute the new hash values along the path to h_i. This takes logarithmic time in the length of st, and does not require knowing the full string st.

Given a deterministic TM with one working, we define a configuration as a tuple $(state, head, tape)$ representing the current state, the current head position and the current working tape, respectively. For general polynomial computations, the size of a single configuration can be polynomial in the length of the input. For simplicity of the description, we denote by k the maximal length of the tape during the computation. In our protocol, the servers will tell the referee their maximal length of their tapes, so the referee could take the bigger value to be this k.

We define a *reduced-configuration* of $(state, head, tape)$ to be the tuple-

$$(\ state, head, tape[head], MH_{proof}(tape, head)\ ).$$

Note that the size of the reduced-configuration is logarithmic in k, the size of the working tape, and therefore is at most logarithmic in the size of the computation.

We observe that given two reduced TM configurations $rc_1 = (s_1, h_1, v_1, p_1),\ rc_2 = (s_2, h_2, v_2, p_2)$ that are claimed to be consecutive, one can efficiently verify this claim by checking the following: 1) Verify that *VerifyMHProof*(h_1, v_1, p_1) returns True and that *VerifyMHProof*(h_2, v_2, p_2) returns True. 2) Simulate the

TM on s_1, h_1, v_1 and get the next state s', new head position h' and a new value v' for the h_1-th character. 3) Verify that s', h' equal to s_2, h_2. 4) Using p_1 and v', compute $r' = MH_{root}(t'_1)$ where t'_1 is the tape of configuration rc_1 except for the value v' in its h_1-th character (note that the full tape is not part of the reduced configuration). 5) Verify that the hash of the root in p_2 equals to r'. 6) If one of the previous checks fail, then the claim is false. Otherwise it is true.

We denote by *VerifyReducedStep*(rc_1, rc_2) the function that given two reduced configurations rc_1, rc_2 outputs False if any of the above checks fails, and True otherwise.

4.2 The Protocol

We describe Protocol II in the Turing Machine model. Given a Collision-Resistant Hash Function, our protocol is the following. The servers and the referee have a TM that computes f. The referee sends x to both servers and asks for $f(x)$. In case they answer the same, by the assumption that one of them is honest, the answer is the correct one. Else, the referee continues to a *binary-search* phase. The referee asks the servers to send him the number of steps it takes to compute TM(x), takes the smaller answer as the current *bad row* variable, n_b, and sets to 1 the current *good row* variable, n_g. The referee also asks for the maximal length of their stored configurations and takes the bigger answer to be k. Now, the referee asks for the reduced configuration of the $(n_b - n_g)/2 + n_g$ configuration. If one of the answers is not a valid reduced configuration, the referee outputs the value of $f(x)$ of the other server (this is the honest server). If answers match, he sets $n_g = (n_b - n_g)/2 + n_g$, otherwise, he sets $n_b = (n_b - n_g)/2 + n_g$. The referee continues the binary search in that way till he gets $n_g + 1 = n_b$. Note that the servers do not have to remember *all* the configurations, instead, they can remember only two configurations, one for the last n_g and one for the last $(n_b - n_g)/2 + n_g$. Then, when asked for the next configuration, the server can continue the TM execution from one of those configurations. Overall, in worst case scenario, the servers execution time is not much more than executing the program twice.

Now, the referee takes the reduced configuration n_g and the reduced configuration that Server 1 sent for row n_b and checks whether those two reduced configurations are consecutive. If they are, he outputs the value of $f(x)$ of Server 1. Otherwise, he outputs the value of $f(x)$ of Server 2.

Overall we have:

Theorem 5. *Assume the hash function in use is collision resistant. Then the above protocol is a computationally sound, full-information, RDoC with two servers and with negligible soundness ε for any poly-size function. For functions that can be computed by TMs taking $T(n)$ steps and $S(n)$ space on input x with $|x| = n$, the protocol takes $\log T(n)$ rounds, the referee runs in time $O(n+\kappa \log T(n) \log S(n)+\kappa \log S(n))$ and the servers run in time $O(T(n) + \kappa S(n) \log T(n))$, where κ is a security parameter.*

Proof (sketch). By the specification of the protocol, a malicious Server 1 can convince the referee only if he can generate a reduced configuration n_b that is not correct, but has the same root of the Merkle Hash Tree as of the correct reduced configuration. However, this means that the referee can find a collision of the hash function (in some node along the path to index $head$), and by the security of the collision resistant hash function, this can only happen with a negligible probability. □

See Appendix G for several extensions of this protocol.

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz, *From secrecy to soundness: efficient verification via secure computation*, ICALP '10: Proceedings of the 37th international colloquium conference on Automata, languages and programming, Springer-Verlag, 2010, pp. 152–163.

[AS98] Sanjeev Arora and Shmuel Safra, *Probabilistic checking of proofs: a new characterization of NP*, J. ACM **45** (1998), 70–122.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy, *Checking computations in polylogarithmic time*, STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing, ACM, 1991, pp. 21–32.

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor, *Does parallel repetition lower the error in computationally sound protocols?*, FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1997, pp. 374–383.

[BOGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson, *Multi-prover interactive proofs: how to remove intractability assumptions*, STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM, 1988, pp. 113–131.

[CHS05] Ran Canetti, Shai Halevi, and Michael Steiner, *Hardness amplification of weakly verifiable puzzles*, TCC '05: Proceedings of the second Theory of Cryptography Conference, Springer-Verlag, 2005, pp. 17–33.

[CKV10] Kai Min Chung, Yael Kalai, and Salil Vadhan, *Improved delegation of computation using fully homomorphic encryption*, CRYPTO '10: Proceedings of the 30th annual conference on Advances in cryptology, Springer-Verlag, 2010, pp. 483–501.

[FK97] Uriel Feige and Joe Kilian, *Making games short (extended abstract)*, STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, ACM, 1997, pp. 506–516.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno, *Non-interactive verifiable computing: outsourcing computation to untrusted workers*, CRYPTO '10: Proceedings of the 30th annual conference on Advances in cryptology, Springer-Verlag, 2010, pp. 465–482.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum, *Delegating computation: interactive proofs for muggles*, STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing, ACM, 2008, pp. 113–122.

[Kil92] Joe Kilian, *A note on efficient zero-knowledge proofs and arguments (extended abstract)*, STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, ACM, 1992, pp. 723–732.

[KR09] Yael Tauman Kalai and Ran Raz, *Probabilistically checkable arguments*, CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, 2009, pp. 143–159.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan, *Algebraic methods for interactive proof systems*, J. ACM **39** (1992), 859–868.

[Mer88] Ralph C. Merkle, *A digital signature based on a conventional encryption function*, CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology, Springer-Verlag, 1988, pp. 369–378.

[Mic00] Silvio Micali, *Computationally sound proofs*, SIAM J. Comput. **30** (2000), 1253–1298.

[Rot09] Guy N. Rothblum, *Delegating computation reliably: paradigms and constructions*, Ph.D. Thesis, Massachusetts Institute of Technology, 2009.

[RV10] Guy N. Rothblum and Salil Vadhan, *Are PCPs inherent in efficient arguments?*, Comput. Complex. **19** (2010), no. 2, 265–304.

A Prior Work

Prior work has studied the question of proving the correctness of general computations. (Most previous works focused on interactive proofs between a verifier and a prover. However, given an interactive protocol for proving the correctness of a computation of f, one can easily get verifiable delegation of computation by asking the server for $y = f(x)$ and a proof that y is the correct result.) Babai *et al.* [BFLS91] consider this question in a setting where the prover is a non-adaptive oracle. Kilian [Kil92] and Micali [Mic00] build

on their techniques and show efficient computationally sound protocols, whose security is based on cryptographic assumptions and where soundness holds only against computationally bounded cheating provers. Micali gets a non-interactive computationally sound proof based on the existence of a Random Oracle whereas Kilian gets a two-round interactive computationally sound proof assuming the existence of collision resistance hash family. Goldwasser *et. al.* [GKR08] present an information theoretically sound interactive proof protocol for verifiable computation for any language in $\mathcal{L}$-uniform $\mathcal{NC}$. The number of rounds between the prover and the verifier is poly-logarithmic in the size of the computation. Using the technique of Kalai and Raz [KR09] this protocol can be transformed into a one-round protocol, assuming the existence of a computational Private Information Retrieval scheme with poly-log communication.

We note that, with the exception of [GKR08], the above works are based on Probabilistic Checkable Proofs (PCP) [AS98]. Although constructions of PCP are very efficient by means of asymptotic complexity, they are far from being practical. Furthermore, Rothblum and Vadhan [RV10] show an evidence that using PCP is an inherent requirement even for computational sound proof of computation with constant number of rounds. Therefore, when considering practical constant round protocols with one prover, it seems that a major efficiency improvement of PCP is required.

Gennaro *et al.* [GGP10], Chung *et. al.* [CKV10] and Applebaum *et al.* [AIK10] consider a model with a pre-processing stage. Based on the existence of a fully homomorphic encryption, they construct computationally sound protocols, where in an offline pre-processing stage the verifier runs in time proportional to the size of the computation. Afterwards, in an online stage, the verifier (using the result of the pre-processing stage) runs in time proportional to the size of its inputs and the computation results. In these works, as long as the verifier does not encounter cheating provers, the same pre-processing information can be used in multiple rounds, yielding improved amortized complexity.

A related proof model with several provers is the model of Multi-Prover Interactive Proofs, suggested by Ben-Or *et al.* [BOGKW88]. In this model, even if *all* of the provers cheat, the verifier will detect that they are cheating. However, soundness is guaranteed assuming that malicious provers *cannot* communicate or coordinate their strategies during the protocol. This is in contrast to the refereed games of Feige and Kilian [FK97] and to our model, where soundness is guaranteed as long as one server is honest, even if some group of malicious servers communicate *during* the protocol. In addition, the referee learns who are the cheating provers.

B Low Degree Extension (LDE)

Given a field F, a subset $H \subseteq F$ and a function $f : H^m \to F$, we let the low degree extension of f, denoted $\widetilde{f} = LDE(f)$, be the *unique* multi-variate polynomial $\widetilde{f} : F^m \to F$ that satisfies:

- (low-degree) $deg(\widetilde{f}) < |H|$ for each variable.
- (extension) $f(x) = \widetilde{f}(x)$ for all $x \in H^m$.

Such polynomials can be constructed using Lagrange Interpolation.

Similarly we define the low degree extension of a vector. Let $\alpha : H^m \to \{0, \ldots |H|^m - 1\}$ be the lexicographic order of H^m. Given a vector $\vec{w} = (w_0, \ldots w_{k-1}) \in F^k$, where $k \leq |H|^m$, we can view this vector as a function $f_{\vec{w}} : H^m \to F$ such that $f_{\vec{w}}(z) = w_{\alpha(z)}$ when $\alpha(z) \leq k-1$ and $f_{\vec{w}}(z) = 0$ otherwise. We define the low degree extension of the vector $\vec{w}$ to be $LDE(f_{\vec{w}})$.

C The Protocol of [FK97]

Following the intuition behind the protocol in Section 3.1.1, we now describe the protocol in detail. Recall that we have a finite field F, a subset of F denoted by H, a fixed number k and a multivariate polynomial $f : F^k \to F$ of degree $\leq d$ in each variable. Server 1 claims that

$$\sum_{x_1, x_2, \ldots, x_k \in H} f(x_1, x_2, \ldots, x_k) = N_0$$

for some value N_0 and Server 2 claims otherwise (denote this value by N_0').

For simplicity, we use the shorthand $a \circ b$ for a vector that is a concatenation of a and b (where a, b are vectors or single elements). We assume the elements of H are $0, 1, \ldots, |H| - 1$. Instead of working with prefixes, all computations are done using low degree parametric curves, which is a more compact representation. (A parametric curve of degree d in $F[t]^j$ is a tuple of j one-parameter polynomials over the field F, each one of degree $\leq d$.)
The protocol is as presented in Figure 3.

R's computations: For $1 \leq j \leq k$, R chooses random vectors $A_j, B_j \in F^j$ and random elements $a_j, b_j \in F$. Let $C_j(t) \in F[t]^j$ be the unique degree-$|H|$ parametric curve going through

$$(0, A_{j-1} \circ 0), \ldots, (|H| - 1, A_{j-1} \circ (|H| - 1)), (|H|, B_j)$$

and let $D_j(t) \in F[t]^j$ be the canonical representation of the unique degree-1 parametric curve going though

$$(a_j, C_j(a_j)), (b_j, A_j).$$

For $j = 1 \ldots k$ we define the functions

$$\Phi_j(x_1, \ldots, x_j) = \sum_{x_{j+1}, \ldots, x_k \in H} f(x_1, \ldots, x_j, x_{j+1} \ldots, x_k).$$

Note that

$$\Phi_j(x_1, \ldots, x_j) = \sum_{x_{j+1} \in H} \Phi_{j+1}(x_1, \ldots, x_{j+1}).$$

R sends to P_1: $A_1, \ldots, A_k, B_1, \ldots, B_k$.

P_1 sends to R: $N_0, N_1, \ldots, N_k, F_1(t), \ldots, F_k(t)$, where $N_j = \Phi_j(A_j)$ and $F_j(t) = \Phi_j(C_j(t))$ for $j \geq 1$.

R sends to P_2: $D_1(t), \ldots, D_k(t)$.

P_2 sends to R: $N_0', G_1(t), \ldots, G_k(t)$, where $G_j(t) = \Phi_j(D_j(t))$.

R declares the winner: R chooses random $r \in F$. P_1 losses immediately if: 1) $N_k \neq f(A_k)$, or 2) $F_k(r) \neq f(C_k(r))$, or 3) For some j, $F_j(t)$ has degree greater than $|H| \cdot j \cdot d$ or $\sum_{i=0}^{|H|-1} F_j(i) \neq N_{j-1}$.

P_2 losses immediately if for some j, $G_j(t)$ has degree greater than $j \cdot d$, or, $G_k(r) \neq f(D_k(r))$.

Let's denote by j the smallest number such that $G_j(b_j) = N_i$. If $G_j(a_j) = F_j(a_j)$ then P_1 is declared as the winner, otherwise P_2.

Figure 3: One-round refereed game for the sum-check task

Theorem 6. *Let F be a finite field and H subset of F. Let $f : F^k \to F$ be a multivariate polynomial of degree $\leq d$ in each variable and let $N = \sum_{x_1, x_2, \ldots, x_k \in H} f(x_1, x_2, \ldots, x_k)$. The above protocol is a refereed game with the following properties:*

- *If P_1 claims that $N_0 = N$, then he will be declared as the winner with probability $\geq 1 - \frac{|H| \cdot 2k^2 \cdot d}{|F|}$.*
- *If P_1 claims that $N_0 \neq N$, then he will be declared as the winner with probability $\leq \frac{|H| \cdot 2k^2 \cdot d}{|F|}$*

The referee is polynomial in $|H|$ and k, the (honest) servers are polynomial in $|F|^k$ and the communication complexity is polynomial in $|F|$ and k.

Proof (sketch). Let S_1 be the event that

$$\sum_{x_1,x_2,\ldots,x_k \in H} f(x_1, x_2, \ldots, x_k) \neq N_0$$

but P_1 is declared as the winner (i.e., P_2 is the honest server). Let U_i be the event that $F_i(t)$ is indeed $\Phi_i(C_i(t))$, let E_i be the event that $F_i(a_i)$ is indeed $\Phi_i(C_i(a_i))$ and let E' be the event that $F_k(r)$ is indeed $f(C_k(r))$.

$$Pr[S_1] \leq Pr[E' \wedge \neg U_k] + Pr[\exists i \in [1..k] \; s.t. \; E_i \wedge \neg U_i] \leq Pr[E' \wedge \neg U_k] + \sum_{i=1}^{k} Pr[E_i \wedge \neg U_i].$$

By the fact that two distinct univariate degree-t polynomials agree on at most t points we get that

$$Pr[E' \wedge \neg U_k] \leq \frac{|H| \cdot k \cdot d}{|F|},$$

and that

$$Pr[E_i \wedge \neg U_i] \leq \frac{|H| \cdot i \cdot d}{|F|} \leq \frac{|H| \cdot k \cdot d}{|F|}.$$

Thus,

$$Pr[S_1] \leq \frac{|H| \cdot k \cdot d}{|F|} + k \cdot \frac{|H| \cdot k \cdot d}{|F|} \leq (k+1) \cdot \frac{|H| \cdot k \cdot d}{|F|}.$$

Let S_2 be the event that P_1 is the honest server but P_2 is declared to be the winner. Using a similar proof, we have that

$$Pr[S_2] \leq (k+1) \cdot \frac{k \cdot d}{|F|}.$$

The only difference is that in this case the degrees of $G_j(t)$ are smaller than the degrees of $F_j(t)$ by a factor of $|H|$.

Therefore, the soundness of the protocol is bounded by $\frac{|H| \cdot 2k^2 \cdot d}{|F|}$. □

We remark that our protocol from Section 3.2 has another difference compared to the above protocol. We increase by one the degrees of the curves C_k and D_k. Using a similar argument to the above it can be shown that the soundness of that protocol is bounded by $\frac{(|H|+1) \cdot 2k^2 \cdot d}{|F|}$.

D Proof of Theorem 3

The crucial point of the proof is that a server can cheat with high probability only if he knows the curves' intersection points. Let's see what information each server has about the other server's curves.

Lemma 7. *Let V_1 be the view of P_1 and let i be a round in the protocol. For all $\alpha, \alpha', \beta, \beta', \gamma, \gamma' \in F$ and $j \in [1 \ldots 3m]$*

$$Pr[z_i = \alpha | V_1] = Pr[z_i = \alpha' | V_1] \quad (1)$$

$$Pr[a_j = \beta | V_1] = Pr[a_j = \beta' | V_1] \quad (2)$$

$$Pr[r = \gamma | V_1] = Pr[r = \gamma' | V_1]. \quad (3)$$

Let V_2 be the view of P_2 and let i be a round in the protocol. For all $\alpha, \alpha', \beta, \beta', \gamma, \gamma', \zeta, \zeta' \in F$ and $j \in [1 \dots 3m]$

$$\begin{aligned} Pr[z_i = \alpha | V_2] &= Pr[z_i = \alpha' | V_2] && (4) \\ Pr[a_j = \beta | V_2] &= Pr[a_j = \beta' | V_2] && (5) \\ Pr[b_j = \gamma | V_2] &= Pr[b_j = \gamma' | V_2] && (6) \\ Pr[r = \zeta | V_2] &= Pr[r = \zeta' | V_2]. && (7) \end{aligned}$$

Proof. The lemma follows from inspecting the protocol.

1. $\varphi_i(t)$ is of degree-2. Even if we give P_1 the exact values of w_1, w_1' (and not only implicitly as part of C_{3m}), there is still one degree of secret information, and therefore $\varphi_i(t)$ can still go through any possible point $(z_i, \gamma_i(z_i))$ for all $z_i \in F$.

2. $D_j(t)$ is of degree at least 1. Even if we give P_1 the value of b_j, he does not have enough information to recover $D_j(t)$, so any $(a_j, C_j(a_j))$ is a possible intersection point.

3. Since P_1 has no information on w_1, w_1' besides from the curve C_{3m}, r is simply a random point on the curve from his point of view.

4. $\gamma_i(t)$ is of degree-4. Even if we give P_2 the exact values of w_0, w_0', w_2, w_2' (and not only implicitly as part of D_{3m}), there is still one degree of secret information, and therefore $\gamma_i(t)$ can still go through any possible point $(z_i, \varphi_i(z_i))$.

5. $C_j(t)$ is of degree at least $|H|$. Even if we give P_2 the value of b_j, he does not have enough information to recover $C_j(t)$, so any $(a_j, D_j(a_j))$ is a possible intersection point.

6. Similar argument as in (5) goes for b_j.

7. Since P_2 has no information on w_2, w_2' besides from the curve D_{3m}, r is simply a random point on the curve from his point of view.

□

Proof of Theorem 3. Using Lemma 7, let's see how much a malicious server can cheat without knowing the intersection points (z_i, a_j, b_j). For a fixed input x and a fixed circuit C, let S_1 be the event that although P_1 is the malicious server and the referee outputs a wrong result (i.e., $M_0(0)$ that is not equal to $C(x)$). Let T_i be the event that $M_i(t)$ is indeed $\widetilde{V}_i(\gamma_i(t))$, and let E_i be the event that $M_i(z_i)$ is indeed $\widetilde{V}_i(\gamma_i(z_i))$. Then,

$$Pr[S_1] \leq Pr[\neg T_0 \wedge T_{d-1}] \leq Pr[\exists i \in [d-1] \; s.t. \; \neg T_i \wedge T_{i+1}] \leq \sum_{i=0}^{d-1} Pr[\neg T_i \wedge T_{i+1}].$$

For every $i \in [d-1]$,

$$Pr[\neg T_i \wedge T_{i+1}] = Pr[\neg T_i \wedge T_{i+1} \wedge E_i] + Pr[\neg T_i \wedge T_{i+1} \wedge \neg E_i].$$

By the soundness property of the protocol from [FK97] (see Appendix C), we have that

$$Pr[\neg T_i \wedge T_{i+1} \wedge \neg E_i] \leq Pr[T_{i+1} \wedge \neg E_i] \leq \frac{(|H|+1) \cdot 2(3m)^2 \cdot 2\delta}{|F|} = \frac{(|H|+1) \cdot 36m^2 \cdot \delta}{|F|}.$$

By the fact that two distinct univariate degree-t polynomials agree on at most t points we get that

$$Pr[\neg T_i \wedge T_{i+1} \wedge E_i] \leq Pr[\neg T_i \wedge E_i] \leq \frac{4m \cdot (|H| - 1)}{|F|}.$$

Therefore, we get that (assuming $m > 4$)

$$Pr[\neg T_i \wedge T_{i+1}] \leq \frac{(|H| + 1) \cdot 36m^2 \cdot \delta}{|F|} + \frac{4m \cdot (|H| - 1)}{|F|} \leq \frac{(|H| + 1) \cdot 37m^2 \cdot \delta}{|F|}.$$

Thus, summing the error probabilities for all layers, we get

$$Pr[S_1] \leq d \cdot \frac{(|H| + 1) \cdot 37m^2 \cdot \delta}{|F|}.$$

Let S_2 be the event that although P_2 is the malicious server the referee outputs a wrong result (i.e., $Q_0(0)$ that is not equal to $C(x)$). Using a similar proof, we have that $Pr[S_2]$ is also bounded by the same probability. The only difference is that in this case the degrees of $Q_i(t)$ are smaller than the degrees of $M_i(t)$ by a factor of 2.

Thus, for any constant soundness ε we can take F to be of size $\geq \frac{(|H|+1)\cdot 37m^2\cdot\delta}{\varepsilon}$ which is $poly(|H|)$. □

E Proof of Theorem 4

Proof of Theorem 4. Note that the information that the referee sends for the verification of $C_{spec(L)}$ is independent of the messages for the verification of C_L. Those proofs share only one piece of information, the values of p_j, w_j, w'_j as the inputs for the circuit $C_{spec(L)}$.

Let's assume P'_1 is the cheater. He can cheat either on some value of $\widetilde{\Upsilon}$ or on the computation of C_L. In the first case, he will be caught with high probability by the soundness of the protocol from Section 3.2. For the second case, if P'_1 cheats on the computation of C_L (while the values of $\widetilde{\Upsilon}$ are correct), then it means he can cheat in the protocol from Section 3.2 in the case where he has an oracle access to $\widetilde{\Upsilon}$.

By a union bound of the cheating probabilities of the $3d + 1$ invocations of the protocol, we can bound the probability of cheating by $(3d + 1) \cdot d \cdot \frac{(|H|+1)\cdot 37m^2\cdot\delta}{|F|}$. Thus, for any constant soundness ε we can take F to be of size $\geq (3d + 1) \cdot d \cdot \frac{(|H|+1)\cdot 37m^2\cdot\delta}{\varepsilon}$ which is $poly(|H|)$.

□

F Merkle Hash Tree

Merkle Hash Tree [Mer88] allows one to hash down a long string of n characters in a way that he can later reveal any part of the hashed string along with a proof of correctness of length $\kappa \cdot \log(n)$, where κ is the output length of the hash function. The idea is that given a collision-resistant hash function $H : \{0,1\}^s \to \{0,1\}^t$ such that $s \geq 2t$, and a string $st[1 : n]$, the algorithm iteratively computes a tree of hash values. The lowest level of the tree has the values of $h_i = H(st[i])$ for $i = 1 \ldots n$. The next level has the values of $H(h_i \circ h_{i+1})$ for $i = 1, 3, \ldots n - 1$, and so on to the other levels. The highest level, the root, will be the hash of $st[1 : n]$ (for simplicity we assume that n is a power of 2).

When asked to reveal $st[m]$, the algorithm answers with the value of $st[m]$ together with the hash values along the path to h_m and their siblings. This will be the *proof of consistency*. Using this information, anyone can check that the hash values in this path were computed properly and that they correspond to the published root value.

G Extensions of Protocol II

Reducing the number of rounds. In some scenarios, the number of rounds might still be the bottle-neck of the protocol. We can reduce the number of rounds by permitting larger messages and longer running time of the servers.

For any constant number t we can reduce the number of rounds to $\log_{t+1} T(n)$ (but slightly increase the communication size, by a factor of t) using the following idea. Instead of asking the servers for only one reduced configuration in each round, the referee asks for t reduced configurations. Specifically, the referee asks for the t steps that are equally spread between n_g and n_b. I.e., given $n_g = 100, n_b = 200, t = 4$, the referee asks for $120, 140, 160$ and 180. Similar to the protocol from Section 4, the referee updates n_g and n_b according to the servers' answers and continues to the next iteration of the binary (or $(t+1)$-ary) search.

Reducing the size of communication. We can reduce the size of the communication by a factor of $\log S(n)$. Recall that in the protocol from Section 4, in each round of the binary-search the servers send a reduced configuration, which is of size $\log S(n)$. Instead, they could send only the hash of root of the Merkle Hash Tree, $MH_{root}(st)$. Later on, after the referee finds n_g and n_b such that $n_b = n_g + 1$, the referee asks specifically for the proof of consistency for those configurations. Then, he checks the correctness of the answers as in the original protocol.

More than two servers. In addition to the general method for extending the protocol to N servers from Section 2.2, we can extend this specific protocol also in the following way. The referee executes a *Playoff* between all servers. In the first round, the referee executes the protocol from Section 2.2 with all servers (he can do that because the protocol uses only public communication), where he marks a row as a good row only if all answers for this row match. At the end of the binary search, the referee checks if the reduced configurations are consecutive for each one of the servers. After the execution of this protocol, at least one malicious server will be caught lying and will be declared as a cheater. The referee continues to the next round with the other servers, again, executes the protocol to find at least one cheater and then excludes him (or them) from the next rounds. The protocol ends when all the remaining servers agree on the output.

Since the referee excludes at least one malicious server in each round of the playoff, the number of rounds is bounded by the number of malicious servers.