On Black-Box Verifiable Outsourcing

Amit Agarwal[1][0000−0002−7642−1341], Navid Alamati[2][0000−0001−8621−7486], Dakshita Khurana[1][0000−0001−5315−4503], Srinivasan Raghuraman[3][0000−0001−6737−6991], and Peter Rindal[2][0000−0003−2402−7411]

[1] University of Illinois Urbana-Champaign, USA
{amita2,dakshita}@illinois.edu
[2] Visa Research, USA
{nalamati,perindal}@visa.com
[3] Visa Research and MIT, USA
srini131293@gmail.com

Abstract. We study verifiable outsourcing of computation in a model where the verifier has black-box access to the function being computed. We introduce the problem of oracle-aided batch verification of computation (OBVC) for a function class $\mathcal{F}$. This allows a verifier to efficiently verify the correctness of any $f \in \mathcal{F}$ evaluated on a batch of n instances $x_1, \ldots, x_n$, while only making λ calls to an oracle for f (along with $O(n\lambda)$ calls to low-complexity helper oracles), for security parameter λ. We obtain the following positive and negative results:

- We build OBVC protocols for the class of all functions that admit *random-self-reductions*. Some of our protocols rely on homomorphic encryption schemes.

- We show that there cannot exist OBVC schemes for the class of all functions mapping λ-bit inputs to λ-bit outputs, for any $n = \mathsf{poly}(\lambda)$.[4]

1 Introduction

We study the problem of *verifiably outsourcing computation* in a model where the verifier has *black-box access* to the function being computed as well as to certain *low-complexity* helper functions.

A large body of work in the study of delegation, starting with [24,26], consider the setting where a computationally bounded prover generates efficiently checkable proofs π attesting to the correctness of relatively inefficient computation. A major downside of existing works is that they require the prover and verifier to agree on and use a *specific* circuit C_f for computing the function f. In other words, the verification scheme is inherently tied to a fixed (arbitrary) implementation of f which is publicly known to both the prover (server) and the verifier (client).

On the other hand, consider a scenario where a cloud-based service provider offers a service computing f (for example, f can be matrix multiplication) on arbitrary client data. The client would like to ensure correctness of returned outcomes. There are a few reasons why the "circuit-dependent verification" approach above poses a barrier to verifiable computation in this scenario. First of all, the service provider may be using a proprietary code/implementation C_f to compute f (e.g. some proprietary matrix multiplication algorithm) which it is unwilling to disclose to its clients. As such, running a verifiable outsourcing protocol where the client/verifier depends on the code C_f is simply not feasible. Second, even if the company is willing to disclose its code/implementation, the client would have to audit it (for e.g. using formal verification) to make sure that C_f is indeed a sound implementation of f, which can be quite complex. Third, the company may make frequent updates to C_f (for e.g. to add performance optimizations) which would require the client to keep checking this code continually. Finally, making verification independent of the code of f may also lead to efficiency improvements for the verifier in certain settings. Motivated by these questions, we study the following problem:

What classes of functions admit oracle-aided verifiable computation schemes?

The notion of oracle-aided computation captures "circuit-independence" in the context of verifiable computation, as we discuss next. We consider a batch verification scenario: suppose a verifier is given access to an oracle O_f for function $f \in \mathcal{F}$. Is it possible for the verifier, using only $\lambda = \log^2 n$ queries to O_f, to verify the correctness of a large batch of computations $y_1 = f(x_1), \ldots, y_n = f(x_n)$? Oracle access to O_f ensures that the verification scheme is oblivious to any specific implementation C_f that the server may use to perform the computation. Indeed, the client can instantiate such an oracle using any arbitrary implementation C'_f which need not depend on the server's implementation C_f. The restriction of λ oracle queries ensures that even if the oracle O_f is instantiated with a naive/inefficient implementation C'_f on the client side, the total work performed by the client over the entire batch will be relatively small (as long as the security parameter λ is smaller than the batch size).

1.1 Our Results

Motivated by the above considerations, we formalize the notion of oracle aided verifiable computation (OBVC) in the batched setting. At a high level, an OBVC protocol for function class $\mathcal{F}$, defined on ℓ bit inputs, consists of a weak client who wishes to outsource the computation of some function $f \in \mathcal{F}$ on a batch of n instances, let's say $x_1, \ldots, x_n$, to a powerful server. The client is assisted by a function oracle $\mathcal{O}_f$ along with some helper oracles $\mathcal{O}_{g_1}, \ldots, \mathcal{O}_{g_m}$ which are computationally "weaker" than $\mathcal{O}_f$. This is formalized by requiring that the combined time complexity of helper oracles be smaller than the time complexity of the function f i.e. $\sum_{i=1}^{m} T_{g_i}(\ell) = o(T_f(\ell))$. The server can use an arbitrary implementation C_f of the function f. The completeness guarantee of OBVC ensures that the client, when interacting with an honest server (i.e. a server holding

a correct circuit C_f for f and following the protocol steps), always outputs the correct evaluation i.e. $f(x_1), \ldots, f(x_n)$. On the other hand, the soundness guarantee of OBVC ensures that a malicious server (i.e. a server who deviates from the protocol or uses an incorrect circuit C'_f) cannot make the client accept incorrect evaluations on any input in the batch, except with some negligible probability.

We require the scheme to have the following efficiency properties: i) the number of oracle queries made by V to the function oracle $\mathcal{O}_f$ is $O(\lambda)$, ii) the number of queries made to each helper oracle $\mathcal{O}_{g_i}$ is $O(n\lambda)$, iii) there is a constant c such that the running time of the verifier (as an oracle machine) is $\lambda^c \cdot o(n \cdot T_f(\ell))$, where $T_f(\ell)$ is the time complexity of computing f on ℓ bit inputs. Note that the efficiency condition ensures that the OBVC protocol is non-trivial in that the verifier efficiency is better than computing the function on all n inputs in time $n \cdot O(T_f(\ell))$ or, by making $O(n)$ oracle queries to $\mathcal{O}_f$.

Random Self Reducible Functions. In this work, we build an OBVC scheme for the class of all Random Self-Reducible (RSR) functions. We now briefly describe this property. If a function f admits K RSR, then computing f on any chosen input x can be reduced to computing f on a set of uniformly random (not necessarily independent) inputs $r^1, \ldots, r^K$, where K is some fixed constant dependent on f. More formally, there exists a randomized algorithm called RSR.Encode which takes as input x and outputs a set of random instances $r^1, \ldots, r^K$. We will sometimes call these random instances as "shares" of the original input x (borrowing the terminology from secret-sharing literature). Given the evaluation of f on these random instances, $f(r^1), \ldots, f(r^K)$, there exists a deterministic algorithm called RSR.Decode which outputs $f(x)$. Moreover, RSR.Encode and RSR.Decode are much "simpler" to compute than f and this is formalized by requiring that the combined time complexity of RSR.Encode and RSR.Decode is much less than that of f. (Note that these only depend on the functionality f and not on its circuit/implementation.) Many useful functions such as integer multiplication, matrix multiplication, polynomial multiplication, integer division, exponentiation, and trigonometric functions such as sine and cos admit RSR. In our positive result, we assume that the RSR.Encode and RSR.Decode functions are available to the verifier as helper oracles.

Theorem 1. *(Informal) Let $\mathcal{F}_\ell$ be the class of all Random Self-Reducible functions on $\ell = \ell(\lambda)$ bit inputs. Assuming homomorphic encryption scheme (HE) for $\mathcal{F}_\ell$, there exists an* OBVC *scheme for $\mathcal{F}_\ell$.*

In this work, we are also interested in studying the limitations of OBVC schemes. In other words, we would like to understand whether all large classes of functions can admit OBVC schemes. To that end, we have the following result:

Theorem 2. *(Informal) Let $\mathcal{F}_\lambda$ be the class of all functions mapping λ bit inputs to λ bit outputs. Then, $\mathcal{F}_\lambda$ does not admit an* OBVC *scheme.*

We will elaborate upon these two results in the next section.

1.2 Our techniques

Positive result. Let us start by describing a simplified version of our idea (which doesn't directly work). Consider the following protocol: The client sends all n instances, $x_1, \ldots, x_n$, to the server and the server is supposed to respond with $y_1 = f(x_1), \ldots, y_n = f(x_n)$. On receiving $y_1, \ldots, y_n$ from the server, the client performs a cut-and-chose style check on some small subset T, where $|T| = \lambda$ (λ being the security parameter), in the following way: It randomly selects $T \subset [n]$ and checks whether $y_i = \mathcal{O}_f(x_i)$ for all $i \in T$, where $\mathcal{O}_f$ is an oracle that returns the evaluation of f. If the check fails, the client aborts. Otherwise, the client outputs $y_1, \ldots, y_n$. On an intuitive level, if the server is cheating on some instance x_{i_0} where $i_0 \in [n]$, then it runs the risk of being caught in the cut-and-chose check. However, this strategy fails since even if $|T| = n - 1$, the prover can get away with a probability atleast $\frac{1}{n}$, which is non-negligible. Hence this basic scheme does not work.

The major downside of the above scheme is that a malicious server can corrupt the computation on a single instance and go undetected with non-negligible probability. One may attempt to resolve this issue is that of error correction. In more detail, we could force a malicious server to corrupt the computation on many parts of a codeword in order to successfully corrupt the computation on a single instance. This would hopefully reduce the probability of a malicious server going undetected. However, this alone does not suffice. The real issue that the above example highlights is that a malicious server can, *with probability* 1, *selectively* corrupt the computation on a single instance x_i in the batch where $i \in [n]$, error-corrected or otherwise. Unless the verifier is invoking the oracle $\mathcal{O}_f$ on all n instances, it runs the risk of accepting a bad set of $y_1, \ldots, y_n$. This is true even if one employs error correction techniques on each instance as the adversary may be able to identify the error-corrected instances corresponding to each instance. Our idea to tackle this is to leverage the property of Random Self-Reduction (RSR). In the following description, we will assume that we are dealing with the class of functions admitting RSR, and that the RSR.Encode and RSR.Decode functions are available to the verifier as helper oracles.

Suppose our function f of interest admits K RSR with $K = 1$. As a first step, we will show that RSR helps us to reduce the probability of *selective* corruptions from 1 to $\frac{1}{n}$. Looking ahead, our next step will be to show that assuming this lower probability of *selective* corruptions, error-correction tools, i.e., repetition and majority decoding, can be used to achieve negligible soundness error. For our first step, we modify our previous basic protocol in the following way: Instead of sending $x_1, \ldots, x_n$ to the server, we will first map each instance x_i to a uniformly random instance r_i using RSR.Encode, shuffle the set $\{r_1, \ldots, r_n\}$, and send this shuffled set to the server. After receiving the answers from the server, the client will perform a cut-and-chose check as described earlier. If the cut-and-chose check passes, it reverse shuffles the server's responses and applies RSR.Decode to each of them to get the actual outputs. We claim that this protocol reduces the probability of *selective* corruptions to $\frac{1}{n}$, i.e., the prover cannot *selectively* corrupt the computation on a particular instance x_{i_0} with probability better

than $\frac{1}{n}$. This follows because a 1 RSR is a random mapping, and have shuffled the random mappings of the instance as well.

Having achieved this lower probability of *selective* corruptions, we move on to our next and final step for the case of $K = 1$. We claim that we can now boost the soundness of this protocol by performing repetitions and majority decoding in the following way: For each instance x_i in the batch, we apply $\mathsf{RSR.Encode}$ independently λ times, where λ is a security parameter, to get $\{r_{i,j}\}_{i\in[n],j\in[\lambda]}$. We then proceed as described earlier i.e. the client randomly shuffles $\{r_{i,j}\}_{i\in[n],j\in[\lambda]}$, sends this shuffled set to the server and performs cut-and-chose check on the server's responses. If the cut-and-chose check passes, it reverse shuffles the server's responses and applies $\mathsf{RSR.Decode}$ to each of them. Additionally, it performs a majority decoding on the results of $\mathsf{RSR.Decode}$ to get the final outputs. If the cut-and-chose check passes, it ensures that any random subset of size λ of the server's responses will have less than $\frac{\lambda}{2}$ corruptions (except with negligible probability) due to Hoeffding's bound. Note that this holds regardless of having achieved a low probability of *selective* corruptions. But crucially, the low probability of *selective* corruptions allows to translate the guarantee on random subsets of size λ to subsets that precisely correspond to the repetitions of each instance. This, in turn, ensures that the majority decoding for each instance will always result in the correct output. To further illustrate this, note that if we skip the shuffling step (that was partially responsible for a low probability of *selective* corruptions) and only perform random mapping (using $\mathsf{RSR.Encode}$) along with repetitions, it won't get us negligible soundness error. This is because a cheating server can again selectively corrupt only $\{r_{i_0,j}\}_{j\in[\lambda]}$ i.e., all the random instances in every repetition corresponding to a particular input x_{i_0} and avoid detection with non-negligible probability.

We now turn towards the case of functions which admit K RSR where $K > 1$. Compared to $K = 1$ case, this case is much more tricky to handle for the following reason. Suppose we invoke $\mathsf{RSR.Encode}$ on each instance x_i (without any repetitions) to form a set of random instances $\{r_i^1, \ldots, r_i^K\}$. As with the $K = 1$ case, a natural extension of the previous approach in order to thwart *selective* corruptions would be to gather all the $n \cdot K$ random instances $\{r_i^k\}_{i\in[n],k\in[K]}$, shuffle them, and send them to the server. In the $K = 1$ setting, we argued that the prover cannot *selectively* corrupt the computation on a particular instance x_{i_0} with probability better than $\frac{1}{n}$ due to the random mapping and shuffling step. However, this is no longer true for the $K > 1$ case. The reason is that although each individual share in the set $\{r_{i_0}^1, \ldots, r_{i_0}^K\}$, corresponding to a particular instance x_{i_0}, is uniformly random, the joint distribution is not necessarily uniform. For example, it may happen that any two shares in the set $\{r_{i_0}^1, \ldots, r_{i_0}^K\}$ completely reveal the instance x_{i_0}. Therefore, an unbounded server can potentially try a brute force approach to find out which shares correspond to a particular instance x_{i_0} and then selectively corrupt the computation on those shares.

To handle this, we make the following observation. Suppose we are dealing with a restricted kind of "non-communicating" prover $\mathsf{P}_{\mathsf{no\text{-}com}}$. Such a prover is defined as a tuple of K non-communicating provers $\mathsf{P}_{\mathsf{no\text{-}com}} = (\mathsf{P}^1_{\mathsf{no\text{-}com}}, \ldots, \mathsf{P}^K_{\mathsf{no\text{-}com}})$.

While each prover in the tuple can be an arbitrary unbounded machine, the restriction is that they are not allowed to communicate with each other during the protocol execution. The idea then is to modify the protocol in the following manner: Instead of sending all K shares corresponding of each instance x_i to a single prover, we will only send the k^{th} shares of each instance to the k^{th} non-communicating prover $\mathsf{P}^k_{\mathsf{no\text{-}com}}$. On receiving the responses from each $\mathsf{P}^k_{\mathsf{no\text{-}com}}$, the verifier applies an independent cut-and-chose check on the responses sent by each $\mathsf{P}^k_{\mathsf{no\text{-}com}}$. Since each individual prover is now receiving only a single share (for each instance x_i), we can re-apply the soundness logic discussed for the $K = 1$ RSR case after doing λ independent repetitions. This means that for each individual non-communicating prover $\mathsf{P}^k_{\mathsf{no\text{-}com}}$, if the cut-and-chose check passes, then any random subset of size λ of the $\mathsf{P}^k_{\mathsf{no\text{-}com}}$ responses will have less than $\frac{\lambda}{2K}$ [5] corruptions (except with negligible probability) due to Hoeffding's bound. It turns out that ensuring fewer than $\frac{\lambda}{2K}$ corruptions with respect to each instance $i \in [n]$ and prover $\mathsf{P}^k_{\mathsf{no\text{-}com}}$ suffices for the majority decoding argument (as mentioned in the $K = 1$ RSR case) to go through.

Note that eventually we would like to construct a protocol which is sound against a single prover P. To this end, we introduce an intermediate notion of a "no-signaling prover" where we ease the non-communicating restriction in $\mathsf{P}_{\mathsf{no\text{-}com}}$. Formally, a "no-signaling prover" is defined as a tuple of K provers $\mathsf{P}_{\mathsf{no\text{-}com}} = (\mathsf{P}^1_{\mathsf{no\text{-}sig}}, \ldots, \mathsf{P}^K_{\mathsf{no\text{-}sig}})$. While each prover in the tuple can be an arbitrary unbounded machine, the restriction is that for all $k \in [K]$, the distribution of the responses of the k^{th} prover $\mathsf{P}^k_{\mathsf{no\text{-}sig}}$ should be independent of the shares received by the other provers $\{\mathsf{P}^i_{\mathsf{no\text{-}sig}}\}_{i\in[K],i\neq k}$. We then show that our modified protocol for handling arbitrary non-communicating provers is also sound against arbitrary no-signaling provers. Intuitively, the reason why this works is because the cut-and-chose check that we apply on each individual $\mathsf{P}^k_{\mathsf{no\text{-}sig}}$ responses is *local*. In more detail, suppose Pred^k is a binary predicate capturing the following event: there exists $i_0 \in [n]$ such that the server $\mathsf{P}^k_{\mathsf{no\text{-}sig}}$ responds incorrectly to more than $\frac{\lambda}{2K}$ fraction of RSR instances $\{r_{i_0,j}\}_{j\in[\lambda]}$ and the cut-and-chose check on its responses passes. Since this predicate is local, i.e., the predicate output depends only on the responses of $\mathsf{P}^k_{\mathsf{no\text{-}sig}}$, it can be shown that any $\mathsf{P}^k_{\mathsf{no\text{-}sig}}$ which makes Pred^k true with non-negligible probability (over the randomness of the verifier) directly implies a non-communicating prover $\mathsf{P}^k_{\mathsf{no\text{-}com}}$ which makes Pred^k true with non-negligible probability (thus contradicting our soundness analysis for arbitrary non-communicating provers).

Finally, we show that the restriction to a no-signaling set of provers can be removed by a slight modification to the protocol where the verifier simply encrypts each RSR instance $\{r^k_{i,j}\}_{i\in[n],j\in[\lambda],k\in[K]}$ under an independent public-key $\mathsf{pk}_{i,j,k}$ before sending it to a single server P. If the public-key encryption scheme is homomorphic, then the server can compute the answers to verifier messages "under the hood" of the HE scheme (using $\mathsf{HE.Eval}$) and send the encrypted responses

[5] We use $\frac{\lambda}{2K}$ as opposed to $\frac{\lambda}{2}$ as this is what we need in the setting of K provers to make the rest of the analysis work out.

back to the verifier. The verifier then simply decrypts all the responses and runs the no-signaling verifier (which is identical to the non-communicating verifier) to derive the final output. With this transformation, it can be shown that the soundness of the previous protocol (i.e., without applying encryption) against arbitrary unbounded no-signaling provers $\mathsf{P}_{\mathsf{no\text{-}sig}}$ directly implies soundness of the transformed protocol (i.e., after applying encryption) against arbitrary *computationally bounded* provers P. Formally, the analysis uses a reduction to the semantic security of the encryption scheme.

Negative result. Towards a negative result, an ideal goal would be to tightly characterize functions that do not admit an OBVC scheme. However, getting such a strong negative result seems difficult as there might be arbitrary properties of functions (other than RSR) that one could potentially leverage in order to construct an OBVC scheme. Therefore, we settle for a weaker goal where we show that it is impossible to construct an OBVC scheme for a "large enough" function class $\mathcal{F}$. Specifically, we consider the function class $\mathcal{F}_\lambda = \{\{0,1\}^\lambda \to \{0,1\}^\lambda\}$, the class consisting of all functions mapping λ bit inputs to λ bit outputs.

We now adopt the following approach: Suppose there exists a OBVC scheme Π for $\mathcal{F}_\lambda$ and let f_λ be a function sampled randomly from $\mathcal{F}_\lambda$. Then we show that there exists a malicious prover P^* that breaks the soundness of Π with non-negligible probability. Allowing f_λ to be sampled randomly from $\mathcal{F}_\lambda$ enables us to model this game in the well-known Random Oracle Model (ROM) [6]. In this terminology, the oracle $\mathcal{O}_f$ will be identical to a Random Oracle (RO). Let n be the number of instances in the batch and t be the number of queries that V is allowed to make to $\mathcal{O}_f$. For the OBVC scheme to be meaningful, we know that t should be strictly less than n. However, note that in our OBVC definition, we also allow the verifier to have access to $\mathsf{poly}(\lambda)$ function-dependent helper oracles, each of which can be invoked $O(n\lambda)$ times. To model these helper oracles faithfully in ROM, we will assume that these are encoded as an s-bit auxiliary input aux and handed over to the verifier as a preprocessing advice. Note that this aux can depend arbitrarily on the entire RO function table, for example, it can contain global information about the entire RO function f.

Our idea to construct a malicious prover P^* that breaks the soundness of any potential OBVC scheme Π in this ROM setting is as follows. Let $\mathcal{Q}$ denote the set of queries that the V makes to $\mathcal{O}_f$ during the protocol. Since $t < n$, it holds that a randomly sampled instance x_ϕ from the batch $\{x_1, \ldots, x_n\}$ will be outside $\mathcal{Q}$ with probability atleast $1 - t/n$. Therefore, we can switch into a hybrid where the prover locally reprograms the value of $f(x_\phi)$ to a random value Δ in the image of f. Intuitively, one could invoke a lazy-sampling argument for ROM to argue that this change will go unnoticed to the verifier if it does not query $\mathcal{O}_f$ at x_ϕ. Indeed, if this were true, then it would have been sufficient to break soundness with non-negligible probability. However, there is a subtle flaw in directly applying such a lazy-sampling argument. Recall that we are in a setting where the verifier is allowed to compute auxiliary information aux about $\mathcal{O}_f$ before the protocol begins. This hinders a direct application of lazy-sampling argument as aux might potentially contain information (for e.g. a small

digest) about the *entire* $\mathcal{O}_f$. Hence, it is no longer true that points outside $\mathcal{Q}$ are independent from the verifier's view.

To resolve this, we apply some of the techniques that were developed in earlier works [16,17,28] which studied security of cryptographic protocols where adversary can contain auxiliary information about the Random Oracle, also known as the Auxiliary Input Random Oracle (AI-RO) model. We specifically use the results in [16] where authors define a new relaxed model called Bit-Fixing Random Oracle (BF-RO) model. At a high level, in the BF-RO model, the aux is constrained so that it only contains information about p points (p is a tunable parameter) in $\mathcal{O}_f$ which can be chosen arbitrarily. Based on this modeling, the authors show that security theorems proved in BF-RO model can be carried over to the AI-RO model with a loss in advantage proportional to st/p (recall that s is the length of advice string in AI-RO model and t is the number of queries to $\mathcal{O}_f$). By setting s, t, p appropriately, one can get negligible loss in advantage.

Returning to our setting, recall that it was not possible to apply lazy sampling in the AI-RO model we were dealing with. Therefore, as a first step, we will restrict ourselves to the BF-RO model where aux is constrained so that it only contains information about/fixes some p points of the random oracle. Let us denote these set of p points by $\mathcal{P}$. Fortunately, in this model, we can apply the lazy-sampling technique for the points outside $\mathcal{P}$. Therefore, as long as we can ensure that x_ϕ is outside both $\mathcal{P}$ and $\mathcal{Q}$ (recall that $\mathcal{Q}$ is the set of queries that the verifier makes during the protocol), then the malicious prover P^* which we described earlier will work. We show formally that this is indeed the case for all $\alpha' \in (0, 1], p \in 2^{(1-\alpha')\lambda}$, thus giving us an impossibility result for OBVC in the BF-RO model. Finally, we are also able to apply a lemma from [16] to lift our impossibility result from the BF-RO model to the AI-RO model with appropriate setting of parameters.

1.3 Related Work

Our idea of verifiable computation of functions in a "circuit-independent" fashion is inspired from the early works on Self-Testing/Self-Correcting programs [7,25]. In these works, it was shown that if a program P correctly computes a random self-reducible (RSR) function f on "most" inputs, then it can be used to correctly compute f on "all" inputs using only oracle access to P. However, a major limitation of these works is that the adversarial program is limited to a stateless machine. In other words, the response provided by P on a particular query is not allowed to depend on the previous queries. In our work, we consider the setting of arbitrary stateful prover which is strictly general than a stateless program.

Later works [8] extended this idea to deal with adaptive programs (i.e. programs whose response in a particular query can depend on the previous queries arbitrarily) but protocols in this setting required two or more independent copies of the program which, analogously, can be thought of as non-communicating provers. This work requires an additional property of "downward self-reducibility" (which roughly means that computing f on input x of size ℓ can be reduced to computing f on random "smaller" instances of size $\ell - 1$).

Thus, our result, which only relies on random-self-reducibility to instances of the same size, is more general.

Rubinfeld [27] extended the work on program checking to a batched setting where the verifier is trying to verify the computation of P on batch of n inputs. Again, this work was limited to stateless program as opposed to stateful prover which we consider. Bellare et. al. [5] proposed a different approach to batch verification for the specific case of modular exponentiation function by allowing the verifier to compute the function on some small number of inputs on its own.

As discussed earlier in the introduction, succinct non-interactive arguments (SNARGs) for P (where proof size and verification time are polylogarithmic in the security parameter) and batch arguments (BARGs) for NP, where a batch of statements can be verified in time that is sublinear in the number of statements [10,22,12,13,21,29] are closely related primitives. A related line of work [19,14,2] similarly considers the possibility of using FHE and a preprocessing stage to perform verifiable computation. Unfortunately, all of these works require the verifier to have non-black-box access to the circuit C_f for the function f, and are therefore not applicable to the setting of black-box verification.

2 Preliminaries

Throughout the paper, we use bold-letters to indicate vectors (which can sometimes be equivalently represented as strings). For a vector $\mathbf{v}$ of length n, we use the notation v_i to indicate the i^{th} co-ordinate of $\mathbf{v}$ where $i \in [n]$. For a subset $S \subseteq [n]$, we use $\mathbf{v}_S := (v_i)_{i \in S}$ to denote the subvector of $\mathbf{v}$ restricted to the positions $i \in S$. For a bit string $\mathbf{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$ of arbitrary length $n \geq 0$, we use $\mathsf{RW}(\mathbf{b})$ and $\mathsf{HW}(\mathbf{b})$ to indicate the relative and absolute hamming weight of $\mathbf{b}$ respectively. Throughout the paper, we use λ to indicate the security parameter. By $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, we mean the class $\lambda^{O(1)}$ and $\frac{1}{\lambda^{\omega(1)}}$. We sometimes abuse notation and use $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to refer to a member from the class $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ respectively. Given a security parameter λ, we use PPT to denote probabilitic $\mathsf{poly}(\lambda)$-time Turing Machines and non-uniform PPT to denote PPT machines with $\mathsf{poly}(\lambda)$-sized advice. We say that two distribution ensembles $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}$ are computationally indistinguishable, denoted by $X \approx_c Y$, if for every non-uniform PPT algorithm $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, we have $|\Pr[\mathcal{D}(X_\lambda) = 1] - \Pr[\mathcal{D}(Y_\lambda) = 1]| \leq \mathsf{negl}(\lambda)$.

2.1 Mathematical Preliminaries and Definitions

Theorem 3 (Hoeffding's inequality [20]). *Let $\mathbf{b} \in \{0,1\}^{nm}$ be a bitstring with relative hamming weight $\mu = \mathsf{RW}(\mathbf{b})$. Let the random variables $X_1, \ldots, X_k$ be obtained by sampling k entries from $\mathbf{b}$ with replacement, i.e. the X_i's are independent and $\Pr[X_i = 1] = \mu$. Furthermore, let the random variables $Y_1, \ldots, Y_k$ be obtained by sampling k entries from $\mathbf{b}$* without *replacement. Then, for any $\delta > 0$, the random variables $\bar{X} = \frac{1}{k}\sum_i X_i$ and $\bar{Y} = \frac{1}{k}\sum_i Y_i$ satisfy:*

$$\Pr[|\bar{Y} - \mu| \geq \delta] \leq \Pr[|\bar{X} - \mu| \geq \delta] \leq 2 \cdot e^{-2\delta^2 k}$$

Definition 1. *An (N, M) source is a random variable X with range $[M]^N$. A source is called p-bit-fixing if it is fixed on at most p coordinates and uniform on the rest.*

Theorem 4 ([16]). *Let X be distributed uniformly over $[M]^N$ and $Z := f(X)$, where $f : [M]^N \to \{0,1\}^s$ is an arbitrary function. For any $\gamma > 0$ and $p \in \mathbb{N}$, there exists a family $\{Y_z\}_{z \in \{0,1\}^s}$ of convex combinations Y_z of p-bit-fixing (N, M)-sources such that for any distinguisher D taking an s-bit input and querying at most $t < p$ coordinates of its oracle,*

$$|\Pr[D^X(f(X) = 1)] - \Pr[D^{Y_{f(X)}}(f(X)) = 1]| \leq \frac{(s + \log 1/\gamma) \cdot t}{p} + \gamma$$

2.2 Bit fixing Random Oracle Model

In this section, we will define the Auxiliary Input Random Oracle ($\mathsf{AI\text{-}RO}$) and Bit fixing Random Oracle ($\mathsf{BF\text{-}RO}$) model as described in Coretti et. al. [16]. An oracle $\mathcal{O}$ consists of two interfaces $\mathcal{O}.\mathsf{pre}$ and $\mathcal{O}.\mathsf{main}$. We will define two types of entities (modeled as turing machines) and their access to $\mathcal{O}$.

- Two-stage entity : Such an entity $\mathcal{E}$ is split up into two parts $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$. The first part $\mathcal{E}_1$ can access $\mathcal{O}.\mathsf{pre}$ and the second part $\mathcal{E}_2$ can access $\mathcal{O}.\mathsf{main}$. Furthermore, $\mathcal{E}_1$ can pass on some auxiliary information to the second part.
- Single-stage entity: Such an entity $\mathcal{E}$ only accesses $\mathcal{O}.\mathsf{main}$.

Let $\mathcal{F}_{M,N}$ be the set of all possible functions $f : [M] \to [N]$. Now we will define different types of oracles that we will use:

- Auxiliary Input Random Oracle $\mathsf{AI\text{-}RO}(M, N)$: Samples a random function table $F \leftarrow \mathcal{F}_{M,N}$; outputs F at $\mathcal{O}.\mathsf{pre}$; answers queries $x \in [M]$ at $\mathcal{O}.\mathsf{main}$ by the corresponding value $F(x) \in [N]$.
- Bit fixing Random Oracle $\mathsf{BF\text{-}RO}(p, M, N)$: Samples a random function table $F \leftarrow \mathcal{F}_{M,N}$; outputs F at $\mathcal{O}.\mathsf{pre}$; takes a list at $\mathcal{O}.\mathsf{pre}$ of at most p query/answer pairs (called "bit-fixing" pairs), $\{(x_i, y_i)\}_{i \in [p]}$, that override F in the corresponding position i.e. $\forall i \in [p]$, we set $F(x_i) = y_i$. Then it answers queries $x \in [M]$ at $\mathcal{O}.\mathsf{main}$ by the corresponding value $F(x) \in [N]$.

2.3 Homomorphic Encryption

A homomorphic (public-key) encryption scheme $\mathsf{HE} = (\mathsf{HE.Keygen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ is a quadruple of PPT algorithms as follows.

- Key Generation: The algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Keygen}(1^\lambda)$ takes a unary representation of the security parameter λ and outputs a public encryption key pk, and a secret decryption key sk.

- Encryption: The algorithm $c \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(\mu)$ takes the public key pk and a single bit message $\mu \in \{0,1\}$ and outputs a ciphertext c. For encrypting ℓ bit messages, we can simply invoke $\mathsf{HE.Enc}$ bit-by-bit.
- Decryption: The algorithm $\mu^* \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(c)$ takes the secret key sk and a ciphertext c and outputs a message $\mu^* \in \{0,1\}$.
- Homomorphic Evaluation: The algorithm $c_f \leftarrow \mathsf{HE.Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_\ell)$ takes the public key pk, a function $f : \{0,1\}^\ell \rightarrow \{0,1\}$ and a set of ciphertexts $c_1, \ldots, c_\ell$ and outputs a ciphertext c_f[6].

As mentioned in [11], the representation of function f can vary between schemes, and it is best to leave this issue outside of the syntactic definition for our purposes.

The above algorithms must satisfy the following properties:

- CPA-security: A scheme HE is IND-CPA secure if the following holds:

$$\{c \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(0) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Keygen}(1^\lambda)\}_\lambda$$

$$\approx_c$$

$$\{c \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(1) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Keygen}(1^\lambda)\}_\lambda$$

  where $\lambda \in \mathbb{N}$.
- $\mathcal{F}$-homomorphism: Let $\mathcal{F}_\ell \subseteq \{\{0,1\}^\ell \rightarrow \{0,1\}\}$ be a set of functions where $\ell = \ell(\lambda)$. A scheme HE is $\mathcal{F}$-homomorphic (or, homomorphic for the class $\mathcal{F}$) if for any sequence of functions $f_\ell \in \mathcal{F}_\ell$ and respective inputs $\mu_1, \ldots, \mu_\ell \in \{0,1\}$, it holds that:

$$\Pr\left[\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_\ell)) \neq f(\mu_1, \ldots, \mu_\ell) : \begin{array}{c}\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{HE.Keygen}(1^\lambda) \\ \forall i \in [\ell], c_i \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(\mu_i)\end{array}\right] = \mathsf{negl}(\lambda)$$

- Compactness: A scheme HE is compact if there exists a polynomial $s = s(\lambda)$ such that the output length of $\mathsf{HE.Eval}$ is at most s bits long (regardless of f or the number of inputs).

2.4 Random Self Reducibility

Intuitively, a function f has Random Self Reducibility (RSR) property if computing f on a given input x can be "easily" reduced to computing f on uniformly random inputs. We now provide a formal definition inspired by [4,7].

Definition 2 (Random Self Reduction (RSR)). *A function $f : \mathsf{D} \rightarrow \mathsf{R}$ is K random self reducible (henceforth denoted by K-RSR) if there exists a pair of algorithms* $(\mathsf{RSR.Encode}, \mathsf{RSR.Decode})$ *where,*

[6] For syntactic simplicity, we only consider functions with a single bit output. The generalization to functions with arbitrary output length can be done by splitting a multi-bit output function into multiple functions with single bit output.

- $\mathsf{RSR.Encode}(x)$ *: This is a randomized algorithm which takes an* ℓ *bit input* $x \in \{0,1\}^\ell \cap \mathsf{D}$ *and outputs* K *values* $r_1, \ldots r_K$*, where each* $r_i \in \{0,1\}^\ell \cap \mathsf{D}$*. It also outputs a state* st*.*
- $\mathsf{RSR.Decode}(\{y_1, \ldots, y_K\}, \mathsf{st})$*: This is a deterministic algorithm which takes as input* K *values* $\{y_i\}_{i \in [K]}$ *from* R*, along with a state* st*, and outputs a value* $y \in \mathsf{R}$*.*

The above algorithms must satisfy the following properties.

- *Correctness: For all* $\ell \in \mathbb{N}$ *and* $x \in \{0,1\}^\ell \cap \mathsf{D}$*, we have:*

$$\Pr\left[\mathsf{RSR.Decode}(\{y_1, \ldots, y_K\}, \mathsf{st}) = f(x) : \begin{matrix} \{r_1, \ldots, r_K\}, \mathsf{st} \leftarrow \mathsf{RSR.Encode}(x) \\ \forall i \in [K] : y_i := f(r_i) \end{matrix}\right] = 1$$

- *Uniformity: For all* $\ell \in \mathbb{N}$*,* $x \in \{0,1\}^\ell \cap \mathsf{D}, i \in [K]$*,*

$$\{r_i : r_1, \ldots, r_K \leftarrow \mathsf{RSR.Encode}(x)\} \equiv \mathcal{U}_\ell$$

  where $\mathcal{U}_\ell$ *is the uniform distribution on* ℓ *bit strings.*
- *Efficiency: Let* $T_{\mathsf{RSR.Encode}}(\ell)$ *and* $T_{\mathsf{RSR.Decode}}(\ell)$ *be the time complexity of* $\mathsf{RSR.Encode}$ *and* $\mathsf{RSR.Decode}$ *respectively on inputs of size* ℓ*. Let* $T_f(\ell)$ *be the (worst-case, over all inputs of size* ℓ*) time complexity of computing* f[7]*. Then, the efficiency condition requires that for all constants* $c > 0$*:*

$$T_{\mathsf{RSR.Encode}}(\ell) + T_{\mathsf{RSR.Decode}}(\ell) = o(T_f(\ell))$$

Blum et. al. [7] showed that many interesting and useful functions, such as modular multiplication, modular exponentiation, integer division, matrix multiplication, polynomial multiplication (over a ring) admit efficient random self reductions. Later works also extended RSR to trigonometric functions such as sine and cosine [15,3], and real-valued functions such as floating-point exponentiation and floating point logarithm [18].

2.5 No-signaling prover

We define the notion of no-signaling prover in a manner similar to prior works [9,23]. Intuitively, for a no-signaling set of provers $\mathsf{P}_{\mathsf{no\text{-}sig}} = (\mathsf{P}_1, \ldots, \mathsf{P}_K)$, the response of each prover P_i is allowed to depend on the queries to all provers as a function but the distribution of each prover's response (modeled as a random variable) should be (computationally) independent of the queries sent to the other provers.

[7] In cases where $T_f(\ell)$ is not known, due to circuit lower bound barriers, we can fix $T_f(\ell)$ to be the best known time complexity for computing f on (worst-case) inputs of size ℓ. For example, if f is the matrix multiplication function of two $\ell \times \ell$ bit matrices, then we can set $T_f(\ell) = \ell^{2.3728596}$ for inputs of length $2\ell^2$ (encoding two $\ell \times \ell$ sized bit-matrix as a bit-string) based on the fastest known matrix multiplication algorithm [1]

Definition 3 (No-signaling prover). *Let $\mathcal{Q}$ denote the alphabet of the queries. A prover system $\mathsf{P}_{\mathsf{no\text{-}sig}} = (\mathsf{P}_1, \ldots, \mathsf{P}_K)$ is called a no-signaling multi-prover system if the following holds:*

$$\left\{\mathsf{Game}_k^0(x, \{y_0^i\}_{i\in[K],i\neq k}, \{y_1^i\}_{i\in[K],i\neq k})\right\}_{k\in[K],x\in\mathcal{Q},y_0^i\in\mathcal{Q},y_1^i\in\mathcal{Q}}$$
$$\approx_c \left\{\mathsf{Game}_k^1(x, \{y_0^i\}_{i\in[K],i\neq k}, \{y_1^i\}_{i\in[K],i\neq k})\right\}_{k\in[K],x\in\mathcal{Q},y_0^i\in\mathcal{Q},y_1^i\in\mathcal{Q}}$$

where the games are formally defined below:

$\mathsf{Game}_k^0(x, \{y_0^i\}_{i\in[K],i\neq k}, \{y_1^i\}_{i\in[K],i\neq k})$		$\mathsf{Game}_k^1(x, \{y_0^i\}_{i\in[K],i\neq k}, \{y_1^i\}_{i\in[K],i\neq k})$	
1 :	*Send x to P_k.*	1 :	*Send x to P_k.*
2 :	$\forall i \in [K], i \neq k$: *send y_0^i to P_i.*	2 :	$\forall i \in [K], i \neq k$: *send y_1^i to P_i.*
3 :	*Receive z from P_k.*	3 :	*Receive z from P_k.*
4 :	*Output z.*	4 :	*Output z.*

3 Defining Oracle-aided Batch Verifiable Computation

We provide two definitions for Oracle-aided Batch Verifiable Computation - one in the single server setting (OBVC) and the other in multi-server setting (MOBVC).

Definition 4 (Oracle-aided Batch Verifiable Computation). *Let $\ell \in \mathbb{N}$ parameterize input length, $m = \mathsf{poly}(\ell)$ for some polynomial $\mathsf{poly}(\cdot)$, n denote a number of instances, and λ denote a security parameter. Let f_ℓ be an arbitrary function in a class $\mathcal{F}_\ell \subseteq \{\{0,1\}^\ell \to \{0,1\}^*\}$, and let $\mathcal{X} = \{0,1\}^\ell$ denote the domain of f_ℓ.*

An oracle-aided batch verifiable computation OBVC *for the function class $\mathcal{F}_\ell$ is an interactive protocol between a randomized client/verifier* V *and a deterministic server/prover* P*, with the following syntax.*

- *The client* V *obtains input a batch of n inputs, $\mathbf{x} = x_1, \ldots, x_n$, where each $x_i \in \mathcal{X}$.*
- *The server* P *obtains a circuit C_f for computing f.*
- *The client* V *interacts with the server* P*, and can additionally make oracle calls to a function oracle $\mathcal{O}_f$ as well as to m helper oracles $\mathcal{O}_{g_1}, \ldots, \mathcal{O}_{g_m}$. Finally,* V *outputs* OUT *where* OUT *is either $y_1, \ldots, y_n$ where $y_i \in \mathsf{Range}(f)$ or $\mathsf{OUT} = \perp$.*

The protocol satisfies the following properties.

- *Non-triviality: The combined time complexity of helper oracles is smaller than the time complexity of the function f i.e. $\sum_{i=1}^m T_{g_i}(\ell) = o(T_f(\ell))$.*

- *Completeness: Let* $\mathsf{OUT}(\langle \mathsf{P}(C_f), \mathsf{V}^{\mathcal{O}_f, \{\mathcal{O}_{g_i}\}_{i\in[m]}}\rangle)$ *denote the output of* V *at the end of protocol. For all* $l \in \mathbb{N}, f_l \in \mathcal{F}_l, n \in \mathbb{N}$, $\mathbf{x} \in \mathcal{X}^n, \lambda \in \mathbb{N}$,

$$\Pr_{\mathsf{V}}[\mathsf{OUT} = f_l(x_1), \ldots, f_l(x_n)] = 1$$

  where the probability is taken over the internal coin tosses of V.
- *Soundness: There exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t. for all adversarial* P^**, for all* $l \in \mathbb{N}, f_l \in \mathcal{F}_l, n = \mathsf{poly}(\lambda), \mathbf{x} \in \mathcal{X}^n$, $\lambda \in \mathbb{N}$,

$$\Pr_{\mathsf{V}}[\mathsf{OUT} = f(x_1), \ldots, f(x_n) \vee \mathsf{OUT} = \perp] \geq 1 - \mathsf{negl}(\lambda)$$

  where the probability is taken over the internal coin tosses of V.
  When referring to computational soundness, we quantify over all non-uniform PPT provers P^*.
- *Privacy: For all adversarial* P^**, there exists a simulator* $\mathsf{Sim}_{\mathsf{P}}$ *s.t. there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t. for all* $\lambda \in \mathbb{N}, f_\lambda \in \mathcal{F}_\lambda, n \in \mathbb{N}, \mathbf{x} \in \mathcal{X}^n$,

$$\mathsf{VIEW}(\mathsf{P}^*) \approx_c \mathsf{Sim}(1^\lambda, 1^n, \mathcal{X})$$

- *Efficiency: For every* $\ell \in \mathbb{N}, f_\ell \in \mathcal{F}_\ell, n \in \mathbb{N}, x \in \mathcal{X}^n$ *and* $\lambda \in \mathbb{N}$*, the number of oracle queries made by* V *to the function oracle* $\mathcal{O}_f$ *is* $O(\lambda)$ *and the number of queries made to each helper oracle* $\mathcal{O}_{g_i}$ *is* $O(n\lambda)$*. Furthermore, there is a constant* c *such that the running time of the verifier (as an oracle machine) is* $\lambda^c \cdot o(n \cdot T_f(\ell))$.

Note that the efficiency condition ensures that the OBVC protocol is non-trivial in the sense that the V is doing something better than the trivial strategies where it computes the function on all n inputs on its own using an internal algorithm in time $n \cdot O(T_f(\ell))$ or, alternatively, does the same task by making $O(n)$ oracle queries to $\mathcal{O}_f$.

We now define K-Multi-server Oracle-aided Batch Verifiable Computation (K-MOBVC) which is a straightforward generalization of the single server definition to a multi-server/multi-prover system $\mathsf{P} = (\mathsf{P}_1, \ldots, \mathsf{P}_K)$ with K provers. Also, in this definition, we do not require the privacy condition.

Definition 5 (Multi-server Oracle-aided Batch Verifiable Computation). *Refer to the full version of this paper.*

4 Protocol for functions admitting 1-RSR

In the following section, we provide a construction of OBVC scheme for functions admitting 1-RSR. The idea behind our protocol is simple: First the verifier maps each of its instance x_i to a uniformly random instance s_i using the $\mathsf{RSR.Encode}$ function. Then it sends all the randomized instances $\{s_i\}_{i\in[n]}$ to the prover in a shuffled order, and the prover is supposed to respond back with $\{f(s_i)\}_{i\in[n]}$. Intuitively, this shuffling, coupled with the fact that $\mathsf{RSR.Encode}$ outputs a uniformly random sample, prevents a malicious prover from selectively providing

incorrect responses on some instances (for e.g. the seventh instance x_7). However, note that a malicious prover might still provide incorrect responses on some indices not knowing which instances they correspond to. To tackle this, the verifier uses a cut-and-choose based checking mechanism. Specifically, it selects a small random subset of the indices, gets the correct answer for those indices from the oracle $\mathcal{O}_f$, and then checks whether the prover's responses match. This check ensures that if the prover is misbehaving on "too many" indices, then he will be caught with "overwhelming" probability. Formally, once the check passes, it is ensured that the prover is not lying on more than some (fixed) constant fraction of indices except with some negligible probability. However, note that, our soundness condition requires the output of the verifier be correct on *all* instances (and not just *most* of the instances). To achieve this, we perform a parallel repetition of each instance for some security parameter λ many times and then select the majority of responses as the correct answer. Intuitively, we can select our parameters in a way so that if the cut-and-chose check passes, then it is ensured that the majority, among λ repetitions, encodes the correct answer for that instance.

Theorem 5. *There exists a* OBVC *scheme, specifically Protocol 4, for the class* $\mathcal{F}_\ell^{1\text{-}RSR}$ *consisting of all* ℓ *bit functions that admit* 1-RSR *with soundness against arbitrary unbounded provers.*

Corollary 6. *For all* $0 < \delta < 1$, $n \in O(2^{\lambda^\delta})$, *Protocol 4 is an* OBVC *scheme for* $\mathcal{F}_\ell^{1\text{-}RSR}$ *with soundness error* $\mathsf{negl}(\lambda)$. *Alternatively, one could set* $\lambda = \omega(\log n)$ *and get a soundness error of* $\mathsf{negl}(n)$.

In the rest of this section, we will prove Theorem 5. We note that the completeness of our protocol follows directly from the correctness property of RSR. We now proceed to discuss non-triviality, privacy, efficiency and prove soundness.

Non-triviality, Privacy and Efficiency Analysis. In our protocol, the verifier uses two helper oracles namely $\mathcal{O}_{\mathsf{RSR.Encode}_f}$ and $\mathcal{O}_{\mathsf{RSR.Decode}_f}$. By Definition 2, we know that $T_{\mathsf{RSR.Encode}}(\ell) + T_{\mathsf{RSR.Decode}}(\ell) = o(T_f(\ell))$. Hence, our protocol satisifes the non-triviality condition.

The privacy of our scheme follows directly from the uniformity condition of RSR. More formally, the simulator $\mathsf{Sim}(1^\lambda, 1^n, \mathcal{X})$ simply samples $n\lambda$ uniformly random instances from $\mathcal{X}$ and outputs it. Since each share $s_{i,j}$ in Protocol 4 is a uniformly random and independent (from everything else) element from $\mathcal{X}$, the simulation is perfect.

For efficiency, we note that each helper oracle is invoked exactly $n\lambda$ times, the function oracle $\mathcal{O}_f$ is invoked exactly λ times and the running time of V is exactly $O(n\lambda)$ as shuffling, majority and cut-and-chose check can be computed in linear time.

Protocol 4

Common input: 1^λ, 1^n

V's additional input: Inputs $x_1, \ldots, x_n$, oracle $\mathcal{O}_f$, helper oracles $\mathcal{O}_{\mathsf{RSR.Encode}_f}$, $\mathcal{O}_{\mathsf{RSR.Decode}_f}$

P's additional input: Circuit C_f for computing f.

1. $\forall i \in [n]$, V generates λ independent RSR instances, $s_{i,1}, \ldots, s_{i,\lambda}$, where $s_{i,j}, \mathsf{st}_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Encode}_f}(x_i)$. It sets $\mathbf{s} := s_{1,1}, \ldots, s_{1,\lambda}, \ldots, s_{n,1}, \ldots, s_{n,\lambda}$.
2. V samples a random permutation π on $[n\lambda]$ and sets $\mathbf{s}' := \pi(\mathbf{s})$. It sends $\mathbf{s}'$ to P.
3. $\forall i \in [n], j \in [\lambda]$, P computes $z'_{i,j} = C_f(s'_{i,j})$.
4. P sets $\mathbf{z}' := z_{1,1}, \ldots, z_{1,\lambda}, \ldots, z_{n,1}, \ldots, z_{n,\lambda}$ and sends $\mathbf{z}'$ to V.
5. V samples a random subset $T \subset [n] \times [\lambda]$ of size λ and checks whether the following holds:

$$\forall (i,j) \in T : z'_{i,j} = f(s'_{i,j})$$

6. If the check fails, then V outputs $\perp$. Otherwise it proceeds.
7. V computes $\mathbf{z} = \pi^{-1}(\mathbf{z}')$.
8. $\forall i \in [n], j \in [\lambda]$, V computes $u_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Decode}_f}(z_{i,j}, \mathsf{st}_{i,j})$.
9. $\forall i \in [n]$, V computes $u_i^{\mathsf{final}} = \mathsf{Majority}(u_{i,1}, \ldots, u_{i,\lambda})$.
10. V outputs $u_1^{\mathsf{final}}, \ldots, u_n^{\mathsf{final}}$.

Soundness Analysis. The high level intuition behind the soundness is the following: If the checking phase in Protocol step 5, 6 passes, then with high probability the verifier will output correct values i.e. with high probability, *all* u_i^{final} will equal $f(x_i)$. To prove this, we will have to show that, for each $i \in [n]$, the majority of $\{u_{i,j}\}_{j \in [\lambda]}$ will be equal to $f(x_i)$ (with high probability) if the testing phase passes. To do so, we first consider the following experiment which basically captures the execution of Protocol 4 with an arbitrary fixed prover P^* and defines random variables $\mathbf{b}$ and its inverse $\mathbf{b}^{\mathrm{inv}}$.

Experiment $\mathsf{Exp}^{\text{1-RSR}}(\mathsf{P}^*, \mathbf{x})$

$\forall i \in [n], j \in [\lambda], s_{i,j} \leftarrow \mathsf{RSR.Encode}(x_i)$

$\mathbf{s} := s_{1,1}, \ldots, s_{1,\lambda}, \ldots, s_{n,1}, \ldots, s_{n,\lambda}$

$\pi \leftarrow$ random permutation on $[n\lambda]$

$\mathbf{s}' := \pi(\mathbf{s})$

$\mathbf{z}' \leftarrow \mathsf{P}^*(\mathbf{s}')$

$T \leftarrow$ random λ sized subset of $[n] \times [\lambda]$

$$\forall i \in [n], j \in [\lambda], b_{i,j} = \begin{cases} 0 & ; z'_{i,j} = f(s'_{i,j}) \\ 1 & ; \text{otherwise} \end{cases}$$

$\mathbf{b} := b_{1,1}, \ldots, b_{1,\lambda}, \ldots, b_{n,1}, \ldots, b_{n,\lambda}$

$\mathbf{b}^{\mathrm{inv}} := \pi^{-1}(\mathbf{b})$

Now, based on the above experiment, we define the advantage of an adversarial prover P^* for an arbitrary instance $\mathbf{x}$:

$$\mathsf{Adv}^{\text{1-RSR}}_{\delta,\Delta}(\mathsf{P}^*, \mathbf{x}) = \Pr\left[\begin{array}{c} \exists i \in [n], \mathsf{RW}(b^{\text{inv}}_{i,1} || \ldots || b^{\text{inv}}_{i,\lambda}) > \delta + \Delta \\ \bigwedge \qquad\qquad : \mathsf{Exp}^{\text{1-RSR}}(\mathsf{P}^*, \mathbf{x}) \\ \mathsf{RW}(\mathbf{b}_T) = 0 \end{array}\right]$$

In a protocol execution with malicious prover P^*, $\mathbf{b}$ will be an arbitrary bitstring. We will now prove some properties about any arbitrary bitstring $\mathbf{b}$ which will enable us to finally establish the soundness claim.

Lemma 1. *Suppose $\mathbf{b} \in \{0,1\}^{n\lambda}$ is an arbitrary bitstring of length $n\lambda$. We sample a uniformly random subset $T \subset [n\lambda]$ and use $\mathbf{b}_T$ to denote the corresponding $|T|$ sized substring of b. Let $B^{\delta}_T = \{\mathbf{b}' \in \{0,1\}^{n\lambda} : |\mathsf{RW}(\mathbf{b}') - \mathsf{RW}(\mathbf{b_T})| < \delta\}$ be the set of all $n\lambda$-length strings which are "δ-close" to the substring b_T in terms of relative Hamming weight. Then, for all $\mathbf{b} \in \{0,1\}^{n\lambda}$ and real-valued $\delta \in (0,1)$:*

$$\Pr_T[\mathbf{b} \notin B^{\delta}_T] \leq 2 \cdot e^{-2\delta^2|T|}$$

where the probability is over the sampling of subset T.

Proof. The proof for the above lemma follows directly from Hoeffding's bound (Theorem 3).

Lemma 2. *Suppose $\mathbf{b} \in \{0,1\}^{n\lambda}$ is an arbitrary bitstring of length $n\lambda$. Let $P_1, \ldots, P_n$ be a random partitioning of the bits of $\mathbf{b}$ where each partition contains exactly λ bits. Then, for all $\mathbf{b} \in \{0,1\}^{n\lambda}$, $\forall i \in [n]$, $\forall \Delta \in (0,1)$:*

$$\Pr[|\mathsf{RW}(\mathbf{b}) - \mathsf{RW}(\mathbf{b}_{P_i})| \geq \Delta] \leq 2 \cdot e^{-2\Delta^2\lambda}$$

where the probability is over the sampling of random partition.

Proof. The proof follows directly from Hoeffding's bound (Theorem 3).

Corollary 7. *Let F denote a indicator random variable denoting the following failure event:*

$$F = \begin{cases} 1 & \exists i \in [n], s.t.\ |\mathsf{RW}(\mathbf{b}) - \mathsf{RW}(\mathbf{b}_{P_i})| \geq \Delta \\ 0 & otherwise \end{cases}$$

Then, we have that:

$$\Pr[F = 1] \leq n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

Proof. The proof follows directly by applying Lemma 2 and union bounding across all n partitions.

Lemma 3. *Suppose* $\mathbf{b}$ *is an arbitrary bitstring from* $\{0,1\}^{n\lambda}$*. We probe a random substring* $\mathbf{b}_T$*, of size* $|T|$*, from* $\mathbf{b}$*. Also, let* $P_1, \ldots, P_n$ *be a random partitioning of the bits of* $\mathbf{b}$ *where each partition contains exactly* λ *bits. Then, for all* $n \in \mathbb{N}, \lambda \in \mathbb{N}$*,* $\mathbf{b} \in \{0,1\}^{n\lambda}$*, real valued* $\delta, \Delta \in (0,1)$*, it holds that:*

$$\Pr\left[\begin{array}{c}\exists i \in [n], \mathsf{RW}(P_i) \geq \delta + \Delta \\ \bigwedge \\ \mathsf{RW}(\mathbf{b}_T) = 0\end{array}\right] \leq 2 \cdot e^{-2\delta^2|T|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

Proof. Consider the following indicator random variables.

$$E_1^{\delta} = \begin{cases} 1 & \mathbf{b} \in \{\mathbf{b}' \in \{0,1\}^{n\lambda} : |\mathsf{RW}(\mathbf{b}') - \mathsf{RW}(\mathbf{b}_T)| \geq \delta\} \\ 0 & \text{otherwise} \end{cases}$$

$$E_2^{\Delta} = \begin{cases} 1 & \exists i \in [n], \text{s.t. } |\mathsf{RW}(\mathbf{b}) - \mathsf{RW}(\mathbf{b}_{P_i})| \geq \Delta \\ 0 & \text{otherwise} \end{cases}$$

$$E_3 = \begin{cases} 1 & \mathsf{RW}(\mathbf{b}_T) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

From the probability bounds from Lemma 1 and Lemma 2, we get the following bound. For all $\mathbf{b} \in \{0,1\}^{n\lambda}$, for all real-valued $\delta, \Delta \in (0,1)$:

$$\Pr[E_1^{\delta} = 1 \wedge E_2^{\Delta} = 1] \leq 2 \cdot e^{-2\delta^2|T|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda} \tag{1}$$

This implies that:

$$\Pr[(E_1^{\delta} = 1 \wedge E_2^{\Delta} = 1) \bigwedge E_3 = 0] \leq 2 \cdot e^{-2\delta^2|T|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

$$\implies \Pr\left[\begin{array}{c}\exists i \in [n], \mathsf{RW}(P_i) \geq \delta + \Delta \\ \bigwedge \\ \mathsf{RW}(\mathbf{b}_T) = 0\end{array}\right] \leq 2 \cdot e^{-2\delta^2|T|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

Claim 1. *For all* $n \in \mathbb{N}, x \in \mathcal{X}^n$ *and for all arbitrary unbounded provers* P^**:*

$$\mathsf{Adv}^{\text{1-RSR}}_{\delta,\Delta}(\mathsf{P}^*, \mathbf{x}) \leq 2 \cdot e^{-2\delta^2|T|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

Proof. This follows directly from Lemma 3 and the definition of $\mathsf{Adv}^{\text{1-RSR}}_{\delta,\Delta}$.

Claim 2. *Fix* $|T| = \lambda$*. Then for all* $0 < \delta < 1$*, for* $n = 2^{\lambda^{\delta}}$*, for all* $\mathbf{x} \in \mathcal{X}^n$ *and for all arbitrary unbounded provers* P^**,*

$$\mathsf{Adv}^{\text{1-RSR}}_{\delta=0.25,\Delta=0.25}(\mathsf{P}^*, \mathbf{x}) = \mathsf{negl}(\lambda)$$

Proof. By setting $\delta = 0.25$, $\Delta = 0.25$ in Claim 1, we get:

$$\mathsf{Adv}^{\text{1-RSR}}_{\delta=0.25,\Delta=0.25}(\mathsf{P}^*, \mathbf{x}) \leq \frac{2}{2^{0.18|T|}} + \frac{2n}{2^{0.18\lambda}}$$

For $n \leq 2^{0.17\lambda}$ and $|T| = \lambda$, we get,

$$\begin{aligned}\mathsf{Adv}^{\text{1-RSR}}_{\delta=0.25,\Delta=0.25}(\mathsf{P}^*, \mathbf{x}) &\leq \frac{2}{2^{0.18\lambda}} + \frac{2n}{2^{0.18\lambda}} \\ &= \mathsf{negl}(\lambda)\end{aligned}$$

which proves the claim.

Remark 1. Claim 2 shows that one of the following two events will happen (except with some negligible probability): 1) the relative hamming weight in each random partition P_i of $\mathbf{b}$ is less than 0.5 or 2) the relative hamming weight of the random substring $\mathbf{b}_T$ is non-zero. In Case 1, this implies that for all $i \in [n]$, more than 50% of the $z_{i,j}$ are correct. This ensures that for all $i \in [n]$, more than 50% of $\{u_{i,j}\}_{j\in[\lambda]}$ will equal to $f(x_i)$. If this happens, for all $i \in [n]$, u_i^{final} will be equal to $f(x_i)$ due to the majority rule. In Case 2, the verifier will simply detect and abort as prescribed in Step 5 and 6 of the protocol. This concludes our soundness analysis.

5 Protocol for functions admitting K-RSR

In this section, we will extend the basic protocol from Section 4 to the more general case of functions which admit K-RSR for any constant $K > 1$. As an intermediate step, we will construct a protocol which is sound against a restricted class of provers. Specifically, we will consider a setting where the prover is a tuple of K no-signaling provers as defined in Definition 3. Finally, we will show how this "no-signaling" constraint can be computationally enforced using homomorphic encryption. Our final protocol will be sound against an arbitrary non-uniform PPT prover P.

5.1 OBVC with multiple provers

Protocol 5.1 describes our OBVC construction for functions that admit K-RSR. At a high level, the protocol is a simple extension of Protocol 4 in the following way: In K-RSR, each invocation of $\mathsf{RSR.Encode}(x_i)$ will yield K shares, each being uniformly random (although jointly they may be not). The verifier simply executes K instances of the protocol for 1-RSR setting where the k^{th} prover P_k receives all the k^{th} shares. In the end, the verifier simply aggregates the result from all the K provers and computes the output.

Theorem 8. *There exists a K-MOBVC scheme, specifically Protocol 5.1, for the class $\mathcal{F}_\ell^{K\text{-}RSR}$ consisting of all ℓ bit functions that admit K-RSR for any $K \geq 1$ with soundness against arbitrary unbounded no-signaling provers $\mathsf{P}_{\mathsf{no\text{-}sig}} = (\mathsf{P}_{\mathsf{no\text{-}sig}_1}, \ldots, \mathsf{P}_{\mathsf{no\text{-}sig}_K})$.*

Corollary 9. *For all $0 < \delta < 1$, $n \in O(2^{\lambda^\delta})$, Protocol 5.1 is an MOBVC scheme for $\mathcal{F}_\ell^{K\text{-}RSR}$ with soundness error $\mathsf{negl}(\lambda)$. Alternatively, one could set $\lambda = \omega(\log n)$ and get a soundness error of $\mathsf{negl}(n)$.*

In the rest of this section, we will prove Theorem 8. We note that the completeness of Protocol 5.1 follows directly from the correctness property of RSR. We now proceed to discuss non-triviality, efficiency and prove soundness.

Protocol 5.1

Common input: 1^λ, 1^n, f

V's additional input: Inputs $x_1, \ldots, x_n$, oracle $\mathcal{O}_f$, helper oracles $\mathcal{O}_{\mathsf{RSR.Encode}_f}$, $\mathcal{O}_{\mathsf{RSR.Decode}_f}$.

P's additional input: Circuit C_f for computing f.

1. For each x_i, V generates λ independent RSR instances. Formally, $\forall i \in [n], j \in [\lambda]$: $\{s_{i,j,k}\}_{k\in[K]}, \mathsf{st}_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Encode}_f}(x_i)$.
2. $\forall k \in [K]$, the following steps are performed:
   (a) V sets $\mathbf{s}^k := s_{1,1,k}, \ldots, s_{1,\lambda,k}, \ldots, s_{n,1,k}, \ldots, s_{n,\lambda,k}$.
   (b) V samples a random permutation π^k on $[n\lambda]$ and sets $\mathbf{s'}^k := \pi^k(\mathbf{s}^k)$. It sends $\mathbf{s'}^k$ to P_k.
   (c) $\forall i \in [n], j \in [\lambda]$, P_k computes $z'_{i,j,k} := C_f(s'^k_{i,j})$
   (d) P_k sets $\mathbf{z'}^k := z'_{1,1,k}, \ldots, z'_{1,\lambda,k}, \ldots, z'_{n,1,k}, \ldots, z'_{n,\lambda,k}$. It sends $\mathbf{z'}^k$ to V.
   (e) V samples a random subset $T^k \subset [n] \times [\lambda]$ of size λ and checks whether the following holds:
   $$\forall (i,j) \in T^k : z'^k_{i,j} = \mathcal{O}_f(s'^k_{i,j})$$
   (f) If the check fails, then V outputs $\perp$. Otherwise it proceeds.
   (g) V computes $\mathbf{z}^k := (\pi^k)^{-1}(\mathbf{z'}^k)$.
3. $\forall i \in [n], j \in [\lambda]$, V computes $u_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Decode}_f}(\{z^k_{i,j}\}_{k\in[K]}, \mathsf{st}_{i,j})$.
4. $\forall i \in [n]$, V sets $u_i^{\mathsf{final}} = \mathsf{Majority}(u_{i,1}, \ldots, u_{i,\lambda})$
5. V outputs $u_1^{\mathsf{final}}, \ldots, u_n^{\mathsf{final}}$

Non-triviality. In our protocol, the verifier uses two helper oracles namely $\mathcal{O}_{\mathsf{RSR.Encode}_f}$ and $\mathcal{O}_{\mathsf{RSR.Decode}_f}$. By Definition 2, we know that $T_{\mathsf{RSR}}.\mathsf{Encode}(\ell)+T_{\mathsf{RSR}}.\mathsf{Decode}(\ell) = o(T_f(\ell))$. Hence, our protocol satisifes the non-triviality condition.

Efficiency. For efficiency, we note that each helper oracle is invoked exactly $n\lambda$ times, the function oracle $\mathcal{O}_f$ is invoked exactly $K\lambda$ times and the running time of V is exactly $O(nK\lambda)$ as shuffling, majority and cut-and-chose check can be computed in linear time. Here K is a constant which depends on the function f (but independent of n, λ and ℓ).

Before proving soundness against no-signaling provers, we consider a relaxed case of "non-communicating" provers as an intermediate step. Such a prover is a tuple of K "non-communicating" local algorithms i.e. $\mathsf{P}_{\mathsf{no\text{-}com}} = (\mathsf{P}_1, \ldots, \mathsf{P}_K)$ where the next-message function of each P_i only depends on the messages it exchanges with V, and not on the interaction of V with other provers $\{\mathsf{P}_j\}_{j\in[K],j\neq i}$.

Soundness analysis for non-communicating provers. We consider the following experiment capturing the execution of Protocol 5.1 with an arbitrary non-communicating prover $\mathsf{P}^*_{\mathsf{no\text{-}com}}$ and defines random variables $\mathbf{b}^k$ and its inverse $\mathbf{b}^{\mathrm{inv}^k}$.

Experiment $\mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}com}}, \mathbf{x})$

$\forall i \in [n], j \in [\lambda], \{s_{i,j,k}\}_{k\in[K]} \leftarrow \mathsf{RSR.Encode}^j(x_i)$

$\forall k \in [K]:$

$\mathbf{s}^k := s_{1,1,k}, \ldots, s_{1,\lambda,k}, \ldots, s_{n,1,k}, \ldots, s_{n,\lambda,k}$

$\pi^k \leftarrow$ random permutation on $[n\lambda]$

$\mathbf{s}'^k := \pi^k(\mathbf{s}^k)$

$\mathbf{z}'^k \leftarrow \mathsf{P}^*_{\mathsf{no\text{-}com}\,k}(\mathbf{s}'^k)$

$T^k \leftarrow$ random λ sized subset of $[n] \times [\lambda]$

$$\forall i \in [n], j \in [\lambda], b^k_{i,j} = \begin{cases} 0 & ; z'^k_{i,j} = f(s'^k_{i,j}) \\ 1 & ; \text{otherwise} \end{cases}$$

$\mathbf{b}^k := b^k_{1,1}, \ldots, b^k_{1,\lambda}, \ldots, b^k_{n,1}, \ldots, b^k_{n,\lambda}$

$\mathbf{b}^{\mathrm{inv}^k} := (\pi^k)^{-1}(\mathbf{b}^k)$

Parse $\mathbf{b}^{\mathrm{inv}^k}$ as $b^{\mathrm{inv}^k}_{1,1}, \ldots, b^{\mathrm{inv}^k}_{1,\lambda}, \ldots, b^{\mathrm{inv}^k}_{n,1}, \ldots, b^{\mathrm{inv}^k}_{n,\lambda}$

Based on the above experiment, we now define the advantage of the k^{th} prover $\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}$, for any arbitrary $k \in [K]$, on an arbitrary instance $\mathbf{x}$ in the following way.

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*{}_{\mathsf{no\text{-}com}_k}, \mathbf{x}) = \Pr\left[\begin{array}{c} \exists i \in [n], \mathsf{RW}(b^{\mathrm{inv}^k}_{i,1} || \ldots || b^{\mathrm{inv}^k}_{i,\lambda}) > \delta + \Delta \\ \bigwedge \qquad : \mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}com}}, \mathbf{x}) \\ \mathsf{RW}(\{b^k_{i,j}\}_{(i,j)\in T^k}) = 0 \end{array}\right] \tag{1}$$

Lemma 4. *For all $n \in \mathbb{N}$, $\lambda \in \mathbb{N}$, $\mathbf{x} \in \mathcal{X}^n$ and for all arbitrary unbounded non-communicating provers $\mathsf{P}^*_{\mathsf{no\text{-}com}} = (\mathsf{P}^*_{\mathsf{no\text{-}com}\,1}, \ldots, \mathsf{P}^*_{\mathsf{no\text{-}com}\,K})$, $k \in [K]$ and real valued $\delta, \Delta \in (0, 1)$,*

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}, \mathbf{x}) \leq 2 \cdot e^{-2\delta^2|T^k|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$$

Proof. This follows from Claim 1 and the fact that each individual share in K-RSR is uniformly random (and hence the view of $\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}$ in Protocol 5.1 is identical to the view of P^* in Protocol 4).

Soundness analysis for no-signaling provers. In this section, we will extend the soundness analysis of Protocol 5.1 from non-communicating multi-provers

to multi-provers who can communicate arbitrarily but follow a special "no-signaling" requirement which we formalize in Definition 3. To do so, we consider an experiment $\mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})$ which captures the execution of Protocol 5.1 with an arbitrary fixed no-signaling prover $\mathsf{P}^*_{\mathsf{no\text{-}sig}} = (\mathsf{P}_{\mathsf{no\text{-}sig}_1}, \ldots, \mathsf{P}_{\mathsf{no\text{-}sig}_K})$ and defines random variables $\mathbf{b}^k$ and its inverse $\mathbf{b}^{\mathrm{inv}^k}$. This experiment is identical to $\mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}com}}, \mathbf{x})$ defined earlier except that we have switched from $\mathsf{P}^*_{\mathsf{no\text{-}com}}$ to $\mathsf{P}^*_{\mathsf{no\text{-}sig}}$.

Based on the experiment $\mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})$, we now define the advantage of the k^{th} prover $\mathsf{P}_{\mathsf{no\text{-}sig}_k}$ in Equation 2 and denote it by $\mathsf{Adv}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}, \mathbf{x})$.

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}, \mathbf{x}) = \Pr\left[\begin{array}{l} \exists i \in [n], \mathsf{RW}(b^{\mathrm{inv}^k}_{i,1} || \ldots || b^{\mathrm{inv}^k}_{i,\lambda}) > \delta + \Delta \\ \bigwedge \qquad : \mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) \\ \mathsf{RW}(\mathbf{b}^k_{T^k}) = 0 \end{array}\right] \tag{2}$$

Lemma 5. *Assume there exists a function $\epsilon(\cdot,\cdot,\cdot,\cdot,\cdot)$ such that for any arbitrary non-communicating multi-prover $\mathsf{P}^*_{\mathsf{no\text{-}com}} = (\mathsf{P}^*_1, \ldots, \mathsf{P}^*_K)$, for all $\delta \in [0,1], \Delta \in [0,1], k \in K$, $n \in \mathsf{poly}(\lambda)$, $x \in \mathcal{X}^n, \lambda \in \mathbb{N}$, it holds that $\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}, \mathbf{x}) \leq \epsilon(\lambda, n, \delta, \Delta, K)$. Then it follows that for any arbitrary no-signaling multi-prover $\mathsf{P}^*_{\mathsf{no\text{-}sig}} = (\mathsf{P}^*_1, \ldots, \mathsf{P}^*_K)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\delta \in [0,1], \Delta \in [0,1], k \in K$, $n = \mathsf{poly}(\lambda)$, $x \in \mathcal{X}^n, \lambda \in \mathbb{N}$, it holds that:*

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}, \mathbf{x}) \leq \epsilon(\lambda, n, \delta, \Delta, K) + \mathsf{negl}(\lambda)$$

Proof. Suppose the lemma is false i.e there exists a no-signaling multi-prover $\mathsf{P}^*_{\mathsf{no\text{-}sig}} = (\mathsf{P}^*_{\mathsf{no\text{-}sig}_1}, \ldots, \mathsf{P}^*_{\mathsf{no\text{-}sig}_K})$ and a fixed polynomial $p(\cdot)$ such that for infinitely many $\lambda \in \mathbb{N}$, there exists $\delta^* \in [0,1], \Delta^* \in [0,1], k^* \in K, n^* \in \mathsf{poly}(\lambda)$, $x^* \in \mathcal{X}^n$ such that

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_{k^*}}, \mathbf{x}^*) \geq \epsilon(\lambda, n^*, \delta^*, \Delta^*, K) + \frac{1}{\mathsf{poly}(\lambda)}$$

Given this, we can construct a new prover $\mathsf{P}^*_{\mathsf{no\text{-}com}} = (\mathsf{P}^*_{\mathsf{no\text{-}com}\,1}, \ldots, \mathsf{P}^*_{\mathsf{no\text{-}com}\,K})$ which will contradict the ϵ upper bound for the advantage of $\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}$.

$\mathsf{P}^*_{\mathsf{no\text{-}com}_{k=k^*}}$

1 : Receive $\mathbf{s'}^{k=k^*}$.
2 : For all $k \in [K], k \neq k^*$, set $\mathbf{s'}^k := \mathbf{0}^{n\lambda}$, where $\mathbf{0}$ is a default element.
3 :
4 : For all $k \in [K]$, send $\mathbf{s'}^k$ to $\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}$.
5 : For all $k \in [K]$, receive $\mathbf{z'}^k$ from $\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}$.
6 : Output $\mathbf{z'}^{k^*}$.

$\mathsf{P}^*_{\mathsf{no\text{-}com}_{k\neq k*}}$

1 : Receive $\mathbf{s'}^k$.
2 : Output $\perp$.

From the above construction, it follows that:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}com}\,k^*}, \mathbf{x}^*) = \Pr\left[\begin{array}{c} \exists i \in [n], \mathsf{RW}(b^{\mathrm{inv}\,k^*}_{i,1}||\ldots||b^{\mathrm{inv}\,k^*}_{i,\lambda}) > \delta + \Delta \\ \bigwedge \\ \mathsf{RW}(\mathbf{b}^{k^*}_T) = 0 \end{array} : \mathsf{Exp'}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})\right] \tag{3}$$

, where the experiment $\mathsf{Exp'}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})$ is defined as follows (the difference from $\mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})$ have been highlighted in blue):

Experiment $\mathsf{Exp'}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})$

1 : $\forall i \in [n], j \in [\lambda], \{s_{i,j,k}\}_{k\in[K]} \leftarrow \mathsf{RSR.Encode}^j(x_i)$
2 : $\forall k \in [K]$:
3 : $\mathbf{s}^k := \begin{cases} s_{1,1,k}, \ldots, s_{1,\lambda,k}, \ldots, s_{n,1,k}, \ldots, s_{n,\lambda,k} & ; k = k^* \\ \mathbf{0}^{n\lambda} & ; \text{otherwise} \end{cases}$
4 : $\pi^k \leftarrow$ random permutation on $[n\lambda]$
5 : $\mathbf{s'}^k := \pi^k(\mathbf{s}^k)$
6 : $\mathbf{z'}^k \leftarrow \mathsf{P}^*_{\mathsf{no\text{-}sig}_k}(\mathbf{s'}^k)$
7 : $T^k \leftarrow$ random λ sized subset of $[n] \times [\lambda]$
8 : $\forall i \in [n], j \in [\lambda], b^k_{i,j} = \begin{cases} 0 & ; z'^k_{i,j} = f(s'^k_{i,j}) \\ 1 & ; \text{otherwise} \end{cases}$
9 : $\mathbf{b}^k := b^k_{1,1}, \ldots, b^k_{1,\lambda}, \ldots, b^k_{n,1}, \ldots, b^k_{n,\lambda}$
10 : $\mathbf{b}^{\mathrm{inv}\,k} := (\pi^k)^{-1}(\mathbf{b}^k)$
11 : Parse $\mathbf{b}^{\mathrm{inv}\,k}$ as $b^{\mathrm{inv}\,k}_{1,1}, \ldots, b^{\mathrm{inv}\,k}_{1,\lambda}, \ldots, b^{\mathrm{inv}\,k}_{n,1}, \ldots, b^{\mathrm{inv}\,k}_{n,\lambda}$

Let p indicate the R.H.S probability value in Equation 3. By the no-signaling property established in Definition 3, there exists $\mathsf{negl}(\cdot)$ such that:

$$p \geq \mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_{k=k^*}}, \mathbf{x}^*) - \mathsf{negl}(\lambda)$$

Since we have assumed (towards contradiction) that $\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_{k=k^*}}, \mathbf{x}^*) \geq \epsilon(\lambda, n^*, \delta^*, \Delta^*, K) + \frac{1}{\mathsf{poly}(\lambda)}$, it follows that:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}com}\,k^*}, \mathbf{x}^*) = p \geq \epsilon(\lambda, n^*, \delta^*, \Delta^*, K) + \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)$$

This directly contradicts the fact that for any arbitrary non-communicating multi-prover $\mathsf{P}^*_{\mathsf{no\text{-}com}} = (\mathsf{P}^*_1, \ldots, \mathsf{P}^*_K)$, for all $\delta \in [0,1], \Delta \in [0,1], k \in K$, $n = \mathsf{poly}(\lambda)$, $x \in \mathcal{X}^n, \lambda \in \mathbb{N}$, it holds that $\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}com}\,k}, \mathbf{x}) \leq \epsilon(\lambda, n, \delta, \Delta, K)$.

We will now define the advantage of the overall prover system $\mathsf{P}^*_{\mathsf{no\text{-}sig}} = (\mathsf{P}^*_{\mathsf{no\text{-}sig}_1}, \ldots, \mathsf{P}^*_{\mathsf{no\text{-}sig}_K})$ as follows:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) = \Pr\left[\begin{array}{c} \exists i \in [n], \mathsf{RW}(\Big|\Big|_{j\in[\lambda],k\in[K]} b^{\mathrm{inv}^k_{i,j}}) > (\delta + \Delta) \\ \bigwedge \\ \mathsf{RW}(\mathbf{b}^1_{T^1} || \ldots || \mathbf{b}^1_{T^K}) = 0 \end{array} : \mathsf{Exp}^{K\text{-}\mathsf{RSR}}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x})\right] \tag{4}$$

Claim 3. *Fix $|T^1| = \ldots = |T^K| = \lambda$ and let K be some fixed constant. Then, for all $0 < \delta < 1$, for $n \in O(2^{\lambda^\delta})$, for all $\mathbf{x} \in \mathcal{X}^n$ and for all arbitrary unbounded no-signaling provers $\mathsf{P}^*_{\mathsf{no\text{-}sig}}$,*

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta=0.25/K,\Delta=0.25/K}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) = \mathsf{negl}(\lambda)$$

Proof. From Lemma 4 and Lemma 5, we know that:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}, \mathbf{x}) \leq \epsilon(\lambda, n, \delta, \Delta, K) + \mathsf{negl}(\lambda)$$

where $\epsilon(\lambda, n, \delta, \Delta, K) = 2 \cdot e^{-2\delta^2|T^k|} + n \cdot 2 \cdot e^{-2\Delta^2\lambda}$.
By union bound, we note that $\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) \leq \Sigma_{k\in K}\mathsf{Adv}^{(1,K)\text{-}\mathrm{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}_k}, \mathbf{x})$.
Assuming $|T^1| = \ldots = |T^K| = |T|$, we get that:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta,\Delta}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) \leq 2K \cdot e^{-2\delta^2|T|} + n \cdot 2K \cdot e^{-2\Delta^2\lambda} + K \cdot \mathsf{negl}(\lambda)$$

By setting $\delta = 0.25/K$, $\Delta = 0.25/K$, we get:

$$\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta=0.25/K,\Delta=0.25/K}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) \leq \frac{2K}{2^{0.18|T|/K^2}} + \frac{2nK}{2^{0.18\lambda/K^2}} + K \cdot \mathsf{negl}(\lambda)$$

For constant K, $n \leq 2^{\frac{0.17\lambda}{K^2}}$ and $|T| = \lambda$, we get,

$$\begin{aligned}\mathsf{Adv}^{K\text{-}\mathsf{RSR}}_{\delta=0.25/K,\Delta=0.25/K}(\mathsf{P}^*_{\mathsf{no\text{-}sig}}, \mathbf{x}) &\leq \frac{2K}{2^{\frac{0.18}{K^2}\lambda}} + \frac{2nK}{2^{\frac{0.18}{K^2}\lambda}} + K \cdot \mathsf{negl}(\lambda) \\ &= \mathsf{negl}(\lambda)\end{aligned}$$

Remark 2. Claim 3 shows that one of the following two events will happen (except with some negligible probability): 1) For all $i \in [n]$, the relative hamming weight of the string $\left|\left|_{j\in[\lambda],k\in[K]} b^{\mathrm{inv}^k}_{i,j}\right.\right.$ is less than $0.5/K$ or 2) the relative hamming weight of the substring $\mathbf{b}^1_{T^1}||\ldots||\mathbf{b}^1_{T^K}$ is non-zero. In Case 1, this implies that for all $i \in [n]$, for more than 50% of the j values, all $\{z^k_{i,j}\}_{k\in[K]}$ are correct. This ensures that for all $i \in [n]$, more than 50% of $\{u_{i,j}\}_{j\in[\lambda]}$ will equal to $f(x_i)$. If this happens, for all $i \in [n]$, u^{final}_i will be equal to $f(x_i)$ due to the majority rule. In Case 2, the verifier will simply detect and abort as prescribed in the protocol. This concludes our soundness proof.

5.2 OBVC with a Single Prover

We will now provide a OBVC protocol for all the class of all K-RSR functions which is sound against a single non-uniform PPT prover. The protocol construction is almost identical to the Protocol 5.1 except for the following modification: The verifier samples a fresh HE key pair for each RSR instance and encrypts it before sending it to the prover. The prover is supposed to respond with HE encrypted values obtained by performing a homomorphic evaluation of the circuit C_f on the ciphertexts sent by the verifier. We describe the protocol formally in Figure 5.2.

Theorem 10. *Let $\mathcal{F}^{K\text{-}RSR}_\ell$ denote the class of all ℓ bit functions that admit K-RSR for any $K \geq 1$. Assuming a homomorphic encryption scheme for $\mathcal{F}^{K\text{-}RSR}_\ell$, there exists a OBVC scheme, specifically Protocol 5.2, for $\mathcal{F}^{K\text{-}RSR}_\ell$ with soundness against arbitrary non-uniform PPT provers.*

Corollary 11. *For all $\lambda = \omega(\log n)$, Protocol 5.2 is an OBVC scheme for $\mathcal{F}^{K\text{-}RSR}_\ell$ with soundness error $\mathsf{negl}(n)$ against non-uniform PPT provers.*

In the rest of this section, we will prove Theorem 10. We note that the completeness of Protocol 5.2 follows directly from the correctness property of RSR and $\mathcal{F}$-homomorphism property of the HE scheme. We now proceed to discuss non-triviality, privacy, efficiency and prove soundness.

Non-triviality, Privacy and Efficiency Analysis. In our protocol, the verifier uses two helper oracles namely $\mathcal{O}_{\mathsf{RSR.Encode}_f}$ and $\mathcal{O}_{\mathsf{RSR.Decode}_f}$. By Definition 2, we know that $T_{\mathsf{RSR}}.\mathsf{Encode}(\ell) + T_{\mathsf{RSR}}.\mathsf{Decode}(\ell) = o(T_f(\ell))$. Hence, our protocol satisfies the non-triviality condition.

The privacy of our protocol follows directly from the CPA-security of the underlying HE scheme. More formally, the simulator $\mathsf{Sim}(1^\lambda, 1^n, \mathcal{X})$ simply runs

the verifier V on inputs $x_1 = \ldots = x_n = \mathbf{0}$ where $\mathbf{0}$ is a default element in the domain of f. By the CPA-security of HE scheme and a standard hybrid argument, the view of the server in the real protocol will be computationally indistinguishable from the simulated view.

For efficiency, note that each helper oracle is invoked exactly $n\lambda$ times and the $\mathcal{O}_f$ is invoked exactly $K\lambda$ times. For security parameter λ, let $T_{\mathsf{HE.Keygen}}(\lambda)$, $T_{\mathsf{HE.Enc}}(\lambda)$ and $T_{\mathsf{HE.Dec}}(\lambda)$ denote the time-complexity of $\mathsf{HE.Keygen}$, $\mathsf{HE.Enc}$ and $\mathsf{HE.Dec}$ respectively. Then the running time of V is exactly $O(nK\lambda(T_{\mathsf{HE.Keygen}}(\lambda)+\ell \cdot T_{\mathsf{HE.Enc}}(\lambda) + \ell \cdot T_{\mathsf{HE.Dec}}(\lambda)))$ as the bottleneck cost comes from generating HE keys for each of the $nK\lambda$ shares i.e. $\{s_{i,j,k}\}_{i\in[n],j\in[\lambda],k\in[K]}$ and then encrypting and decrypting them. The other steps like shuffling, majority and cut-and-chose check can be computed in linear time. Here K is a constant which depends on the function f (but independent of n, λ and ℓ).

Protocol 5.2

Common input: 1^λ, 1^n, f

V**'s additional input**: Inputs $x_1, \ldots, x_n$, oracle $\mathcal{O}_f$, helper oracles $\mathcal{O}_{\mathsf{RSR.Encode}_f}$, $\mathcal{O}_{\mathsf{RSR.Decode}_f}$.

P**'s additional input**: Circuit C_f for computing f.

1. For each x_i, V generates λ independent RSR instances Formally, $\forall i \in [n], j \in [\lambda]$: $\{s_{i,j,k}\}_{k\in[K]}, \mathsf{st}_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Encode}_f}(x_i)$.
2. $\forall i \in [n], j \in [\lambda], k \in [K]$, V generates $\mathsf{pk}_{i,j,k}, \mathsf{sk}_{i,j,k} \leftarrow \mathsf{HE.Keygen}(1^\lambda)$.
3. $\forall i \in [n], j \in [\lambda], k \in [K]$, V computes $\mathsf{ct}_{i,j,k} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}_{i,j,k}}(1^\lambda, s_{i,j,k})$.
4. For all $k \in [K]$, it sets $\mathbf{s}^k := (\mathsf{ct}_{1,1,k}, \mathsf{pk}_{1,1,k}), \ldots, (\mathsf{ct}_{1,\lambda,k}, \mathsf{pk}_{1,\lambda,k}), \ldots, (\mathsf{ct}_{n,1,k}, \mathsf{pk}_{n,1,k}), \ldots, (\mathsf{ct}_{n,\lambda,k}, \mathsf{pk}_{n,\lambda,k})$.
5. $\forall k \in [K]$, V samples a random permutation π_k on $[n\lambda]$ and sets $\mathbf{s'}^k := \pi_k(\mathbf{s}^k)$.
6. V sends $\mathbf{s'}^1, \ldots, \mathbf{s'}^K$ to P.
7. $\forall k \in [K]$, P parses $s'^k_{i,j}$ as $(\mathsf{ct}_*, \mathsf{pk}_*)$ and computes $\mathsf{ct}'_{i,j,k} := \mathsf{HE.Eval}_{\mathsf{pk}_*}(C_f, \mathsf{ct}_*)$.
8. $\forall k \in [K]$, P sets $\mathbf{z'}^k := \mathsf{ct}'_{1,1,k}, \ldots, \mathsf{ct}'_{1,\lambda,k}, \ldots, \mathsf{ct}'_{n,1,k}, \ldots, \mathsf{ct}'_{n,\lambda,k}$.
9. P sends $\mathbf{z'}^1, \ldots, \mathbf{z'}^K$ to V.
10. $\forall k \in [K]$, V samples a random subset $T^k \subset [n] \times [\lambda]$ of size λ and checks whether the following holds:

$$\forall (i,j) \in T^k : \mathsf{HE.Dec}_{\mathsf{sk}_{i',j',k}}(\mathbf{z'}^k_{i,j}) = f(s^k_{i',j'})$$

    where $(i', j') := \pi_k^{-1}(i, j)$.
11. If the check fails, then V outputs $\perp$. Otherwise it proceeds.
12. $\forall k \in [K]$, V computes $\mathbf{z}^k := \pi_k^{-1}(\mathbf{z'}^k)$.
13. $\forall i \in [n], j \in [\lambda]$, V computes $u_{i,j} \leftarrow \mathcal{O}_{\mathsf{RSR.Decode}_f}(\{w_{i,j,k}\}_{k\in[K]}, \mathsf{st}_{i,j})$, where $w_{i,j,k} = \mathsf{Dec}_{sk_{i,j,k}}(z^k_{i,j})$
14. $\forall i \in [n]$, V sets $u_i^{\mathsf{final}} = \mathsf{Majority}(u_{i,1}, \ldots, u_{i,\lambda})$
15. V outputs $u_1^{\mathsf{final}}, \ldots, u_n^{\mathsf{final}}$

Soundness Analysis. Now we will show how the security of Protocol 5.1 against arbitrary no-signaling multi-prover $\mathsf{P}_{\mathsf{no\text{-}sig}}$ can be carried over to the security of Protocol 5.2 against arbitrary non-uniform PPT prover P. As mentioned earlier, the main ingredient used in Protocol 5.2 is an HE scheme. The main idea behind the security proof amounts to showing that any malicious PPT prover in Protocol 5.2 will conform to the notion of no-signaling prover as defined in Definition 3. The formal proof follows via reduction to the CPA-security of the HE scheme. We refer the readers to the full version.

6 Impossibility of oracle-aided batch verifiable computation

Definition 6. *A* $(s(\lambda), t(\lambda), q(\lambda), n(\lambda))$ OBVC *scheme* $\Pi = (\mathsf{P}, \mathsf{V})$ *in the* $\mathcal{O}$ *model is defined as follows.*

- *The verifier* V *which is a two-staged entity i.e.* $\mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2)$. V_1 *is computationally unbounded; it interacts with* $\mathcal{O}.\mathsf{pre}$ *and outputs an* s*-bit "advice" string.* V_2 *is computationally bounded and also query bounded. It takes an* s*-bit auxiliary input and makes at most* t *queries to* $\mathcal{O}.\mathsf{main}$.
- *The prover* P *which is a single staged entity and makes at most* q *queries to* $\mathcal{O}.\mathsf{main}$. *There is no computational bound on the prover.*

We will use the notation $\langle \mathsf{P}^{\mathcal{O}}, \mathsf{V}^{\mathcal{O}} \rangle_{\Pi}$ *to denote the following protocol interaction:*

- V_1 *interacts with* $\mathcal{O}.\mathsf{pre}$ *and outputs an* s*-bit "advice" string.*
- V_1 *passes a* s*-bit auxiliary input* aux *to* V_2.
- *Sample a batch of instances* $I \subseteq [M]$ *where* $|I| = n$. *Send* I *to* V_2.
- P *and* V_2 *interact with each other while having access to* $\mathcal{O}.\mathsf{main}$.
- V_2 *returns* OUT *in the end.*

The scheme Π *satisfies the following properties.*

- *Completeness: For all* $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{OUT} = \mathcal{O}(x_1^I), \ldots, \mathcal{O}(x_n^I)] = 1$$

- *Soundness: For all adversarial* P^*, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t. for all* $\lambda \in \mathbb{N}$*:*

$$\Pr[\mathsf{OUT} = \mathcal{O}(x_1^I), \ldots, \mathcal{O}(x_n^I) \vee \mathsf{OUT} = \perp] = 1 - \mathsf{negl}(\lambda)$$

- *Efficiency: We say that an* OBVC *scheme is efficient if the* $s(\lambda) \in \mathsf{poly}(\lambda)$ *and* $t(\lambda) \in o(n(\lambda))$.

Theorem 12. *For all* $n \in \mathsf{poly}(\lambda)$, $\alpha' \in (0, 1]$, $t \in o(n)$, $q = q(\lambda)$, $s \in \mathsf{poly}(\lambda)$, *for every* (s, t, q, n) OBVC *scheme* $\Pi = (\mathsf{P}, \mathsf{V})$ *in the* $\mathcal{O} := \mathsf{BF\text{-}RO}(M = 2^{\lambda}, N = 2^{\lambda}, p = 2^{(1-\alpha')\lambda})$ *model, there exists a malicious prover* P_{mal} *and noticeable function* $\epsilon'(\lambda)$ *s.t. for all* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathsf{OUT} \neq \mathcal{O}(x_1^I), \ldots, \mathcal{O}(x_n^I) \wedge \mathsf{OUT} \neq \perp : \mathsf{OUT} \leftarrow \langle \mathsf{P}_{mal}^{\mathcal{O}}, \mathsf{V}^{\mathcal{O}} \rangle_{\Pi}\right] \geq \epsilon'(\lambda)$$

Proof. Refer to the full version.

We will now lift the above theorem from the Bit-fixing RO model to Auxiliary-input RO model.

Theorem 13. *For all* $n \in \mathsf{poly}(\lambda)$, $\alpha \in (0,1]$, $q = 2^{(1-\alpha)\lambda}$, $t \in o(n)$, $s \in \mathsf{poly}(\lambda)$, *for every* (s,t,q,n) OBVC *scheme* $\Pi = (\mathsf{P},\mathsf{V})$ *in the* $\mathcal{O} := \mathsf{AI\text{-}RO}(M = 2^\lambda, N = 2^\lambda)$, *there exists a malicious prover* P_{mal} *and noticeable function* $\epsilon(\lambda)$ *s.t. for all* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathsf{OUT} \neq \mathcal{O}(x_1^I), \ldots, \mathcal{O}(x_n^I) \wedge \mathsf{OUT} \neq \perp : \mathsf{OUT} \leftarrow \langle \mathsf{P}_{mal}^{\mathcal{O}}, \mathsf{V}^{\mathcal{O}} \rangle_{\Pi}\right] \geq \epsilon(\lambda)$$

Proof. The formal proof leverages Theorem 4. We refer the readers to the full version.

Acknowledgments

A. Agarwal and D. Khurana were supported in part by NSF CAREER CNS-2238718, DARPA SIEVE and an award from Visa Research. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024.

Disclaimer

Case studies, comparisons, statistics, research and recommendations are provided "AS IS" and intended for informational purposes only and should not be relied upon for operational, marketing, legal, technical, tax, financial or other advice. Visa Inc. neither makes any warranty or representation as to the completeness or accuracy of the information within this document, nor assumes any liability or responsibility that may result from reliance on such information. The Information contained herein is not intended as investment or legal advice, and readers are encouraged to seek the advice of a competent professional where such advice is required.

These materials and best practice recommendations are provided for informational purposes only and should not be relied upon for marketing, legal, regulatory or other advice. Recommended marketing materials should be independently evaluated in light of your specific business needs and any applicable laws and regulations. Visa is not responsible for your use of the marketing materials, best practice recommendations, or other information, including errors of any kind, contained in this document.

References

1. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 522–539. SIAM (2021)

2. Applebaum, B., Ishai, Y., Kushilevitz, E.: From secrecy to soundness: Efficient verification via secure computation. In: International Colloquium on Automata, Languages, and Programming. pp. 152–163. Springer (2010)
3. Ar, S., Blum, M., Codenotti, B., Gemmell, P.: Checking approximate computations over the reals. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing. pp. 786–795 (1993)
4. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Locally random reductions: Improvements and applications. Journal of Cryptology **10**(1), 17–36 (1997)
5. Bellare, M., Garay, J.A., Rabin, T.: Batch verification with applications to cryptography and checking. In: Latin American Symposium on Theoretical Informatics. pp. 170–191. Springer (1998)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)
7. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 73–83 (1990)
8. Blum, M., Luby, M., Rubinfeld, R.: Program result checking against adaptive programs. In: Distributed Computing and Cryptography: Proceedings of a DIMACS Workshop, October 4-6, 1989. vol. 2, p. 107. American Mathematical Soc. (1991)
9. Brakerski, Z., Holmgren, J., Kalai, Y.: Non-interactive ram and batch np delegation from any pir. Cryptology ePrint Archive (2016)
10. Brakerski, Z., Holmgren, J., Kalai, Y.: Non-interactive delegation and batch np verification from standard computational assumptions. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 474–482 (2017)
11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on computing **43**(2), 831–871 (2014)
12. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part IV. Lecture Notes in Computer Science, vol. 12828, pp. 394–423. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84259-8_14
13. Choudhuri, A.R., Jain, A., Jin, Z.: Snargs for $\mathcal{P}$ from LWE. In: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022. pp. 68–79. IEEE (2021). https://doi.org/10.1109/FOCS52979.2021.00016, https://doi.org/10.1109/FOCS52979.2021.00016
14. Chung, K.M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Annual Cryptology Conference. pp. 483–501. Springer (2010)
15. Cleve, R., Luby, M.: A note on self-testing/correcting methods for trigonometric functions. International Computer Science Inst. (1990)
16. Coretti, S., Dodis, Y., Guo, S., Steinberger, J.: Random oracles and non-uniformity. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I. pp. 227–258. Springer (2018)
17. Dodis, Y., Guo, S., Katz, J.: Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In: Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part II. pp. 473–495. Springer (2017)

18. Gemmell, P., Lipton, R., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: STOC. vol. 91, pp. 32–42. Citeseer (1991)
19. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Annual Cryptology Conference. pp. 465–482. Springer (2010)
20. Hoeffding, W.: Probability inequalities for sums of bounded random variables. The collected works of Wassily Hoeffding pp. 409–426 (1994)
21. Hulett, J., Jawale, R., Khurana, D., Srinivasan, A.: SNARGs for P from sub-exponential DDH and QR. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part II. Lecture Notes in Computer Science, vol. 13276, pp. 520–549. Springer, Heidelberg, Germany, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-07085-3_18
22. Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.Y.: Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In: Khuller, S., Williams, V.V. (eds.) STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021. pp. 708–721. ACM (2021). https://doi.org/10.1145/3406325.3451055, https://doi.org/10.1145/3406325.3451055
23. Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: Proceedings of the forty-sixth annual ACM symposium on Theory of computing. pp. 485–494 (2014)
24. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)
25. Lipton, R.: New directions in testing. Distributed computing and cryptography **2**, 191–202 (1991)
26. Micali, S.: Computationally sound proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000)
27. Rubinfeld, R.: Batch checking with applications to linear functions. Information Processing Letters **42**(2), 77–80 (1992)
28. Unruh, D.: Random oracles and auxiliary input. In: Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27. pp. 205–223. Springer (2007)
29. Waters, B., Wu, D.J.: Batch arguments for np and more from standard bilinear group assumptions. In: Annual International Cryptology Conference. pp. 433–463. Springer (2022)