Efficient Rational Proofs for Space Bounded Computations

Matteo Campanelli(✉) and Rosario Gennaro

The City University of New York, New York, USA
mcampanelli@gradcenter.cuny.edu, rosario@ccny.cuny.edu

Abstract. We present new protocols for the verification of *space bounded polytime computations* against a rational adversary. For such computations requiring sublinear space our protocol requires only a verifier running in sublinear-time. We extend our main result in several directions: (i) we present protocols for randomized complexity classes, using a new *composition theorem* for rational proofs which is of independent interest; (ii) we present lower bounds (i.e. conditional impossibility results) for Rational Proofs for various complexity classes.

Our new protocol is the first rational proof not based on the circuit model of computation, and the first *sequentially composable* protocols for a well-defined language class.

1 Introduction

Consider the problem of Outsourced Computation where a computationally "weak" client hires a more "powerful" server to store data and perform computations on its behalf. This paper is concerned with the problem of designing outsourced computation schemes that incentivize the server to perform correctly the tasks assigned by the client.

The rise of the *cloud computing* paradigm where business do not maintain their own IT infrastructure, but rather hire "providers" to run it, has brought this problem to the forefront of the research community. The goal is to find solutions that are efficient and feasible in practice for problems such as: How do we check the integrity of data that is stored remotely? How do we check computations performed on this remotely stored data? How can a client do this in the most efficient way possible?

For all the scenarios above, what mechanisms can be designed to incentivize parties to perform correctly no matter what the cost of the correct behavior might be?

1.1 Complexity Theory and Cryptography

The problem of efficiently checking the correctness of a computation performed by an untrusted party has been central in Complexity Theory for the last 30 years since the introduction of Interactive Proofs by Babai and Goldwasser, Micali and Rackoff [5,14].

S. Rass et al. (Eds.): GameSec 2017, LNCS 10575, pp. 53–73, 2017.
DOI: 10.1007/978-3-319-68711-7_4

Verifiable Outsourced Computation is now a very active research area in Cryptography and Network Security (see [27] for a survey) with the aim to design protocols where it is impossible (under suitable cryptographic assumptions) for a provider to "cheat" in the above scenarios. While much progress has been done in this area, we are still far from solutions that can be deployed in practice. Part of the reason is that Cryptographers consider a very strong adversarial model that prevents any adversary from cheating. A different approach is to restrict ourselves to *rational adversaries*, whose motivation is not just to disrupt the protocol or computation, but simply to maximize a well defined utility function (e.g. profit).

1.2 Rational Proofs

In our work we use the concept of Rational Proofs introduced by Azar and Micali in [3] and refined in a subsequent paper [4].

In a Rational Proof, given a function f and an input x, the server returns the value $y = f(x)$, and (possibly) some auxiliary information, to the client. The client will in turn pay the server for its work with a reward which is a function of the messages sent by the server and some randomness chosen by the client. The crucial property is that this reward is maximized in expectation when the server returns the correct value y. Clearly a rational prover who is only interested in maximizing his reward, will always answer correctly.

The most striking feature of Rational Proofs is their simplicity. For example in [3], Azar and Micali show single-message Rational Proofs for any problem in $\#P$, where an (exponential-time) prover convinces a (poly-time) verifier of the number of satisfying assignment of a Boolean formula.

For the case of "real-life" computations, Azar and Micali in [4] consider the case of efficient provers (i.e. poly-time) and "super-efficient" (log-time) verifiers and present d-round Rational Proofs for functions computed by (uniform) Boolean circuits of depth d, for $d = O(\log n)$.

Recent work [16] shows how to obtain Rational Proofs with sublinear verifiers for languages in NC. Recalling that $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}_2$, one can use the protocol in [16] to verify a logspace polytime computation (deterministic or nondeterministic) in $O(\log^2 n)$ rounds and $O(\log^2 n)$ verification.

The work by Chen et al. [9] focuses on rational proofs with multiple provers and the related class MRIP of languages decidable by a polynomial verifier interacting with an arbitrary number of provers. Under standard complexity assumptions, MRIP includes languages not decidable by a verifier interacting only with one prover. The class MRIP is equivalent to $\mathsf{EXP}^{||\mathsf{NP}}$.

1.3 Repeated Executions with a Budget

In [8] we present a critique of the rational proof model in the case of "repeated executions with a budget". This model arises in the context of "volunteer computations" [1,22] where many computational tasks are outsourced and provers compete in solving as many as possible to obtain rewards. In this scenario assume that a prover has a certain budget B of "computational effort": how can one guarantee that the rational strategy is to provide the correct answer in *all* the proof he provides? The notion of rational proof guarantees that if the prover engages in a single rational proof then it is in his best interest to provide the correct output. But in [8] we show that in the presence of many computations, it might be more profitable for the prover to use his budget B to provide many incorrect answers than to provide a single correct answer. That's because incorrect (e.g. random) answers are "cheaper" to compute than the correct one and with the same budget B the prover can provide many of them while the entire budget might be necessary to solve a single problem correctly. If the difference in reward between correct and incorrect answers is not high enough then many incorrect answers may be more profitable and a rational prover will choose that strategy, and indeed this is the case for many of the protocols in [3,4,15,16].

In [8] we put forward a stronger notion of *sequentially composable rational proofs* which avoids the above problem and guarantees that the rational strategy is always the one to provide correct answers. We also presented sequentially composable rational proofs, but only for some ad-hoc cases, and were not able to generalize them to well-defined complexity classes.

1.4 Our Contribution

This paper presents new protocols for the verification of *space-bounded polytime computations* against a rational adversary. More specifically, let L be a language in the class $\mathsf{DTISP}(T(n), S(n))$, i.e. L is recognized by a deterministic Turing Machine M_L which runs in time $T(n)$ and space $S(n)$. We construct a protocol where a rational prover can convince the verifier that $x \in L$ or $x \notin L$ with the following properties:

- The verifier runs in time $O(S(n) \log n)$
- The protocol has $O(\log n)$ rounds and communication complexity $O(S(n) \log n)$
- The prover simply runs $M_L(x)$

Under suitable assumptions, our protocol can be proven to correctly incentivize a prover in **both** the stand-alone model of [3] and the sequentially composable definition of [8]. This is the first protocol which is sequentially composable for a well-defined complexity class.

For the case of "real-life" computations (i.e. poly-time computations verified by a "super-efficient" verifier) we note that for computations in sublinear space our general results yields a protocol in which the verifier is sublinear-time. More specifically, we introduce the first rational proof for SC (also known as $\mathsf{DTISP}(\mathsf{poly}(n), \mathsf{polylog}(n))$) with polylogarithmic verification and logarithmic rounds.

To compare this with the results in [16], we note that it is believed that $\mathsf{NC} \neq \mathsf{SC}$ and that the two classes are actually incomparable (see [10] for a discussion). For these computations our results compare favorably to the one in [16] in at least one aspect: our protocol requires $O(\log n)$ rounds and has the same verification complexity[1].

We present several extensions of our main result:

- Our main protocol can be extended to the case of space-bounded randomized computations using Nisan's pseudo-random generator [24] to derandomize the computation.
- We also present a different protocol that works for BPNC (bounded error randomized NC) where the Verifier runs in polylog time (note that this class is not covered by our main result since we do not know how to express NC with a polylog-space computation). This protocol uses in a crucial way a new *composition theorem* for rational proofs which we present in this paper and can be of independent interest.
- Finally we present lower bounds (i.e. conditional impossibility results) for Rational Proofs for various complexity classes.

1.5 The Landscape of Rational Proof Systems

Rational Proof systems can be divided in roughly two categories, both of them presented in the original work [3].

Scoring Rules. The work in [3] uses *scoring rules* to compute the reward paid by the verifier to the prover. A scoring rule is used to asses the "quality" of a prediction of a randomized process. Assume that the prover declares that a certain random variable X follows a particular probability distribution D. The verifier runs an "experiment" (i.e. samples the random variable in question) and computes a "reward" based on the distribution D announced by the prover and the result of the experiment. A scoring rule is maximized if the prover announced the real distribution followed by X. The novel aspect of many of the protocols in [3] was how to cast the computation of $y = f(x)$ as the announcement of a certain distribution D that could be tested efficiently by the verifier and rewarded by a scoring rule.

A simple example is the protocol for $\#P$ in [3] (or its "scaled-down" version for Hamming weight described more in detail in Sect. 2.1). Given a Boolean formula $\varPhi(x_1, \ldots, x_n)$ the prover announces the number m of satisfying assignments. This can be interpreted as the prover announcing that if one chooses an assignment at random it will be a satisfying one with probability $m \cdot 2^{-n}$. The verifier then chooses a random assignment and checks if it satisfies $\varPhi$ or not and uses m and the result of the test to compute the reward via a scoring rule. Since the scoring rule is maximized by the announcement of the correct m, a rational prover will announce the correct value.

[1] We also point out that in [16] a rational protocol for P, polytime computations, is presented, but for the case of a computationally bounded prover, i.e. a *rational argument*.

As pointed out in [8] the problem with the scoring rule approach is that the reward declines slowly as the distribution announced by the Prover becomes more and more distant from the real one. The consequence is that incorrect results still get a substantial reward, even if not a maximal one. Since those incorrect results can be computed faster than the correct one, a Prover with "budget" B might be incentivized to produce many incorrect answers instead of a single correct one. All of the scoring rule based protocols in [3,4,15,16] suffer from this problem.

Weak Interactive Proofs. In the definition of rational proofs we require that the expected reward is maximized for the honest prover. This definition can be made stronger (as done explicitly in [15]) requiring that every *systematically dishonest* prover would incur a polynomial loss (this property is usually described in terms of a *noticeable reward gap*). Obviously we can use classical interactive proofs to trivially obtain this property. In fact, recall standard interactive proofs: at the end of the interaction with a prover, the verifier applies a "decision function" D to a transcript in order to *accept* or *reject* the input x. A verifier may then pay the prover a reward $R = \mathsf{poly}(|x|)$ iff D accepts. The honest prover will clearly maximize its reward since, by definition of interactive proof, the probability of a wrong acceptance/rejection is negligible. Notice hoverer that we can obtain rational proofs with noticeble reward gap even if the protocol has a much higher error probability. In fact, for an appropriate choice of a (polynomial) reward R, the error probability can be as high as $1 - n^{-k}$ for some $k \in \mathbb{N}$. We call an interactive proof with such a high error probability a *weak interactive proof*[2].

Weak interactive proofs can be turned into strong (i.e. with negligible error) classical ones by repetition, which however increases the computational cost of the verifier. But since to obtain a rational proof it is not necessary to repeat them, we can use them to obtain rational proofs which are very efficient for the verifier. Indeed, some of the protocols in [3,8] are rational proofs based on weak interactive proofs. This approach is also the main focus in the present work.

Discussion. There are two intriguing questions when we compare the "scoring rules" approach to build rational proofs, to the one based on "weak interactive proofs".

- Is one approach more powerful than the other?
- All the known sequentially composable proofs are weak interactive proofs. Does sequential composition requires a weak interactive proof?

We do not know the answers to the above questions. For a more detailed discussion we refer the reader to the end of Sect. 7.

[2] This is basically the *covert adversary* model for multiparty computation introduced in [2].

1.6 Other Related Work

Interactive Proofs. As already discussed, a "traditional" interactive proof (where security holds against any adversary, even a computationally unbounded one) would work in our model. In this case the most relevant result is the recent independent work in [26] that presents breakthrough protocols for the deterministic (and randomized) restriction of the class of language we consider. If L is a language which is recognized by a deterministic (or randomized) Turing Machine M_L which runs in time $T(n)$ and space $S(n)$, then their protocol has the following properties:

- The verifier runs in $O(\mathsf{poly}(S(n)) + n \cdot \mathsf{polylog}(n))$ time;
- The prover runs in polynomial time;
- The protocol runs in *constant* rounds, with communication complexity $O(\mathsf{poly}(S(n)n^{\delta})$ for a constant δ.

Apart from round complexity (which is the impressive breakthrough of the result in [26]) our protocols fares better in all other categories. Note in particular that a sublinear space computation does not necessarily yield a sublinear-time verifier in [26]. On the other hand, we stress that our protocol only considers weaker rational adversaries.

Computational Arguments. There is a large class of protocols for *arguments* of correctness (e.g. [12,13,19]) even in the rational model [15,16]. Recall that in an argument, security is achieved only against computationally bounded prover. In this case even single round solutions can be achieved. We do not consider this model in this paper, except in Sect. 5.2 as one possible option to obtain sequential composability.

Computational Decision Theory. Other works in theoretical computer science have studied the connections between cost of computation and utility in decision problems. The work in [17] proposes a framework for *computational decision problems*, where the Decision Maker's (DM) utility depends on the algorithm chosen for computing its strategy. The Decision Maker runs the algorithm, assumed to be a Turing Machine, on the input to the computational decision problem. The output of the algorithm determines the DM's strategy. Thus the choice of the DM reduces to the choice of a Turing Machine from a certain space. The DM will have beliefs on the running time (cost) of each Turing Machine. The actual cost of running the chosen TM will affect the DM's reward. Rational proofs with costly computation could be formalized in the language of *computational decision problems* in [17]. There are similarities between the approach in this work and that in [17], as both take into account the cost of computation in a decision problem.

2 Rational Proofs

The following is the definition of Rational Proof from [3]. As usual with $\mathsf{neg}(\cdot)$ we denote a *negligible* function, i.e. one that is asymptotically smaller than the inverse of any polynomial. Conversely a *noticeable* function is the inverse of a polynomial.

Definition 1 (Rational Proof). *A function f: $\{0,1\}^n \rightarrow \{0,1\}^*$ admits a rational proof if there exists an interactive proof (P,V) and a randomized reward function* $\mathsf{rew} : \{0,1\}^* \rightarrow \mathbb{R}_{\geq 0}$ *such that*

1. *For any input $x \in \{0,1\}^n$,* $\Pr[\mathsf{out}((P,V)(x)) = f(x)] \geq 1 - \mathsf{neg}(n)$.
2. *For every prover $\widetilde{P}$, and for any input $x \in \{0,1\}^n$ there exists a $\delta_{\widetilde{P}}(x) \geq 0$ such that* $\mathbb{E}[\mathsf{rew}((\widetilde{P},V)(x))] + \delta_{\widetilde{P}}(x) \leq \mathbb{E}[\mathsf{rew}((P,V)(x))]$.

The expectations and the probabilities are taken over the random coins of the prover and verifier.

We note that differently than [3] we allow for non-perfect completeness: a negligible probability that even the correct prover will prove the wrong result. This will be necessary for our protocols for randomized computations.

Let $\epsilon_{\widetilde{P}} = \Pr[\mathsf{out}((P,V)(x)) \neq f(x)]$. Following [15] we define the $\mathsf{reward\ gap}$ as $\Delta(x) = min_{P^*:\epsilon_{P^*}=1}[\delta_{P^*}(x)]$, i.e. the minimum reward gap over the provers that always report the incorrect value. It is easy to see that for arbitrary prover $\widetilde{P}$ we have $\delta_{\widetilde{P}}(x) \geq \epsilon_{\widetilde{P}} \cdot \Delta(x)$. Therefore it suffices to prove that a protocol has a strictly positive reward gap $\Delta(x)$ for all x.

Definition 2 [3,4,15]**.** *The class* $\mathsf{DRMA}[r,c,T]$ *(Decisional Rational Merlin Arthur) is the class of boolean functions $f : \{0,1\}^* \rightarrow \{0,1\}$ admitting a rational proof $\Pi = (P,V,\mathsf{rew})$ s.t. on input x:*

- *Π terminates in $r(|x|)$ rounds;*
- *The communication complexity of P is $c(|x|)$;*
- *The running time of V is $T(|x|)$;*
- *The function* rew *is bounded by a polynomial;*
- *Π has noticeable reward gap.*

Remark 1. The requirement that the reward gap must be noticeable was introduced in [4,15] *and is explained in Sect.* 5*.*

2.1 A Warmup Example

Consider the function $f : \{0,1\}^n \rightarrow [0 \ldots n]$ which on input x outputs the Hamming weight of x (i.e. $\sum_i x_i$ where x_i are the bits of x).

In [4] the prover announces a number $\tilde{m}$ which he claims to be equal to $m = f(x)$. This can be interpreted as the prover announcing that if one chooses an input bit x_i at random it will be equal to 1 with probability $\tilde{p} = \tilde{m}/n$. The verifier then chooses a random input bit x_i and uses $\tilde{m}, x_i$ to compute the reward via a scoring rule. Since the scoring rule is maximized by the announcement of the correct m, a rational prover will announce the correct value. The scoring rule used in [4] (and in all other rational proofs based on scoring rules) is Brier's rule where the reward is computed as $BSR(\tilde{p}, x_i)$ where $BSR(\tilde{p}, 1) = 2\tilde{p}(2 - \tilde{p})$ and $BSR(\tilde{p}, 0) = 2(1 - \tilde{p}^2)$. Notice that $p = m/n$ is the actual probability to get 1 when selecting an input bit at random, so the expected reward of the prover is $pBSR(\tilde{p}, 1)+ (1 - p)BSR(\tilde{p}, 0)$ which is easily seen to be maximized for $\tilde{p} = p$, i.e. $\tilde{m} = m$.

In [8] we propose an alternative protocol for f (motivated by the issues we discuss in Sect. 5). In our protocol we compute f via an "addition circuit", organized as a complete binary tree with n leaves which are the input, and where each internal node is a (fan-in 2) addition gate – note that this circuit has depth $d = \log n$. The protocol has d rounds: at the first round the prover announces $\tilde{m}$ (the claimed value of $f(x)$) and its two "children" y_L, y_R in the output gate, i.e. the two input values of the last output gate G. The Verifier checks that $y_L + y_R = \tilde{m}$, and then asks the Prover to verify that y_L or y_R (chosen a random) is correct, by recursing on the above test. At the end the verifier has to check the last addition gate on two input bits: she performs this test on her own by reading just those two bits. If any of the tests fails, the verifier pays a reward of 0, otherwise she will pay R. The intuition is that a cheating prover will be caught with probability 2^{-d} which is exactly the reward gap (and for log-depth circuits like this one is noticeable). Note that the first protocol is a scoring-rule based one, while the second one is a weak-interactive proof.

3 Rational Proofs for Space-Bounded Computations

We are now ready to present our protocol. It uses the notion of a Turing Machine *configuration*, i.e. the complete description of the current state of the computation: for a machine M, its state, the position of its heads, the non-blank values on its tapes.

Let $L \in \mathsf{DTISP}(T(n), S(n))$ and M be the deterministic TM that recognizes L. On input x, let $\gamma_1, \ldots, \gamma_N$ (where $N = T(|x|)$) be the configurations that M goes through during the computation on input x, where γ_{i+1} is reached from γ_i according to the transition function of M. Note, first of all, that each configuration has size $O(S(n))$. Also if $x \in L$ (resp. $x \notin L$) then γ_N is an accepting (resp. rejecting) configuration.

The protocol presented below is a more general version of the one used in [8] and described above. The prover shows the claimed final configuration $\hat{\gamma}_N$ and then prover and verifier engage in a "chasing game", where the prover "commits" at each step to an intermediate configuration. If the prover is cheating (i.e. $\hat{\gamma}_N$ is wrong) then the intermediate configuration either does not follow from the initial configuration or does not lead to the final claimed configuration. At each step and after P communicates the intermediate configuration γ', the verifier then randomly chooses whether to continue invoking the protocol on the left or the right of γ'. The protocol terminates when V ends up on two previously declared adjacent configurations that he can check. Intuitively, the protocol works since, if $\hat{\gamma}_N$ is wrong, for any possible sequence of the prover's messages, there is at least one choice of random coins that allows V to detect it; the space of such choices is polynomial in size.

We assume that V has oracle access to the input x. What follows is a formal description of the protocol.

1. P sends to V:
   - γ_N, the final accepting configuration (the starting configuration, γ_1, is known to the verifier);
   - N, the number of steps between the two configurations.
2. Then V invokes the procedure $\mathsf{PathCheck}(N, \gamma_1, \gamma_N)$.

The procedure $\mathsf{PathCheck}(m, \gamma_l, \gamma_r)$ is defined for $1 \leq m \leq N$ as follows:

- If $m > 1$, then:
  1. P sends intermediate configurations γ_p and γ_q (which may coincide) where $p = \lfloor \frac{l+m-1}{2} \rfloor$ and $q = \lceil \frac{l+m-1}{2} \rceil$.
  2. If $p \neq q$, V checks whether there is a transition leading from configuration γ_p to configuration γ_q. If yes, V accepts; otherwise V halts and rejects.
  3. V generates a random bit $b \in_R \{0, 1\}$
  4. If $b = 0$ then the protocol continues invoking $\mathsf{PathCheck}(\lfloor \frac{m}{2} \rfloor, \gamma_l, \gamma_p)$; If $b = 1$ the protocol continues invoking $\mathsf{PathCheck}(\lfloor \frac{m}{2} \rfloor, \gamma_q, \gamma_r)$
- If $m = 1$, then V checks whether there is a transition leading from configuration γ_l to configuration γ_r. If $l = 1$, V checks that γ_l is indeed the initial configuration γ_1. If $r = N$, V checks that γ_r is indeed the final configuration sent by P at the beginning. If yes, V accepts; otherwise V rejects.

Theorem 1. $\mathsf{DTISP}[\mathsf{poly}(n), S(n)] \subseteq \mathsf{DRMA}[O(\log n), O(S(n) \log n), O(S(n) \log n)]$

Proof. Let us consider the efficiency of the protocol above. It requires $O(\log n)$ rounds. Since the computation is in $\mathsf{DTISP}[\mathsf{poly}(n), S(n)]$, the configurations P sends to V at each round have size $O(S(n))$. The verifier only needs to read the configurations and, at the last round, check the existence of a transition leading from γ_l to γ_r. Therefore the total running time for V is $O(S(n) \log n)$.

Let us now prove that this is a rational proof with noticeable reward gap. Observe that the protocol has perfect completeness. Let us now prove that the soundness is at most $1 - 2^{-\log N} = 1 - \frac{1}{O(\mathsf{poly}(n))}$. We aim at proving that, if there is no path between the configurations γ_1 and γ_N then V rejects with probability at least $2^{-\log N}$. Assume, for sake of simplicity, that $N = 2^k$ for some k. We will proceed by induction on k. If $k = 1$, P provides the only intermediate configuration γ' between γ_1 and γ_N. At this point V flips a coin and the protocol will terminate after testing whether there exists a transition between γ_1 and γ' or between γ' and γ_N. Since we assume the input is not in the language, there exists at most one of such transitions and V will detect this with probability 1/2.

Now assume $k > 1$. At the first step of the protocol P provides an intermediate configuration γ'. Either there is no path between γ_1 and γ' or there is no path between γ' and γ_N. Say it is the former: the protocol will proceed on the left with probability 1/2 and then V will detect P cheating with probability 2^{-k+1} by induction hypothesis, which concludes the proof.

The theorem above implies the results below.

Corollary 1. $\mathsf{L} \subseteq \mathsf{DRMA}[O(\log n), O(\log^2 n), O(\log^2 n)]$

This improves over the construction of rational proofs for L in [16] due to the better round complexity.

Corollary 2. $\mathsf{SC} \subseteq \mathsf{DRMA}[O(\log n), O(\mathsf{polylog}(n)), O(\mathsf{polylog}(n))]$

No known result was known for SC before.

3.1 Rational Proofs for Randomized Bounded Space Computation

We now describe a variation of the above protocol, for the case of randomized bounded space computations. Let $\mathsf{BPTISP}[t, s]$ denote the class of languages recognized by randomized machines using time t and space s with error bounded by 1/3 on both sides. In other words, $L \in \mathsf{BPTISP}[\mathsf{poly}(n), S(n)]$ if there exists a (deterministic) Turing Machine M such that for any $x \in \{0,1\}^*$ $\Pr_{r \in_R \{0,1\}^{\rho(|x|)}}[M(x,r) = L(x)] \geq \frac{2}{3}$ and that runs in $S(|x|)$ space and polynomial time. Let $\rho(n)$ be the maximum number of random bits used by M for input $x \in \{0,1\}^n$; $\rho(\cdot)$ is clearly polynomial.

We can bring down the 2/3 probability error to $\mathsf{neg}(n)$ by constructing a machine M'. M' would simulate the M on x iterating the simulation $m = \mathsf{poly}(|x|)$ times using fresh random bits at each execution and taking the majority output of $M(x; \cdot)$. The machine M' uses $m\rho(|x|)$ random bits and runs in polynomial time and $S(|x|) + O(\log(n))$ space.

The work in [24] introduces pseudo-random generators (PRG) resistant against space bounded adversaries. An implication of this result is that any randomized Turing Machine M_1 running in time T and space S can be simulated by a randomized Turing Machine M_2 running in time $O(T)$, space $O(S \log(T))$ and using only $O(S \log(T))$ random bits[3] (see in particular Theorem 3 in [24]). Let $L \in \mathsf{BPTISP}[(\mathsf{poly}(n), S(n)]$ and M' defined as above. We denote by $\hat{M}$ the simulation of M' that uses Nisan's result described above.

[3] We point out that the new machine M_2 introduces a small error. For our specific case this error keeps the overall error probability negligible and we can ignore it.

By using the properties of the new machine $\hat{M}$, we can directly construct rational proofs for $\mathsf{BPTISP}(\mathsf{poly}(n), S(n))$. We let the verifier picks a random string r (of length $O(S\log(T))$) and sends it to the prover. They then invoke a rational proof for the computation $\hat{M}(x;r)$.

By the observations above and Theorem 1 we have the following result:

Corollary 3. $\mathsf{BPTISP}[\mathsf{poly}(n), S(n)] \subseteq \mathsf{DRMA}[\log(n), S(n)\log^2(n), S(n)\log^2(n)]$

We note that for this protocol, we need to allow for non-perfect completeness in the definition of DRMA in order to allow for the probability that the verifier chooses a bad random string r.

4 A Composition Theorem for Rational Proofs

In this Section we prove an intuitively simple, but technically non-trivial, *composition theorem* that states that we while proving the value of a function f, we can replace oracle access to a function g, with a rational proof for g. The technically interesting part of the proof is to make sure that the *total* reward of the prover is maximized when the result of the computation of f is correct. In other words, while we know that lying in the computation of g will not be a rational strategy for just that computation, it may turn out to be the best strategy as it might increase the reward of an incorrect computation of f. A similar issue (arising in a particular rational proof for depth d circuits) was discussed in [4]: our proof generalizes their technique.

Definition 3. *We say that a rational proof (P, V, rew) for f is a g-oracle rational proof if V has oracle access to the function g and carries out at most one oracle query. We allow the function g to depend on the specific input x.*

Theorem 2. *Assume there exists a g-oracle rational proof $(P_f^o, V_f^o, \mathsf{rew}_f^o)$ for f with noticeable reward gap and with round, communication and verification complexity respectively r_f, c_f and T_f. Let t_I the time necessary to invoke the oracle for g and to read its output.Assume there exists a rational proof $(P_g, V_g, \mathsf{rew}_g)$ with noticeable reward gap for g with round, communication and verification complexity respectively r_g, c_g and T_g. Then there exists a (non g-oracle) rational proof with noticeable reward gap for f with round, communication and verification complexity respectively $r_f + 1 + r_g, c_f + t_I + c_g$ and $T_f - t_i + T_g$.*

Before we embark on the proof of Theorem 2 we state a technical Lemma whose simple proof is omitted for lack of space. The definition of rational proof requires that the expected reward of the honest prover is not lower than the expected reward of any other prover. The following intuitive lemma states we necessarily obtain this property if an honest prover has a polynomial expected gain in comparison to provers that *always* provide a wrong output.

Lemma 1. *Let (P, V) be a protocol and rew a reward function as in Definition 1. Let f be a function s.t. $\forall x \ \Pr[\mathsf{out}(P, V)(x)] = 1$. Let Δ be the corresponding reward gap w.r.t. the honest prover P and f. If $\Delta > \frac{1}{\mathsf{poly}(n)}$ then (P, V, rew) is a rational proof for f and admits noticeable reward gap.*

Now we can start the proof of Theorem 2.

Proof. Let rew^o_f and rew_g be the reward functions of the g-oracle rational proof for f and the rational proof for g respectively. We now construct a new verifier V for f. This verifier runs exactly like the g-oracle verifier for f except that every oracle query to g is now replaced with an invocation of the rational proof for g. The new reward function rew is defined as: $\mathsf{rew}(\mathcal{T}) = \delta\mathsf{rew}^o_f(\mathcal{T}^o_f \circ y_g) + \mathsf{rew}_g(\mathcal{T}_g)$. Here $\mathcal{T}$ is the complete transcript of the new rational proof, $\mathcal{T}^o_f$ is the transcript of the oracle rational proof for f, $\mathcal{T}_g$ and y_g are respectively the transcript and the output of the rational proof for g. Finally δ is multiplicative factor in $(0, 1])$. The intuition behind this formula is to "discount" the part of the reward from f so that the prover is incentivized to provide the true answer for g. In turn, since rew^o_f rewards the honest prover more when the verifier has the right answer for a query to g (by hypothesis), this entails that the whole protocol is rational proof for f.

To prove the theorem we will use Lemma 1 and it will suffice to prove that the new protocol has a noticeable reward gap.

Consider a prover $\widetilde{P}$ that always answer incorrectly on the output of f. Let p_g be the probability that the prover outputs a correct y_g. Then the difference between the expected reward of the honest prover and $\widetilde{P}$ is:

$$\delta(R^o_f - \tilde{R}^o_f) + (R_g - \tilde{R}_g) = \tag{1}$$

$$\delta(R^o_f - p_g\tilde{R}^{o,\text{good}(g)}_f - (1-p_g)\tilde{R}^{o,\text{wrong}(g)}_f)$$

$$+ (R_g - p_g\tilde{R}^{\text{good}(g)}_g - (1-p_g)\tilde{R}^{\text{wrong}(g)}_g) = \tag{2}$$

$$\delta(p_g(R^o_f - \tilde{R}^{o,\text{good}(g)}_f) + (1-p_g)(R_f - \tilde{R}^{o,\text{wrong}(g)}_f))$$

$$+ p_g(R_g - \tilde{R}^{\text{good}(g)}_g) + (1-p_g)(R_g - \tilde{R}^{\text{wrong}(g)}_g) > \tag{3}$$

$$p_g\delta\Delta^o_f + (1-p_g)(\Delta_g - \delta b^o_f(n)) \geq \tag{4}$$

$$\min\{\delta\Delta^o_f, \Delta_g - \delta b^o_f(n)\} > \tag{5}$$

$$\frac{1}{\mathsf{poly}(n)} \tag{6}$$

where the last inequality holds for $\delta = \frac{\Delta_g}{2b^o_f(n)}$.

The round, communication and verification complexity of the construction is given by the sum of the respective complexities from the two rational proofs modulo minor adjustments. These adjustments account for the additional round by which the verifier communicates to the prover the requested instance for g. □

We can use this result as a design tool of rational proofs for a function f: First build a rational proof for a function g and then one for f where we assume the verifier has oracle access to g. This automatically provides a complete rational proof for f.

Remark 2. Theorem 2 assumes that verifier in the oracle rational proof for f carries out a single oracle query. Notice however that the proof of the theorem can be generalized to any verifier carrying out a constant number of adaptive oracle queries, possibly all for distinct functions. This can be done by iteratively applying the theorem to a sequence of $m = O(1)$ oracle rational proofs for functions $f_1, ..., f_m$ where the i-th rational proof is f_{i+1}-oracle for $1 \leq i < m$.

4.1 Rational Proofs for Randomized Circuits

As an application of the composition theorem described above we present an alternative approach to rational proofs for randomized computations. We show that by assuming the existence of a *common reference string (CRS)*[4] we obtain rational proofs for randomized circuits of polylogarithmic depth and polynomial size, i.e. BPNC the class of uniform $\mathsf{polylog}$-depth poly-size randomized circuits with error bounded by $1/3$ on both sides.

If we insist on a "super-efficient" verifier (i.e. with sublinear running time) we cannot use the same approach as in Sect. 3.1 since we do not know how to bound the space $S(n)$ used by a computation in NC (and the verifier's complexity in our protocol for bounded space computations, depends on the space complexity of the underlying language). We get around this problem by assuming a CRS, to which the verifier has oracle access.

We start by describing a rational proof with oracle access for BPP and then we show how to remove the oracle access (via our composition theorem) for the case of BPNC.

Let $L \in \mathsf{BPP}$ and let M a PTM that decides L in polynomial time and $\rho(\cdot)$ the randomness complexity of M. For $x \in \{0,1\}^*$ we denote by L_x the (deterministically decidable) language $\{(x,r) : r \in \{0,1\}^{\rho(|x|)} \wedge M(x,r) = L(x)\}$.

Lemma 2. *Let L be a language in* BPP*. Then there exists a L_x-oracle rational proof with CRS σ for L where $|\sigma| = \mathsf{poly}(n)\rho(n)$.*

Proof. Our construction is as follows. W.l.o.g. we will assume σ to be divided in $\ell = \mathsf{poly}(n)$ blocks $r_1, ..., r_\ell$, each of size $\rho(n)$.

1. The honest prover P runs $M(x, r_i)$ for $1 \leq i \leq \ell$ and announces m the number of strings r_i s.t. $M(x, r_i)$ accepts, i.e. $\sum_i M(x, r_i)$;
2. P sends m to x.
3. The Verifier accepts if $m > \ell/2$

We note that if we set $y_i = M(x, r_i)$ then the prover is announcing the Hamming weight of the string $y_1, \ldots, y_\ell$. At this point we can use the Hamming weight verification protocol in Sect. 2.1 where the Verifier use the oracle for L_x to verify on her own the value of y_i.

[4] A common reference string is a string generated by a trusted party to which both the prover and the verifier have access; it is a common assumption in cryptographic literature, e.g. Non-Interactive Zero Knowledge [7].

We note that no matter which protocol is used, round complexity, communication complexity and verifier running time (not counting the oracle calls) are all $\mathsf{polylog}(n)$.

To obtain our result for BPNC we invoke the following result from [16]:

Theorem 3. $\mathsf{NC} \subseteq \mathsf{DRMA}[\mathsf{polylog}(n), \mathsf{polylog}(n), \mathsf{polylog}(n)]$

The theorem above, together with Theorem 2 and Lemma 2 yields:

Corollary 4. *Let $x \in \{0,1\}^n$ and $L \in \mathsf{BPNC}$. Assuming the existence of a (polynomially long) CRS then there exists a rational proof for L with polylogarithmically many rounds, polylogarithmic communication and verification complexity.*

Notice that some problems (e.g. perfect matching) are not known to be in NC but are known to be in $\mathsf{RNC} \subseteq \mathsf{BPNC}$ [20].

5 Sequential Composability

Until now we have only considered agents who want to maximize their reward. But the reward alone, might not capture the complete utility function that the Prover is trying to maximize in his interaction with the Verifier. In particular we have not considered the *cost* incurred by the Prover to compute f and engage in the protocol. It makes sense then to define the *profit* of the Prover as the difference between the reward paid by the verifier and such cost.

As already pointed out in [4,15] the definition of Rational Proof is sufficiently robust to also maximize the profit of the honest prover and not just the reward. Indeed consider the case of a "lazy" prover $\tilde{P}$ that does not evaluate the function: let $\tilde{R}(x), \tilde{C}(x)$ be the reward and cost associated with $\tilde{P}$ on input x (while $R(x), C(x)$ are the values associated with the honest prover).

Obviously we want $R(x) - C(x) \geq \tilde{R}(x) - \tilde{C}(x)$ or equivalently $R(x) - \tilde{R}(x) \geq C(x) - \tilde{C}(x)$. Recall the notion of *reward gap*: the minimum difference between the reward of the honest prover and any other prover $\Delta(x) \leq R(x) - \tilde{R}(x)$. To maximize the profit it is then sufficient to change the reward by a a multiplier $M = C(x)/\Delta(x)$. Thus we have that $M(R(x) - \tilde{R}(x)) \geq C(x) \geq C(x) - \tilde{C}(x)$ as desired. This explains why we require the reward gap to be at least the inverse of a polynomial: this will maintain the total reward paid by the Verifier bounded by a polynomial.

5.1 Profit in Repeated Executions

In [8] we showed how if Prover and Verifier engage in repeated execution of a Rational Proof, where the Prover has a "budget" of computation cost that he is willing to invest, then there is no guarantee anymore that the profit is maximized by the honest prover. The reason is that it might be more profitable for the prover to use his budget to provide many incorrect answers than to provide a single correct answer. That's because incorrect (e.g. random) answers are "cheaper" to compute than the correct one and with the same budget B the prover can provide many of them while the entire budget might be necessary to solve a single problem correctly. If incorrect answers still receive a substantial reward then many incorrect answers may be more profitable and a rational prover will choose that strategy.

We refer the reader to [8] for concrete examples of situations where this happens in many of the protocols in [3,4,15,16].

This motivated us to consider a stronger definition which requires the reward to be somehow connected to the "effort" paid by the prover. The definition (stated below) basically says that if a (possibly dishonest) prover invests less computation than the honest prover then he must collect a smaller reward.

Definition 4 (Sequential Rational Proof). *A rational proof* (P, V) *for a function* $f : \{0,1\}^n \rightarrow \{0,1\}^n$ *is* (ϵ, K)-sequentially composable *for an input distribution* $\mathcal{D}$*, if for every prover* $\widetilde{P}$*, and every sequence of inputs* $x, x_1, \ldots, x_k$ *drawn according to* $\mathcal{D}$ *such that* $C(x) \geq \sum_{i=1}^{k} \tilde{C}(x_i)$ *and* $k \leq K$ *we have that* $\sum_i \tilde{R}(x_i) - R \leq \epsilon$.

The following Lemma is from [8].

Lemma 3. *Let* (P, V) *and* rew *be respectively an interactive proof and a reward function as in Definition 1; if* rew *can only assume the values* 0 *and* R *for some constant* R*, let* $\tilde{p}_x = \Pr[\mathsf{rew}((\widetilde{P}, V)(x)) = R]$*. If for* $x \in \mathcal{D}$*,* $\tilde{p}_x \leq \frac{\tilde{C}(x)}{C} + \epsilon$ *then* (P, V) *is* $(KR\epsilon, K)$*-sequentially composable for* $\mathcal{D}$*.*

The intuition behind our definition and Lemma 3 is that to produce the correct result, the prover must run the computation and incur its full cost; moreover for a dishonest prover his probability of "success" has to be no bigger than the fraction of the total cost incurred.

This intuition is impossible to formalize if we do not introduce a probability distribution over the input space. Indeed, for a specific input x a "dishonest" prover $\widetilde{P}$ could have the correct $y = f(x)$ value "hardwired" and could answer correctly without having to perform any computation at all. Similarly, for certain inputs x, x' and a certain function f, a prover $\widetilde{P}$ after computing $y = f(x)$ might be able to "recycle" some of the computation effort (by saving some state) and compute $y' = f(x')$ incurring a much smaller cost than computing it from scratch. This is the reason our definition is parametrized over an input distribution $\mathcal{D}$ (and all the expectations, including the computation of the reward, are taken over the probability of selecting a given input x).

A way to address this problem was suggested in [6] under the name of *Unique Inner State Assumption (UISA)*: when inputs x are chosen according to $\mathcal{D}$, then we assume that computing f requires cost T from any party: this can be formalized by saying that if a party invests $t = \gamma T$ effort (for $\gamma \leq 1$), then it computes the correct value only with probability negligibly close to γ (since a party can always have a "mixed" strategy in which with probability γ it runs the correct computation and with probability $1 - \gamma$ does something else, like guessing at random).

Using this assumption [6] solve the problem of the "repeated executions with budget" by requiring the verifier to check the correctness of a random subset of the the prover's answer by running the computation herself on that subset. This makes the verifier "efficient" only in an amortized sense.

In [8] we formalized the notion of Sequential Composability in Definition 4 and, using a variation of the UISA, we showed protocols that are sequentially composable where the verifier is efficient (i.e. polylog verification time) on *each execution*. Unfortunately that proof of sequential composability works only for a limited subclass of log-depth circuits.

5.2 Sequential Composability of Our New Protocol

To prove our protocol to be sequentially composable we need two main assumptions which we discuss now.

Hardness of Guessing States. Our protocol imposes very weak requirements on the prover: the verifier just checks a single computation step in the entire process, albeit a step chosen at random among the entire sequence. We need an equivalent of the UISA which states that for every correct transition that the prover is able to produce he must pay "one" computation step. More formally for any Turing Machine M we say that pair of configuration γ, γ' is M-correct if γ' can be obtained from γ via a single computation step of M.

Definition 5 (Hardness of State Guessing Assumption). *Let M be a Turing Machine and let L_M be the language recognized by M. We say that the* Hardness of State Guessing Assumption *holds for M, for distribution $\mathcal{D}$ and security parameter ϵ if for any machine A running in time t the probability that A on input x outputs more than t, M-correct pairs of configurations is at most ϵ (where the probability is taken over the choice of x according to the distribution $\mathcal{D}$ and the internal coin tosses of A).*

Adaptive vs. Non-Adaptive Provers. Assumption 5 guarantees that to come up with t correct transitions, the prover must invest at least t amount of work. We now move to the ultimate goal which is to link the amount of work invested by the prover, to his probability of success. As discussed in [8] it is useful to distinguish between *adaptive and non-adaptive provers.*

When running a rational proof on the computation of M over an input x, an *adaptive* prover allocates its computational budget *on the fly* during the execution of the rational proof. Conversely a *non-adaptive* prover $\widetilde{P}$ uses his computational budget to compute as much as possible about $M(x)$ before starting the protocol with the verifier. Clearly an adaptive prover strategy is more powerful than a non-adaptive one (since the adaptive prover can direct its computation effort where it matters most, i.e. where the Verifier "checks" the computation).

As an example, it is not hard to see that in our protocol an adaptive prover can succesfully cheat without investing much computational effort at all. The prover will answer at random until the very last step when he will compute and answer with a correct transition. Even if we invoke Assumption 5 a prover that invests only one computational step has a probability of success of $1 - \frac{1}{\mathsf{poly}(n)}$ (indeed the prover fails only if we end up checking against the initial configuration – this is the attack that makes Theorem 1 tight.).

Is it possible to limit the Prover to a non-adaptive strategy? As pointed out in [8] this could be achieved by imposing some "timing" constraints to the execution of the protocol: to prevent the prover from performing large computations while interacting with the Verifier, the latter could request that prover's responses be delivered "immediately", and if a delay happens then the Verifier will not pay the reward. Similar timing constraints have been used before in the cryptographic literature, e.g. see the notion of *timing assumptions* in the concurrent zero-knowedge protocols in [11]. Note that in order to require an "immediate" answer from the prover it is necessary that the latter stores all the intermediate configurations, which is why we require the prover to run in space $O(T(n)S(n))$ – this condition is not needed for the protocol to be rational in the stand-alone case, since even the honest prover could just compute the correct transition on the fly. Still this could be a problematic approach if the protocol is conducted over a network since the network delay will most likely be larger than the computation effort required by the above "cheating" strategy.

Another option is to assume that the Prover is computationally bounded (e.g. the rational argument model introduced in [15]) and ask the prover to commit to all the configurations in the computation before starting the interaction with the verifier. Then instead of sending the configuration, the prover will decommit it (if the decommitment fails, the verifier stops and pays 0 as a reward). If we use a Merkle-tree commitment, these steps can be performed and verified in $O(\log n)$ time.

In any case, for the proof we assume that non-adaptive strategies are the only rational ones and proceed in analyzing our protocol under the assumption that the prover is adopting a non-adaptive strategy.

The Proof. Under the above two assumptions, the proof of sequential composability is almost immediate.

Theorem 4. *Let $L \in \mathsf{NTISP}[\mathsf{poly}(n), S(n)]$ and M be a TM recognizing L. Assume that Assumption 5 holds for M, under input distribution $\mathcal{D}$ and parameter ϵ. Moreover assume the prover follows a non-adaptive strategy. Then the protocol of Sect. 3 is a $(KR\epsilon, K)$-sequentially composable rational proof under $\mathcal{D}$ for any $K \in \mathbb{N}, R \in \mathbb{R}_{\geq 0}$.*

Proof. Let $\widetilde{P}$ be a prover with a running time of t on input x. Let T be the total number of transitions required by M on input x, i.e. the computational cost of the honest prover.

Observe that $\tilde{p}_x$ is the probability that V makes the final check on one of the transitions correctly computed by $\widetilde{P}$. Because of Assumption 5 we know that the probability that $\widetilde{P}$ can compute more than t correct transitions is ϵ, therefore an upper bound on $\tilde{p}_x$ is $\frac{t}{T} + \epsilon$ and the Theorem follows from Corollary 3. □

6 Lower Bounds for Rational Proofs

In this section we discuss how likely it is will be able to find very efficient non-cryptographic rational protocols for the classes P and NP.

We denote by BPQP the class of languages decidable by a randomized algorithm running in quasi-polynomial time, i.e. $\mathsf{BPQP} = \bigcup_{k>0} \mathsf{BPTIME}[2^{O(log^k(n))}]$. Our theorem follows the same approach of Theorem 16 in [15][5].

Theorem 5. $\mathsf{NP} \not\subseteq \mathsf{DRMA}[\mathsf{polylog}(n), \mathsf{polylog}(n), \mathsf{poly}(n)]$ *unless* $\mathsf{NP} \subseteq \mathsf{BPQP}$.

Proof Sketch. Assume there exists a rational proof π_L for a language $L \in \mathsf{NP}$ with parameters as the ones above. We can build a PTM M to decide L as follows: *(i)* M generates all possible transcripts $\mathcal{T}$ for π_L; *(ii)* for each $\mathcal{T}$, M estimates the expected reward $R_{\mathcal{T}}$ associated to that transcript by sampling $\mathsf{rew}(\mathcal{T})$ t times (recall the reward function is probabilistic); *(iii)* M returns the output associated to transcript $\mathcal{T}^* = \arg\max_{\mathcal{T}} R_{\mathcal{T}}$.

Consider the space of the transcripts with a polylogarithmic number of rounds and bits exchanged. The number of possible transcripts in such protocol is bounded by $(2^{\mathsf{polylog}(n)})^{\mathsf{polylog}(n)} = 2^{\mathsf{polylog}(n)}$. Let Δ be the (noticeable) reward gap of the protocol. By using Hoeffding's inequality we can prove M can approximate each $R_{\mathcal{T}}$ within $\Delta/3$ with probability $2/3$ after $t = \mathsf{poly}(n)$ samples. Recalling the definition of reward gap (see Remark 1), we conclude M can decide L in randomized time $2^{\mathsf{polylog}(n)}$. □

It is not known whether $\mathsf{NP} \not\subseteq \mathsf{BPQP}$ is true, although this assumption has been used to show hardness of approximation results [21,23]. Notice that this assumption implies $\mathsf{NP} \not\subseteq \mathsf{BPP}$ [18].

Let us now consider rational proofs for P. By the following theorem they might require $\omega(\log(n))$ total communication complexity (since we believe $\mathsf{P} \subseteq \mathsf{BPNC}$ to be unlikely [25]).

Theorem 6. $\mathsf{P} \not\subseteq \mathsf{DRMA}[O(1), O(\log(n)), \mathsf{polylog}(n)]$ *unless* $\mathsf{P} \subseteq \mathsf{BPNC}$.

Proof Sketch. Given a language $L \in \mathsf{P}$ we build a machine M to decide L as in the proof of Theorem 5. The only difference is that M can be simulated by a randomized circuit of $\mathsf{polylog}(n)$ depth and polynomial size. In fact, all the possible $2^{O(\log(n))} = \mathsf{poly}(n)$ transcripts can be simulated in parallel in $O(\log(n))$ sequential time. The same holds computing the $t = \mathsf{poly}(n)$ sample rewards for each of these transcripts. By assumption on the verifier's running time, each reward can be computed in polylogarithmic sequential time. Finally, the estimate of each transcript's expected reward and the maximum among them can be computed in $O(\log(n))$ depth. □

Remark 3. Theorem 6 can be generalized to rational proofs with round and communication complexities r and c such that $r \cdot c = O(\log(n))$.

[5] Since we only sketch our proof the reader is invited to see details of the proof [15].

7 Conclusions and Open Problems

We presented a rational proof for languages recognized by (deterministic) space-bounded computations. Our protocol is the first rational proof for a general class of languages that does not use circuit representations. Our protocol is secure both in the standard stand-alone notion of rational proof [3] and in the stronger composable version in [8].

Our work leaves open a series of questions:

- Can we build efficient rational proofs for arbitrary poly-time computations, where the verifier runs in sub-linear (or even linear) time?
- Our proof of sequential composability considers only non-adaptive adversaries, and enforces this condition by the use of timing assumptions or computationally bounded provers. Is it possible to construct protocols that are secure against adaptive adversaries? Or is it possible to relax the timing assumption to something less stringent than what is required in our protocol?
- It would be interesting to investigate the connection between the model of Rational Proofs and the work on Computational Decision Theory in [17]. In particular looking at realistic cost models that could affect the choice of strategy by the prover particularly in the sequentially composable model.

In Sect. 1.5 we described the two main approaches to Rational Proofs design: scoring rules and weak interactive proofs. Trying and compare the power of these approaches, two natural questions arise:

- Does one approach systematically lead to more efficient rational proofs (in terms of rounds, communication and verifying complexity) than the other?
- Is one approach more suitable for sequential composability than the other?

We believe these two open questions are worth pursuing. Some discussion follows.

Regarding the first question: in the context of "stand-alone" (non sequential) rational proofs it is not clear which approach is more powerful. We know that for every language class known to admit a scoring rule based protocol we also have a weak interactive proof with similar performance metrics (i.e. number of rounds, verifier efficiency, etc.). Our result is the first example of a language class for which we have rational proofs based on weak interactive proofs but no example of a scoring rule based protocol exist[6]. This suggests that the weak interactive proof approach might be the more powerful technique. It is open if all rational proofs are indeed weak interactive proofs: i.e. that given a rational proof with certain efficiency parameters, one can construct a weak interactive proof with "approximately" the same parameters.

On the issue of sequential composability, we have already proven in [8] that some rational proofs based on scoring rules (such as Brier's scoring rule) are not

[6] We stress that in this comparison we are interested in protocols with similar efficiency parameters. For example, the work in [3] presents several large complexity classes for which we have rational proofs. However, these protocols require a polynomial verifier and do not obtain a noticeable reward gap.

sequentially composable. This problem might be inherent at least for scoring rules that pay a substantial reward to incorrect computations. What we can say is that all known sequentially composable proofs are based on weak interactive proofs ([4,8][7] and this work). Again it is open if this is required, i.e. that all sequentially composable rational proofs are weak interactive proofs.

Acknowledgments. The authors would like to thank Jesper Buus Nielsen for suggesting the approach of the construction in Theorem 1.

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@ home: an experiment in public-resource computing. Commun. ACM **45**(11), 56–61 (2002)
2. Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. J. Cryptology **23**(2), 281–343 (2010)
3. Azar, P.D., Micali, S.: Rational proofs. In: Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing, pp. 1017–1028. ACM (2012)
4. Azar, P.D., Micali, S.: Super-efficient rational proofs. In: Proceedings of the Fourteenth ACM Conference on Electronic Commerce, pp. 29–30. ACM (2013)
5. Babai, L.: Trading group theory for randomness. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, pp. 421–429. ACM (1985)
6. Belenkiy, M., Chase, M., Christopher Erway, C., Jannotti, J., Küpçü, A., Lysyanskaya, A.: Incentivizing outsourced computation. In: Proceedings of the ACM SIGCOMM 2008 Workshop on Economics of Networked Systems, NetEcon 2008, Seattle, WA, USA, 22 August 2008, pp. 85–90 (2008)
7. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. **20**(6), 1084–1118 (1991)
8. Campanelli, M., Gennaro, R.: Sequentially composable rational proofs. In: Khouzani, M.H.R., Panaousis, E., Theodorakopoulos, G. (eds.) GameSec 2015. LNCS, vol. 9406, pp. 270–288. Springer, Cham (2015). doi:10.1007/978-3-319-25594-1_15
9. Chen, J., McCauley, S., Singh, S.: Rational proofs with multiple provers. In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, pp. 237–248. ACM (2016)
10. De Agostino, S., Silvestri, R.: Bounded size dictionary compression: SC_k-completeness and NC algorithms. Inf. Comput. **180**(2), 101–112 (2003)
11. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. J. ACM **51**(6), 851–898 (2004)
12. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14623-7_25
13. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38348-9_37

[7] The construction in Theorem 5.1 in [4] is shown to be sequentially composable in [8].

14. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
15. Guo, S., Hubáček, P., Rosen, A., Vald, M.: Rational arguments: single round delegation with sublinear verification. In: Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, pp. 523–540. ACM (2014)
16. Guo, S., Hubáček, P., Rosen, A., Vald, M.: Rational sumchecks. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 319–351. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_12
17. Halpern, J.Y., Pass, R.: I don't want to think about it now: Decision theory with costly computation. arXiv preprint arXiv:1106.2657 (2011)
18. Johnson, D.S.: The np-completeness column: the many limits on approximation. ACM Trans. Algorithms (TALG) **2**(3), 473–489 (2006)
19. Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 485–494. ACM (2014)
20. Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in random nc. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, pp. 22–32. ACM (1985)
21. Khot, S., Ponnuswami, A.K.: Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 226–237. Springer, Heidelberg (2006). doi:10.1007/11786986_21
22. Larson, S.M., Snow, C.D., Shirts, M., Pande, V.S.: Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology. arXiv preprint arXiv:0901.0866 (2009)
23. Makarychev, K., Manokaran, R., Sviridenko, M.: Maximum quadratic assignment problem: reduction from maximum label cover and LP-based approximation algorithm. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 594–604. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14165-2_50
24. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica **12**(4), 449–461 (1992)
25. Papakonstantinou, P.A.: Constructions, lower bounds, and new directions in Cryptography and Computational Complexity. Ph.D. Thesis, University of Toronto (2010)
26. Reingold, O., Rothblum, R., Rothblum, G.: Constant-round interactive proofs for delegating computation. In: Proceedings of the Forty-Eighth Annual ACM on Symposium on Theory of Computing. ACM (2016). (page to appear)
27. Walfish, M., Blumberg, A.J.: Verifying computations without reexecuting them. Commun. ACM **58**(2), 74–84 (2015)