9-2018

Rationality and Efficient Verifiable Computation

Matteo Campanelli
The Graduate Center, City University of New York

Rationality and Efficient Verifiable Computation

by

Matteo Campanelli

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2018

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Professor Rosario Gennaro

Date	Chair of Examining Committee

Professor Robert Haralick

Date	Executive Officer

Professor Rosario Gennaro

Professor William Skeith III

Professor Itai Feigenbaum

Professor Mariana Raykova

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

RATIONALITY AND EFFICIENT VERIFIABLE COMPUTATION

by

MATTEO CAMPANELLI

Advisor: Professor Rosario Gennaro

In this thesis, we study protocols for delegating computation in a model where one of the parties is rational. In our model, a *delegator* outsources the computation of a function f on input x to a *worker*, who receives a (possibly monetary) reward. Our goal is to design *very efficient* delegation schemes where a worker is economically incentivized to provide the correct result $f(x)$. In this work we strive for not relying on cryptographic assumptions, in particular our results do not require the existence of one-way functions.

We provide several results within the framework of rational proofs introduced by Azar and Micali (STOC 2012). We make several contributions to efficient rational proofs for general feasible computations. First, we design schemes with a sublinear verifier with low round and communication complexity for space-bounded computations. Second, we provide evidence, as lower bounds, against the existence of rational proofs: with logarithmic communication and polylogarithmic verification for P and with polylogarithmic communication for NP.

We then move to study the case where a delegator outsources multiple inputs. First, we formalize an extended notion of rational proofs for this scenario (sequential composability) and we show that existing schemes do not satisfy it. We show how these protocols incentivize workers to provide many "fast" incorrect answers which allow them to solve more problems and collect more rewards. We then design a d-rounds rational proof for sufficiently "regular" arithmetic circuit of depth $d = O(\log n)$ with sublinear verification. We show, that under certain cost assumptions, our scheme is sequentially composable, i.e. it can be used to delegate multiple inputs. We finally show that our scheme for space-bounded computations is also sequentially composable under certain cost assumptions.

In the last part of this thesis we initiate the study of *Fine Grained Secure Computation*: i.e. the construction of *secure computation primitives* against "moderately complex" adversaries. Such fine-grained protocols can be used to obtain sequentially composable rational proofs. We present definitions and constructions for *compact* Fully Homomorphic Encryption and Verifiable Computation secure against (*non-uniform*) NC^1 adversaries. Our results hold under a widely believed separation assumption, namely $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$. We also present two application scenarios for our model: *(i)* hardware chips that prove their own correctness, and *(ii)* protocols against rational adversaries potentially relevant to the *Verifier's Dilemma* in smart-contracts transactions such as Ethereum.

Acknowledgments

This is one of the most important sections in a dissertation. In fact, although one's technical contributions may be modest or become obsolete soon after the time of writing, the acts by and the interactions with the people mentioned here are the real timeless stuff. G.H. Hardy would have disagreed, but that's another matter.

First, a special thanks to my advisor Rosario Gennaro. Thanks for many things, but especially for one. It is a hard search problem that of finding the intersection of what we can solve and what is interesting to us. Rosario constantly made an effort to give me the time, the space and the travel stipends (an often overlooked complexity measure) to eventually encounter some progress on it.

Thanks to the people that, directly or not, acted as mentors and teachers: Amotz Bar-Noy, Nelly Fazio, Dario Fiore, Leonid Gurvits, Tancrède Lepoint, Jesper Buus Nielsen, William Skeith, Noson Yanofsky. Noson taught the computational complexity course in my first term as a Ph.D. student. I owe to this course a stretch in my interests of geological proportions. My friends and colleagues, Kelly Eckenrode and Alessio Sammartano, offered meta-technical support and precious conversations throughout my Ph.D.; thank you for being an inspiration.

The work in this thesis would not have been possible without the interaction with so many people. My thanks go: to Marios Georgiou, for stretching my mind at every opportunity and tolerating my research-related phone calls at uncomfortable times of the day; to my colleagues at CCNY, Bertrand Ithurburn, Sima Jafarikhah, Nihal Vatandas, for the good chats and a great lab atmosphere; to my coauthors Steven Goldfeder and Luca Nizzardo, for their dedication and for making it fun.

I am deeply indebted to my parents, Domenico and Rita, for their unconditional love and support. I want to thank Nigel Deans and Annika Karinen; that little passion for science I may have, I virtually owe it all to you. I am indebted to $\mathcal{L}$, for being an outstanding listener and for telling me about The Rest.

Finally, thanks to Itai Feigenbaum and Mariana Raykova for being part of my committee, and to the indispensable support of the City College of New York through the Robert Kahn Fellowship and the NSF through grant 1545759.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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 [6, 29].

Verifiable Outsourced Computation is now a very active research area in Cryptography and Network Security (see [53] 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).

A different approach is to consider a model where "cheating" might actually be possible, but the provider would have no motivation to do so. In other words while cryptographic protocols prevent any adversary from cheating, one considers protocols that work against rational adversaries whose motivation is to maximize a well defined utility function.

We investigate this approach through two theoretical "lenses": *(i) rational proofs*, a variant of interactive proofs where provers lose (in economic terms) whenever they attempt to "prove" a false statement; *(ii) fine-grained protocols*, a model where parties's resources are assumed to be limited [1] and cheating is possible only through a larger amount of resources (e.g., cheating requires time λ^3, for some parameter λ, but all participating parties are assumed to be able to run in time at most λ^2). The connections between rationality and the latter model will be explored in the subsequent sections and in Appendix A.

[1]In a more specific sense than the usual "probabilistic polynomial time".

1.1 Rational Proofs

In the first two part of this thesis (Chapters 2 and 3) we use the concept of Rational Proofs introduced by Azar and Micali in [4] and refined in a subsequent paper [5].

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 [4], 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, where the Prover is polynomial and the Verifier is as efficient as possible, Azar and Micali in [5] show d-round Rational Proofs for functions computed by (uniform) Boolean circuits of depth d, for $d = O(\log n)$ (which can be collapsed to a single round under some well-defined computational assumption as shown in [31]). The problem of rational proofs for any polynomial-time computable function remains tantalizingly open.

Recent work [32] 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 [32] 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. [15] 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 $\mathsf{EXP}^{||\mathsf{NP}}$.

1.1.1 Contributions for Rational Proofs

Expressivity.

We present new protocols for the verification of *space-bounded polytime computations* against a rational adversary. More specifically, consider a language $L \in \mathsf{DTISP}(T(n), S(n))$, i.e. recognized by a deterministic Turing Machine M_L which runs in time $T(n)$ and space $S(n)$. In Section 2.3 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 $M_L(x)$

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. Our protocols is 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 [32], we note that it is believed that $\mathsf{NC} \neq \mathsf{SC}$ and that the two classes are actually incomparable (see [19] for a discussion). For these computations our results compare favorably to the one in [32] in at least one aspect: our protocol requires $O(\log n)$ rounds and has the same verification complexity.

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 [47] 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 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 presented in this work and can be of independent interest.

- Finally, we present lower bounds (i.e. conditional impossibility results) for Rational Proofs for various complexity classes.

Repeated Executions and Costly Computation.

Motivated by the problem of volunteer computation, our first result is to show that the definition of Rational Proofs in [4, 5] does not satisfy a basic compositional property which would make them applicable in that scenario. Consider the case where a large number of ”computation problems” are outsourced. Assume that solving each problem takes time T. Then in a time interval of length T, the honest prover can only solve and receive the reward for a single problem. On the other hand a dishonest prover, can answer up to T problems, for example by answering at random, a strategy that takes $O(1)$ time. To assure that answering correctly is a rational strategy, we need that at the end of the T-time interval the reward of the honest prover be larger than the reward of the dishonest one. But this is not necessarily the case: for some of the protocols in [4, 5, 31] we can show that a “fast” incorrect answer is more remunerable for the prover, by allowing him to solve more problems and collect more rewards.

The next questions, therefore, was to come up with a definition and a protocol that achieves rationality both in the stand-alone case, and in the composition described above. We first present an enhanced definition of Rational Proofs that removes the economic incentive for the strategy of fast incorrect answers, and then we present a protocol that achieves it for the case of some (uniform) bounded-depth circuits. Next, we design a d-rounds rational proof for sufficiently “regular” arithmetic circuit of depth $d = O(\log n)$ with sublinear verification. We show, that under certain cost assumptions, our scheme is sequentially composable, i.e. it can be used to delegate multiple inputs. We finally show that our scheme for space-bounded computations from Section 2.3 is also sequentially composable under certain cost assumptions.

1.1.2 Comparison with Other Prior Work

OTHER DECISION-THEORETIC FRAMEWORKS. An earlier work in the line of “rational verifiable computation” is [8] where the authors describe a system based on a scheme of rewards [resp. penalties] that the client assesses to the server for computing the function correctly [resp. incorrectly]. However in this system checking the computation may require re-executing it, something that the client does only on a randomized subset of cases, hoping that the penalty is sufficient to incentivize the server to perform honestly. Morever the scheme might require an ”infinite” budget for the rewards, and has no way to ”enforce” payment of penalties from cheating servers. For these reasons the best application scenario of this approach is the incentivization of volunteer computing schemes (such as SETI@Home or Folding@Home), where the rewards are non-fungible ”points” used for ”social-status”.

Because verification is performed by re-executing the computation, in this approach the client is ”efficient” (i.e. does ”less” work than the server) only in an amortized sense, where the cost of the subset of executions verified by the client is offset by the total number of computations performed by the server. This implies that the server must perform many executions for the client.

INTERACTIVE PROOFS. Obviously 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 [50] 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 [50]) 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 [50]. 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. [23, 24, 40]) even in the rational model [31, 32]. Recall that in an argument, security is achieved only against computationally bounded prover. In this case even single round solutions can be achieved. We will consider a variant of this model in Chapter 4.

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 [34] 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 [34]. There are similarities between the approach in this work and that in [34], as both take into account the cost of computation in a decision problem.

1.1.3 Future Directions

Our work leaves open a series of questions:

- What is the relationship between scoring rule based protocols vs weak interactive proofs? Our work seems to indicate that the latter technique is more powerful (our work shows an example of a class of language which is not known to be recognizable using scoring rules and that scoring rules seem inherently insecure in a composable setting). Is it possible to show, however, that a scoring-rule based protocol can be transformed into a weak interactive proof (without a substantial loss of efficiency) therefore showing that it is enough to focus on the latter?
- Can we build efficient rational proofs for arbitrary poly-time computations, where the verifier runs in sub-linear, or even in linear, time? Even in the standalone model of [4]?
- 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 [34]. In particular it would be interesting to look at realistic cost models that could affect the choice of strategy by the prover particularly in the sequentially composable model.

1.2 Fine-Grained Verifiable Computation

One of the crucial developments in Modern Cryptography has been the adoption of a more "fine-grained" notion of computational hardness and security. The traditional cryptographic approach modeled computational tasks as "easy" (for the honest parties to perform) and "hard" (infeasible for the adversary). Yet we have also seen a notion of *moderately hard* problems being used to attain certain security properties. The best example of this approach might be the use of moderately hard inversion problems used in blockchain protocols such as Bitcoin. Although present in many works since the inception of Modern Cryptography, this approach was first formalized in a work of Dwork and Naor [21].

In the second part of this thesis (Chapter 4) we consider the following model (which can be traced back to the seminal paper by Merkle [46] on public key cryptography). Honest parties will run a protocol which will cost[2] them C while an adversary who wants to compromise the security of the protocol will incur a $C' = \omega(C)$ cost. Note that while C' is asymptotically larger than C, it might still be a feasible cost to incur – the only guarantee is that it is substantially larger than the work of the honest parties. For example in Merkle's original proposal for public-key cryptography the honest parties can exchange a key in time T but the adversary can only learn the key in time T^2. Other examples include primitives introduced by Cachin and Maurer [11] and Hastad [35] where the cost is the space and parallel time complexity of the parties, respectively.

Recently there has been renewed interest in this model. Degwekar et al. [20] show how to construct certain cryptographic primitives in NC^1 [resp. AC^0] which are secure against all adversaries in NC^1 [resp. AC^0]. In conceptually related work Ball et al. [7] present computational problems which are "moderately hard" on average, if they are moderately hard in the worst case, a useful property for such problems to be used as cryptographic primitives.

The goal of this work is to initiate a study of *Fine Grained Secure Computation*. The question we ask is if it is possible to construct *secure computation primitives* that are secure against "moderately complex" adversaries. We answer this question in the affirmative, by presenting definitions and constructions for the task of Fully Homomorphic Encryption and Verifiable Computation in the fine-grained model. We also present two application scenarios for our model: i) hardware chips that prove their own correctness and ii) protocols against rational adversaries including potential

[2] We intentionally refer to it as "cost" to keep the notion generic. For concreteness one can think of C as the running time required to run the protocol.

solutions to the *Verifier's Dilemma* in smart-contracts transactions such as Ethereum.

Rationality and Fine-Grained Secure Computation.

In Chapter 3 we studied "sequentially composable" rational protocols, i.e. protocols where the reward is strictly connected, not just to the correctness of the result, but to the amount of work done by the prover. Our work on Fine-Grained Secure Computation complements those results, which only apply to a limited class of computations. In fact, a protocol secure in a fine-grained sense is also a sequentially composable rational proof: consider for example a protocol where the prover collects the reward only if he produces a proof of correctness of the result. Assume that the cost to produce a valid proof for an incorrect result, is higher than just computing the correct result and the correct proof. Then obviously a rational prover will always answer correctly, because the above strategy of fast incorrect answers will not work anymore.

1.2.1 Contributions for Fine-Grained Secure Computation

Our starting point is the work in [20] and specifically their public-key encryption scheme secure against NC^1 circuits. Recall that $\mathsf{AC}^0[2]$ is the class of Boolean circuits with constant depth, unbounded fan-in, augmented with parity gates. If the number of AND gates of non constant fan-in is constant we say that the circuit belongs to the class $\mathsf{AC}^0_\mathsf{Q}[2] \subset \mathsf{AC}^0[2]$.

Our results can be summarized as follows

- We first show that the techniques in [20] can be used to build a somewhat homomorphic encryption (SHE) scheme. We note that because honest parties are limited to NC^1 computations, the best we can hope is to have a scheme that is homomorphic for computations in NC^1. However our scheme can only support computations that can be expressed in $\mathsf{AC}^0_\mathsf{Q}[2]$.

- We then use our SHE scheme, in conjunction with protocols described in [23, 16, 2], to construct verifiable computation protocols for functions in $\mathsf{AC}^0_\mathsf{Q}[2]$, secure and input/output private against any adversary in NC^1.

Our somewhat homomorphic encryption also allows us to obtain the following protocols secure against NC^1 adversaries: *(i)* constant-round 2PC, secure in the presence of semi-honest static adversaries for functions in $\mathsf{AC}^0_{\mathsf{Q}}[2]$; *(ii) Private Function Evaluation* in a two party setting for circuits of constant *multiplicative* depth without relying on universal circuits. These results stem from well-known folklore transformations and we do not prove them formally.

The class $\mathsf{AC}^0_{\mathsf{Q}}[2]$ includes many natural and interesting problems such as: fixed precision arithmetic, evaluation of formulas in 3CNF (or kCNF for any constant k), a representative subset of SQL queries, and S-Boxes [9] for symmetric key encryption.

Our results (like [20]) hold under the assumption that $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$, a widely believed worst-case assumption on separation of complexity classes. Notice that this assumption does not imply the existence of one-way functions (or even $\mathsf{P} \neq \mathsf{NP}$). Thus, our work shows that it is possible to obtain "advanced" cryptographic schemes, such as somewhat homomorphic encryption and verifiable computation, even if we do not live in Minicrypt[3][4].

COMPARISON WITH OTHER APPROACHES. One important question is: on what features are our schemes better than "generic" cryptographic schemes that after all are secure against *any* polynomial time adversary.

One such feature is the type of assumption one must make to prove security. As we said above, our schemes rely on a very mild worst-case complexity assumption, while cryptographic SHE and VC schemes rely on very specific assumptions, which are much stronger than the above.

For the case of Verifiable Computation, we also have information-theoretic protocols which are secure against *any* (possibly computationally unbounded) adversary. For example the "Muggles" protocol in [28] which can compute any (log-space uniform) NC function, and is also reasonably efficient in practice [18]. Or, the more recent work [27], which obtains efficient VC for functions

[3]This is a reference to Impagliazzo's "five possible worlds" [36].
[4]Naturally the security guarantees of these schemes are more limited compared to their standard definitions.

in a subset of $\mathsf{NC} \cap \mathsf{SC}$. Compared to these results, one aspect in which our protocol fares better is that our Prover/Verifier can be implemented with a constant-depth circuit (in particular in $\mathsf{AC}^0[2]$, see Section 4.3) which is not possible for the Prover/Verifier in [28, 27], which needs to be in TC^0[5]. Moreover our protocol is non-interactive (while [28, 27] requires $\Omega(1)$ rounds of interaction) and because our protocols work in the "pre-processing model" we do not require any uniformity or regularity condition on the circuit being outsourced (which are required by [28] and [18]). Finally, out verification scheme achieves input and output privacy.

Finally, we compare our results with the information-theoretic approaches (mostly based on randomized encodings) in [30, 37, 1, 52]. From the techniques in these works one could obtain somewhat homomorphic encryption and verifiable computation in low-depth circuits (even in NC^0). Here, however, we stress that we are interested in *compact* homomorphic encryption schemes (where the ciphertexts do not grow in size with each homomorphic operation) and in verifiable computation schemes where the total work of the verifier approximately linear in the I/O size (i.e. the size of the verification circuit should be $O(\mathsf{poly}(\lambda)(n + m))$ where n and m are the size of the input and output respectively). The techniques in these works cannot directly achieve these goals. In fact, for homomorphic encryption, they lead to ciphertexts of size exponential in d, where d is the depth of the (fan-in two) evaluation circuit. For verifiable computation, they lead to verification with quadratic running time[6].

1.2.2 Technical Highlights

In [20] the authors already point out that their scheme is linearly homomorphic. We make use of the *re-linearization* technique from [10] to construct a leveled homomorphic encryption.

[5]The techniques in [28, 27] are based on properties of finite fields. Arithmetic in such fields can be carried out by threshold circuits of constant depth, but not in $\mathsf{AC}^0[2]$.

[6]On why this running time: in a straightforward application of these approaches we would have the verifier computing the (randomized) encoding of a function $f \in \mathsf{NC}^1$. The work necessary for this is quadratic in the size of the branching program computing f [37] (this is cubic if we use the approach in [30], described in Guy Rothblum's thesis [51]).

Our scheme (as the one in [20]) is secure against adversaries in the class of (*non-uniform*) NC^1. This implies that we can only evaluate functions in NC^1 otherwise the evaluator would be able to break the semantic security of the scheme. However we have to ensure that the *whole* homomorphic evaluation stays in NC^1. The problem is that homomorphically evaluating a function f might increase the depth of the computation.

In terms of circuit depth, the main overhead will be (as usual) the computation of multiplication gates. As we show in Section 4.2 a single homomorphic multiplication can be performed by a depth two $\mathsf{AC}^0[2]$ circuit, but this requires depth $O(\log(n))$ with a circuit of fan-in two. Therefore, a circuit for f with $\omega(1)$ multiplicative depth would require an evaluation of $\omega(\log(n))$ depth, which would be out of NC^1. Therefore our first scheme can only evaluate functions with constant multiplicative depth, as in that case the evaluation stays in $\mathsf{AC}^0[2]$.

We then present a second scheme that extends the class of computable functions to $\mathsf{AC}^0_{\mathsf{Q}}[2]$ by allowing for a negligible error in the correctness of the scheme. We use techniques from a work by Razborov [49] on approximating $\mathsf{AC}^0[2]$ circuits with low-degree polynomials – the correctness of the approximation (appropriately amplified) will be the correctness of our scheme.

1.2.3 Application Scenarios

The applications described in this section refer to the problem of Verifying Computation, where a Client outsources an algorithm f and an input x to a Server, who returns a value y and a proof that $y = f(x)$. The security property is that it should be infeasible to convince the verifier to accept $y' \neq f(x)$, and the crucial efficiency property is that verifying the proof should cost less than computing f (since avoiding that cost was the reason the Client hired the Server to compute f).

HARDWARE CHIPS THAT PROVE THEIR OWN CORRECTNESS Verifiable Computation (VC) can be used to verify the execution of hardware chips designed by untrusted manufacturers. One could envision chips that provide (efficient) *proofs of their correctness* for every input-output computation they perform. These proofs must be *efficiently verified* in less time and energy than it takes to re-execute the computation itself.

When working in hardware, however, one may not need the full power of cryptographic protection against *any* malicious attacks since one could bound the computational power of the malicious chip. The bound could be obtained by making (reasonable and evidence-based) assumptions on how much computational power can fit in a given chip area. For example one could safely assume that a malicious chip can perform at most a constant factor more work than the original function because of the basic physics of the size and power constraints. In other words, if C is the cost of the honest Server in a VC protocol, then in this model the adversary is limited to $O(C)$-cost computations, and therefore a protocol that guarantees that succesful cheating strategies require $\omega(C)$ cost, will suffice. This is exactly the model in our paper. Our results will apply to the case in which we define the cost as the depth (i.e. the parallel time complexity) of the computation implemented in the chip.

THE VERIFIER'S DILEMMA. In blockchain systems such as Ethereum, transactions can be expressed by arbitrary programs. To add a transaction to a block miners have to verify its validity, which could be too costly if the program is too complex. This creates the so-called *Verifier's Dilemma* [43]: given a costly valid transaction Tr a miner who spends time verifying it is at a disadvantage over a miner who does not verify it and accept it "uncritically" since the latter will produce a valid block faster and claim the reward. On the other hand if the transaction is invalid, accepting it without verifying it first will lead to the rejection of the entire block by the blockchain and a waste of work by the uncritical miner. The solution is to require efficiently verifiable proofs of validity for transactions, an approach already pursued by various startups in the Ethereum ecosystem (e.g. TrueBit[7]). We note that it suffices for these proofs to satisfy the condition above: i.e. we do not need the full power of information-theoretic or cryptographic security but it is enough to guarantee that to produce a proof of correctness for a false transaction is more

[7]TrueBit: *https://truebit.io/*

costly than producing a valid transaction and its correct proof, which is exactly the model we are proposing.

1.2.4 Future Directions

Our work opens up many interesting future directions.

First of all, it would be nice to extend our results to the case where cost is the actual running time, rather than "parallel running time"/"circuit depth" as in our model. The techniques in [7] (which presents problems conjectured to have $\Omega(n^2)$ complexity on the average), if not even the original work of Merkle [46], might be useful in building a verifiable computation scheme where if computing the function takes time T, then producing a false proof of correctness would have to take $\Omega(T^2)$.

For the specifics of our constructions it would be nice to "close the gap" between what we can achieve and the complexity assumption: our schemes can only compute $\mathsf{AC}^0_{\mathsf{Q}}[2]$ against adversaries in NC^1, and ideally we would like to be able to compute all of NC^1 (or at the very least all of $\mathsf{AC}^0[2]$).

Finally, to apply these schemes in practice it is important to have tight concrete security reductions and a proof-of-concept implementations.

1.3 Notation and Common Preliminaries

For a distribution D, we denote by $x \leftarrow D$ the fact that x is being sampled according to D. We remind the reader that an ensemble $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ is a family of probability distributions over a family of domains $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ are statistically indistinguishable if $\frac{1}{2}\sum_x |D(x) - D'(x)| < \mathsf{neg}(\lambda)$.

1.4 Thesis Roadmap

This thesis combines research done in several works. **Chapter 2** contains results on rational proofs for subclasses of polynomial-time computations in the model of Azar and Micali [4]. **Chapter 3** contains a critique of the original model of raitonal proofs, definitions for sequential composability and related results. The results of these two chapters are joint work with Rosario Gennaro, originally presented in [14] and [12] (notice that each of the two chapters do not correspond to each of the two works). **Chapter 4** contains results on homomorphic encryptions and delegating computation in a fine-grained model where adversaries are NC^1 circuits. The results in this chapter are joint work with Rosario Gennaro, originally presented in [13].

Chapter 2

Rational Proofs for Subclasses of P

2.1 The Landscape of Rational Proof Systems

Rational Proof systems can be divided in roughly two categories, both of them presented in the original work [4].

SCORING RULES. The more "novel" approach in [4] 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 [4] 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 [4] (or its "scaled-down" version for Hamming weight described more in detail in Section 2.2). Given a Boolean formula $\Phi(x_1, \ldots, x_n)$ 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 Φ 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.

As pointed out in [14] 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 [4, 5, 31, 32] suffer from this problem.

WEAK INTERACTIVE PROOFS. The definition of rational proofs requires that the expected reward is maximized for the honest prover. This definition can be made stronger (as done explicitly in [31]) and require a that every systematically dishonest prover would incur a polynomial loss (this property is usually described in terms of a *noticeable reward gap*). As discussed above, the elegant device of scoring rules is the basis for most rational proof protocols in literature, some of which achieve noticeable reward gap. Another simple way in which we can obtain this stronger type of rational proof is the following. Imagine having a test where the prover can be caught cheating with "low", but non-negligible probability, e.g. n^{-k} for some $k \in \mathbb{N}$. We will informally call this test a *weak interactive proof*[1]. Indeed for such proofs we can always pay a fixed reward R to the prover unless we catch him cheating in which case we pay 0. These are rational proofs since obviously the expected reward of the prover is maximized by the honest behavior. Some of the proofs in [4] and the proofs in [14] are weak interactive proofs. Those proofs also turn out to be secure in the sequential model of [14] (under appropriate assumptions).

The protocols in this work are weak interactive proof, which is why we can prove them to be sequentially composable[2].

SCORING RULES VS. WEAK INTERACTIVE PROOFS. Comparing approaches based on scoring rules and weak interactive proofs the following two questions come up:

- 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 do not have a precise answer to the above questions, which we believe are interesting open problems to consider. However we can make the following statements.

[1]This is basically the *covert adversary* model for multiparty computation introduced in [3].

[2] One exception is the protocol for BPNC, which depends on the underlying protocol for (deterministic) NC. If we use the one in [32], then the resulting protocol uses scoring rules and is not sequentially composable. However an alterative protocol for NC can be used, based on our work in [14], which is a weak interactive proof and can be proven sequentially composable.

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.). The result in this paper 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[3]. This suggests that the weak interactive proof approach might be the more powerful technique. It would be interesting to prove that 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. This question is left as future work.

On the issue of sequential composability, we have already proven in [14] that some rational proofs based on scoring rules (such as Brier's scoring rule) are not 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 ([14], [5][4] and this work). Again it would be interesting to prove that this is required, i.e. that all sequentially composable rational proofs are weak interactive proofs.

2.2 Definitions and Preliminaries

The following is the definition of Rational Proof from [4]. 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 2.2.1 (Rational Proof)**.** A function $f : \{0, 1\}^n \rightarrow \{0, 1\}^*$ admits a rational proof if there

[3] We stress that in this comparison we are interested in protocols with similar efficiency parameters. For example, the work in [4] 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.

[4]The construction in Theorem 5.1 in [5] is shown to be sequentially composable in [14].

exists an interactive proof (P, V) and a randomized reward function $\mathsf{rew} : \{0,1\}^* \to \mathbb{R}_{\geq 0}$ 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 [4] 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 [31] we define the 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 2.2.2 ([4, 5, 31])**.** The class $\mathsf{DRMA}[r, c, T]$ (Decisional Rational Merlin Arthur) is the class of boolean functions $f : \{0,1\}^* \to \{0,1\}$ 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 [5, 31] and is explained in Section 3.5.*

EXAMPLES OF RATIONAL PROOFS. For concreteness here we show the protocol for a single threshold gate (readers are referred to [4, 5, 31] for more examples).

Let $G_{n,k}(x_1, \ldots, x_n)$ be a threshold gate with n Boolean inputs, that evaluates to 1 if at least k of the input bits are 1. The protocol in [5] to evaluate this gate goes as follows. The Prover announces the number $\tilde{m}$ of input bits equal to 1, which allows the Verifier to compute $G_{n,k}(x_1, \ldots, x_n)$. The Verifier select a random index $i \in [1..n]$ and looks at input bit $b = x_i$ and rewards the Prover using Brier's Rule $BSR(\tilde{p}, b)$ where $\tilde{p} = \tilde{m}/n$ i.e. the probability claimed by the Prover that a randomly selected input bit be 1. Then

$$BSR(\tilde{p}, 1) = 2\tilde{p} - \tilde{p}^2 - (1-\tilde{p})^2 + 1 = 2\tilde{p}(2-\tilde{p})$$

$$BSR(\tilde{p}, 0) = 2(1-\tilde{p}) - \tilde{p}^2 - (1-\tilde{p})^2 + 1 = 2(1-\tilde{p}^2)$$

Let m be the true number of input bits equal to 1, and $p = m/n$ the corresponding probability, then the expected reward of the Prover is

$$pBSR(\tilde{p}, 1) + (1-p)BSR(\tilde{p}, 0) \tag{2.1}$$

which is easily seen to be maximized for $p = \tilde{p}$ i.e. when the Prover announces the correct result. Moreover one can see that when the Prover announces a wrong $\tilde{m}$ his reward goes down by $2(p-\tilde{p})^2 \geq 2/n^2$. In other words for all n-bit input x, we have $\Delta(x) = 2/n^2$ and if a dishonest Prover $\widetilde{P}$ cheats with probability $\epsilon_{\widetilde{P}}$ then $\delta_{\widetilde{P}} > 2\epsilon_{\widetilde{P}}/n^2$.

2.3 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 [14] 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 2.3.1. $\mathsf{DTISP}[\mathsf{poly}(n), S(n)] \subseteq \mathsf{DRMA}[O(\log n), O(S(n) \log n), O(S(n) \log n)]$

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

This improves over the construction of rational proofs for L in [32] due to the better round complexity.

Corollary 2.3.3. $\mathsf{SC} \subseteq \mathsf{DRMA}[O(\log n), O(\mathsf{polylog}(n)), O(\mathsf{polylog}(n))]$

No known result was known for SC before.

2.3.1 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 [47] 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[5] (see in particular Theorem 3 in [47]). 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.

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 2.3.1 we have the following result:

Corollary 2.3.4. $\mathsf{BPTISP}[\mathsf{poly}(n), S(n)] \subseteq \mathsf{DRMA}[\log(n), S(n)\log^2(n), S(n)\log^2(n)]$

We note that for this protocol, we need to allow for non-perfect completeness in the definition

[5]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.

of DRMA in order to allow for the probability that the verifier chooses a bad random string r.

2.4 A Composition Theorem for Rational Proofs

In this Section we prove a relatively simple *composition theorem* that states that 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 [5]: our proof generalizes their technique.

Definition 2.4.1. 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 2.4.2. 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 2.4.2, we prove a technical Lemma. 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 2.4.3. Let (P, V) be a protocol and rew a reward function as in Definition 2.2.1. 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 $\Delta > \frac{1}{\mathsf{poly}}$ then (P, V, rew) is a rational proof for f and admits noticeable reward gap.

Proof. Assume w.l.o.g that for all $P' \neq P$ and such that $\forall x \Pr[\mathsf{out}(P', V)(x)] = 1$ it holds that $\mathbb{E}[\mathsf{rew}(P, V)(x)] \geq \mathbb{E}[\mathsf{rew}(P', V)(x)]$.

Fix x. Let $\widetilde{P}$ be an arbitrary prover, $R = \mathbb{E}[\mathsf{rew}(P, V)(x)]$, $\tilde{y} = \mathsf{out}(\widetilde{P}, V)(x)$, $\tilde{R} = \mathbb{E}[\mathsf{rew}(\widetilde{P}, V)(x)]$, $\tilde{R}_{corr} = \mathbb{E}[\mathsf{rew}(\widetilde{P}, V)(x) | \tilde{y} = f(x)]$, $\tilde{R}_{err} = \mathbb{E}[\mathsf{rew}(\widetilde{P}, V)(x) | \tilde{y} \neq f(x)]$. Then:

$$R - \tilde{R} \quad = \tag{2.2}$$

$$R - \Pr[\tilde{y} = f(x)]\tilde{R}_{corr} - \Pr[\tilde{y} \neq f(x)] \quad = \tag{2.3}$$

$$\Pr[\tilde{y} = f(x)](R - \tilde{R}_{corr}) + \Pr[\tilde{y} \neq f(x)](R - \tilde{R}_{err}) \quad \geq \tag{2.4}$$

$$\Pr[\tilde{y} \neq f(x)](R - \tilde{R}_{err}) \quad \geq \tag{2.5}$$

$$\Pr[\tilde{y} \neq f(x)]\Delta \quad > \tag{2.6}$$

$$0 \tag{2.7}$$

The inequality above shows that (P, V, rew) is a rational proof for f. By the hypothesis on Δ this protocol already admits a noticeable reward gap. □

The proof of Theorem 2.4.2 follows.

Proof. Let rew_f^o 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 follows:

$$\mathsf{rew}(\mathcal{T}) = \delta \mathsf{rew}_f^o(\mathcal{T}_f^o \circ y_g) + \mathsf{rew}_g(\mathcal{T}_g)$$

where $\mathcal{T}$ is the complete transcript of the new rational proof, $\mathcal{T}_f^o$ 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_f^o 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 2.4.3 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:

$$
\begin{aligned}
\delta(R_f^o - \tilde{R}_f^o) + (R_g - \tilde{R}_g) = && (2.1)\\
\delta(R_f^o - p_g \tilde{R}_f^{o,\text{good}(g)} - (1-p_g)\tilde{R}_f^{o,\text{wrong}(g)}) + &&\\
(R_g - p_g \tilde{R}_g^{\text{good}(g)} - (1-p_g)\tilde{R}_g^{\text{wrong}(g)}) = && (2.2)\\
\delta(p_g(R_f^o - \tilde{R}_f^{o,\text{good}(g)}) + (1-p_g)(R_f - \tilde{R}_f^{o,\text{wrong}(g)})) &&\\
+p_g(R_g - \tilde{R}_g^{\text{good}(g)}) + (1-p_g)(R_g - \tilde{R}_g^{\text{wrong}(g)}) > && (2.3)\\
\delta(p_g\Delta_f^o + (1-p_g)(-b_f^o(n))) + 0 + (1-p_g)\Delta_g = && (2.4)\\
p_g\delta\Delta_f^o + (1-p_g)(\Delta_g - \delta b_f^o(n)) \geq && (2.5)\\
\min\{\delta\Delta_f^o, \Delta_g - \delta b_f^o(n)\} > && (2.6)\\
\frac{1}{\mathsf{poly}} && (2.7)
\end{aligned}
$$

Where the last inequality holds for $\delta = \frac{\Delta_g}{2b_f^o(n)}$.

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.

□

The theorem above can be used as design tool of rational proofs for a function f: first build a rational proof assuming the verifier has oracle access to a function g, then build a rational proof for g. This automatically provides a complete rational proof for f.

Remark 2. Theorem 2.4.2 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 $1 \leq i < m$.

2.4.1 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)* we obtain rational proofs for randomized circuits of polylogarithmic depth and polynomial size, i.e. BPNC the class of uniform 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 Section 2.3.1 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 remoe the oracle access (via our composition theorem) for the case of BPNC.

Let $L \in$ 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 2.4.4. 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 Section 2.2 where the Verifier use the oracle for L_x to verify on her own the value of y_i. □

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 [32]:

Theorem 2.4.5. $\mathsf{NC} \subseteq \mathsf{DRMA}[\mathsf{polylog}(n), \mathsf{polylog}(n), \mathsf{polylog}(n)]$

The theorem above, together with Theorem 2.4.2 and Lemma 2.4.4 yields:

Corollary 2.4.6. 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 $\mathsf{RNC} \subseteq \mathsf{BPNC}$ [41].

2.5 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 [31][6].

Theorem 2.5.1. $\mathsf{NP} \not\subseteq \mathsf{DRMA}[\mathsf{polylog}(n), \mathsf{polylog}(n), \mathsf{poly}(n)]$ unless $\mathsf{NP} \subseteq \mathsf{BPQP}$.

[6]Since we only sketch our proof the reader is invited to see details of the proof [31]

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:

- M generates all possible transcripts $\mathcal{T}$ for π_L;
- 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);
- M returns the output associated to transcript $\mathcal{T}^* = \arg\max_{\mathcal{T}} R_{\mathcal{T}}$.

Consider that space of the transcripts of rational proof 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 4), 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 [44, 42]. Notice that this assumption implies $\mathsf{NP} \not\subseteq \mathsf{BPP}$ [39].

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 [48]).

Theorem 2.5.2. $\mathsf{P} \not\subseteq \mathsf{DRMA}[O(1), O(\log(n)), \mathsf{polylog}(n)]$ 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 2.5.1. 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.

$\square$

Remark 3. Theorem 2.5.2 can be generalized to rational proofs with round and communication complexities r and c such that $r \cdot c = O(\log(n))$.

Chapter 3

Rational Proofs for Costly Multiple Delegations

3.1 Profit vs. Reward

Let us now define the profit of the Prover as the difference between the reward paid by the verifier and the cost incurred by the Prover to compute f and engage in the protocol. As already pointed out in [5, 31] the definition of Rational Proof is sufficiently robust to also maximize the profit of the honest prover and not the reward. Indeed consider the case of a "lazy" prover $\widetilde{P}$ that does not evaluate the function: even if $\widetilde{P}$ collects a "small" reward, his total profit might still be higher than the profit of the honest prover P.

Set $R(x) = \mathbb{E}[\mathsf{rew}((P, V)(x))]$, $\tilde{R}(x) = \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x))]$ and $C(x)$ [resp. $\tilde{C}(x)$] the cost for P [resp. $\widetilde{P}$] to engage in the protocol. Then we want

$$R(x) - C(x) \geq \tilde{R}(x) - \tilde{C}(x) \implies \delta_{\widetilde{P}}(x) \geq C(x) - \tilde{C}(x)$$

In general this is not true (see for example the previous protocol), but it is always possible to change the reward by a multiplier M. Note that if $M \geq C(x)/\delta_{\widetilde{P}}(x)$ then we have that

$$M(R(x) - \tilde{R}(x)) \geq C(x) \geq C(x) - \tilde{C}(x)$$

as desired. Therefore by using the multiplier M in the reward, the honest prover P maximizes its profit against all provers $\widetilde{P}$ except those for which $\delta_{\widetilde{P}}(x) \leq C(x)/M$, i.e. those who report the incorrect result with a "small" probability $\epsilon_{\widetilde{P}}(x) \leq \frac{C(x)}{M\Delta(x)}$.

We note that M might be bounded from above, by budget considerations (i.e. the need to keep the total reward $MR(x) \leq B$ for some budget B). This point out to the importance of a large reward gap $\Delta(x)$ since the larger $\Delta(x)$ is, the smaller the probability of a cheating prover $\widetilde{P}$ to report an incorrect result must be, in order for $\widetilde{P}$ to achieve an higher profit than P.

EXAMPLE. In the above protocol we can assume that the cost of the honest prover is $C(x) = n$, and we know that $\Delta(x) = n^2$. Therefore the profit of the honest prover is maximized against all the provers that report an incorrect result with probability larger than n^3/M, which can be made sufficiently small by choosing the appropriate multiplier.

Remark 4. *If we are interested in an asymptotic treatment, it is important to notice that as long as* $\Delta(x) \geq 1/\mathsf{poly}(|x|)$ *then it is possible to keep a polynomial reward budget, and maximize the honest prover profit against all provers who cheat with a substantial probability* $\epsilon_{\widetilde{P}} \geq 1/\mathsf{poly}'(|x|)$.

3.2 Sequential Composition

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 [5, 31] 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

$$R(x) - C(x) \geq \tilde{R}(x) - \tilde{C}(x) \text{ or equivalently } R(x) - \tilde{R}(x) \geq C(x) - \tilde{C}(x)$$

Recall the notion of reward gap which is 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 is therefore sufficient to change the reward by a a multiplier $M = C(x)/\Delta(x)$

since then 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, since this will maintain the total reward paid by the Verifier bounded by a polynomial.

3.2.1 Motivating Example

We now show how in repeated executions of a Rational Proof a nd 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.

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.

Consider the protocol in the previous section for the computation of the function $G_{n,k}(\cdot)$. Assume that the honest execution of the protocol (including the computation of $G_{n,k}(\cdot)$) has cost $C = n$.

Assume now that we are given a sequence of n inputs $x^{(1)}, \ldots, x^{(i)}, \ldots$ where each $x^{(i)}$ is an n-bit string. In the following let m_i be the Hamming weight of $x^{(i)}$ and $p_i = m_i/n$.

Therefore the honest prover investing $C = n$ cost, will be able to execute the protocol only

only once, say on input $x^{(i})$. By setting $p = \tilde{p} = p_i$ in Eq. 2.1, we see that P obtains reward

$$R(x^{(i)}) = 2(p_i^2 - p_i + 1) \leq 2$$

Consider instead a prover $\widetilde{P}$ which in the execution of the protocol outputs a random value $\tilde{m} \in [0..n]$. The expected reward of $\widetilde{P}$ on any input $x^{(i)}$ is (by setting $p = p_i$ and $\tilde{p} = m/n$ in Eq. 2.1 and taking expectations):

$$\begin{aligned}\tilde{R}(x^{(i)}) &= \mathop{\mathbb{E}}_{m,b}[BSR(\frac{m}{n}, b)] \\ &= \frac{1}{n+1}\sum_{m=0}^{n}\mathop{\mathbb{E}}_{b}[BSR(\frac{m}{n}, b] \\ &= \frac{1}{n+1}\sum_{m=0}^{n}(2(2p_i \cdot \frac{m}{n} - \frac{m^2}{n^2} - p_i + 1)) \\ &= 2 - \frac{2n+1}{3n} > 1 \text{ for } n > 1.\end{aligned}$$

Therefore by "solving" just two computations $\widetilde{P}$ earns more than P. Moreover t the strategy of $\widetilde{P}$ has cost 1 and therefore it earns more than P by investing a lot less cost[1].

Note that "scaling" the reward by a multiplier M does not help in this case, since both the honest and dishonest prover's rewards would be multiplied by the same multipliers, without any effect on the above scenario.

We have therefore shown a rational strategy, where cheating many times and collecting many rewards is more profitable than collecting a single reward for an honest computation.

[1] If we think of cost as time, then in the same time interval in which P solves one problem, $\widetilde{P}$ can solve up to n problems, earning a lot more money, by answering fast and incorrectly.

3.2.2 Sequentially Composable Rational Proofs

The above counterexample motivates the following Definition which formalizes that the reward of the honest prover P must always be larger than the total reward of any prover $\widetilde{P}$ that invests less computation cost than P.

Technically this is not trivial to do, since it is not possible to claim the above for *any* prover $\widetilde{P}$ and *any* sequence of inputs, because it is possible that for a given input $\tilde{x}$, the prover $\widetilde{P}$ has "hard-wired" the correct value $\tilde{y} = f(\tilde{x})$ and can compute it without investing any work. We therefore propose a definition that holds for inputs randomly chosen according to a given probability distribution $\mathcal{D}$, and we allow for the possibility that the reward of a dishonest prover can be "negligibly" larger than the reward of the honest prover (for example if $\widetilde{P}$ is lucky and such "hardwired" inputs are selected by $\mathcal{D}$).

Definition 3.2.1 (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 $\widetilde{P}$, for a 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 $\sum_i \tilde{R}(x_i) - R \leq \epsilon$.

A few sufficient conditions for sequential composability follow.

Lemma 3.2.2. Let (P, V) be a rational proof. If for every input x it holds that $R(x) = R$ and $C(x) = C$ for constants R and C, and the following inequality holds for every $\widetilde{P} \neq P$ and input $x \in \mathcal{D}$:

$$\frac{\tilde{R}(x)}{R} \leq \frac{\tilde{C}(x)}{C} + \epsilon$$

then (P, V) is $(KR\epsilon, K)$-sequentially composable for $\mathcal{D}$

Proof. It suffices to observe that, for any k inputs $x_1, ..., x_k$, the inequality above implies

$$\sum_{i=1}^{k} \tilde{R}(x_i) \leq R[\sum_{i=1}^{k}(\frac{\tilde{C}(x_i)}{C} + \epsilon)] \leq R + kR\epsilon$$

where the last inequality holds whenever $\sum_{i=1}^{k} \tilde{C}(x_i) \leq C$ as in Definition 3.2.1. □

Corollary 3.2.3. Let (P, V) and rew be respectively an interactive proof and a reward function as in Definition 2.2.1; 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 $\mathcal{D}$.

Proof. Observe that $\tilde{R}(x) = \tilde{p}_x \cdot R$ and then apply Lemma 3.2.2. □

3.2.3 Sequential Rational Proofs in the PCP model

We now describe a rational proof appeared in [5] and prove that is sequentially composable. The protocol assumes the existence of a trusted memory storage to which both Prover and Verifier have access, to realize the so-called ”PCP” (Probabilistically Checkable Proof) model. In this model, the Prover writes a very long proof of correctness, that the verifier checks only in a few randomly selected position. The trusted memory is needed to make sure that the prover is ”committed” to the proof before the verifier starts querying it.

The following protocol for proofs on a binary logical circuit $\mathcal{C}$ appeared in [5]. The Prover writes all the (alleged) values α_w for every wire in $w \in \mathcal{C}$, on the trusted memory location. The Verifier samples a single randome gate value to check its correctness and determines the reward accordingly:

1. The Prover writes the vector $\{\alpha_w\}_{w \in \mathcal{C}}$
2. The Verifier samples a random gate $g \in \mathcal{C}$.

- The Verifier reads $\alpha_{g_{out}}, \alpha_{g_L}, \alpha_{g_R}$, with g_{out}, g_L, g_R being respectively the output, left and right input wires of g; the verifier checks that $\alpha_{g_{out}} = g(\alpha_{g_L}, \alpha_{g_R})$;
- If g in an input gate the Verifier also checks that $\alpha_{g_L}, \alpha_{g_R}$ correspond to the correct input values;

The Verifier pays R if both checks are satisfied, otherwise it pays 0.

Theorem 3.2.4 ([5]). The protocol above is a rational proof for any boolean function in $P^{||NP}$, the class of all languages decidable by a polynomial time machine that can make non-adaptive queries to NP.

We will now show a cost model where the rational proof above is sequentially composable. We will assume that the cost for any prover is given by the number of gates he writes. Thus, for any input x, the costs for honest and dishonest provers are respectively $C(x) = S$, where $S = |\mathcal{C}|$, and $\tilde{C}(x) = \tilde{s}$ where $\tilde{s}$ is the number of gates written by the dishonest prover. Observe that in this model a dishonest prover may not write all the S gates, and that not all of the $\tilde{s}$ gates have to be correct. Let $\sigma \leq \tilde{s}$ the number of correct gates written by $\widetilde{P}$. Then

Theorem 3.2.5. In the cost model above the PCP protocol protocol in [5] is sequentially composable.

Proof. Observe that the probability $\tilde{p}_x$ that $\widetilde{P} \neq P$ earns R is such that

$$\tilde{p}_x = \frac{\sigma}{S} \leq \frac{\tilde{s}}{S} = \frac{\tilde{C}}{C}$$

Applying Corollary 3.2.3 completes the proof. □

The above cost model, basically says that the cost of writing down a gate dominates everything else, in particular the cost of *computing* that gate. In other cost models a proof of sequential composition may not be as straightforward. Assume, for example, that the honest prover pays \$1 to compute the value of a single gate while writing down that gate is "free". Now $\tilde{p}_x$ is still equal to $\frac{\sigma}{S}$ but to prove that this is smaller than $\frac{\tilde{C}}{C}$ we need some additional assumption that limits the ability for $\widetilde{P}$ to "guess" the right value of a gate without computing it (which we will discuss in the next Section).

3.2.4 Sequential Composition and the Unique Inner State Assumption

Definition 3.2.1 for sequential rational proofs requires a relationship between the reward earned by the prover and the amount of "work" the prover invested to produce that result. The intuition is that to produce the correct result, the prover must run the computation and incur its full cost. Unfortunately this intuition is difficult, if not downright impossible, to formalize. 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.

A way to circumvent this problem was suggested in [8] under the name of *Unique Inner State Assumption*: the idea is to assume a distribution $\mathcal{D}$ over the input space. When inputs x are chosen according to $\mathcal{D}$, then we assume that computing f requires cost C from any party: this can be formalized by saying that if a party invests $\tilde{C} = \gamma C$ 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).

Assumption 3.2.6. We say that the *(C,ϵ)-Unique Inner State Assumption* holds for a function f and a distribution $\mathcal{D}$ if for any algorithm $\widetilde{P}$ with cost $\tilde{C} = \gamma C$, the probability that on input $x \in \mathcal{D}$, $\widetilde{P}$ outputs $f(x)$ is at most $\gamma + (1-\gamma)\epsilon$.

Note that the assumption implicitly assumes a "large" output space for f (since a random guess of the output of f will be correct with probability 2^{-n} where n is the binary length of $f(x)$).

More importantly, note that Assumption 3.2.6 immediately yields our notion of sequential composability, if the Verifier can detect if the Prover is lying or not. Assume, as a mental experiment for now, that given input x, the Prover claims that $\tilde{y} = f(x)$ and the Verifier checks by recomputing $y = f(x)$ and paying a reward of R to the Prover if $y = \tilde{y}$ and 0 otherwise. Clearly this is not a very useful protocol, since the Verifier is not saving any computation effort by talking to the Prover. But it is sequentially composable according to our definition, since $\tilde{p}_x$, the probability that $\widetilde{P}$ collects R, is equal to the probability that $\widetilde{P}$ computes $f(x)$ correctly, and by using Assumption 3.2.6 we have that

$$\tilde{p}_x = \gamma + (1-\gamma)\epsilon \leq \frac{\tilde{C}}{C} + \epsilon$$

satisfying Corollary 3.2.3.

To make this a useful protocol we adopt a strategy from [8], which also uses this idea of verification by recomputing. Instead of checking every execution, we check only a random subset of them, and therefore we can amortize the Verifier's effort over a large number of computations. Fix a parameter m. The prover sends to the verifier the values $\tilde{y}_j$ which are claimed to be the result of computing f over m inputs $x_1, \ldots, x_m$. The verifier chooses one index i randomly between 1 and m, and computes $y_i = f(x_i)$. If $y_i = \tilde{y}_i$ the verifier pays R, otherwise it pays 0.

Let T be the total cost by the honest prover to compute m instances: cleary $T = mC$. Let $\tilde{T} = \Sigma_i \tilde{C}_i$ be the total effort invested by $\tilde{P}$, by investing $\tilde{C}_i$ on the computation of x_i. In order to satisfy Corollary 3.2.3 we need that $\tilde{p}_x$, the probability that $\widetilde{P}$ collects R, be less than $\tilde{T}/T + \epsilon$.

Let $\gamma_i = \tilde{C}_i/C$, then under Assumption 3.2.6 we have that $\tilde{y}_i$ is correct with probability at most $\gamma_i + (1-\gamma_i)\epsilon$. Therefore if we set $\gamma = \sum_i \gamma_i/m$ we have

$$\tilde{p}_x = \frac{1}{m}\sum_i [\gamma_i + (1-\gamma_i)\epsilon] = \gamma + (1-\gamma)\epsilon \leq \gamma + \epsilon$$

But note that $\gamma = \tilde{T}/T$ as desired since

$$\tilde{T} = \sum_i \tilde{C}_i = \sum_i \gamma_i C = T \sum_i \gamma_i / m$$

EFFICIENCY OF THE VERIFIER. If our notion of ”efficient Verifier” is a verifier who runs in time $o(C)$ where C is the time to compute f, then in the above protocol m must be sufficiently large to amortize the cost of computing one execution over many (in particular a constant – in the input size n – value of m would not work). In our ”concrete analysis” treatment, if we requests that the Verifier runs in time δC for an ”efficiency” parameter $\delta \leq 1$, then we need $m \geq \delta^{-1}$.

Therefore we are still in need of a protocol which has an efficient Verifier, and would still works for the ”stand-alone” case ($m = 1$) but also for the case of sequential composability over any number m of executions.

3.3 Our Protocol

We now present a protocol that works for functions $f : \{0, 1\}^n \to \{0, 1\}^n$ expressed by an arithmetic circuit $\mathcal{C}$ of size C and depth d and fan-in 2, given as a common input to both Prover and Verifier together with the input x.

Intuitively the idea is for the Prover to provide the Verifier with the output value y and its two ”children” y_L, y_R in the gate, i.e. the two input values of the last output gate G. The Verifier checks that $G(y_L, y_R) = y$, and then asks the Prover to verify that y_L or y_R (chosen a random) is correct, by recursing on the above test. The protocol description follows.

1. The Prover evaluates the circuit on x and send the output value y_1 to the Verifier.
2. **Repeat r times:** The Verifier identifies the root gate g_1 and then invokes $Round(1, g_1, y_1)$,

where the procedure $Round(i, g_i, y_i)$ is defined for $1 \leq i \leq d$ as follows:

1. The Prover sends the value of the input wires z_i^0 and z_i^1 of g_i to the Verifier.
2. The Verifiers performs the following
   - Check that y_i is the result of the operation of gate g_i on inputs z_i^0 and z_i^1. If not **STOP** and pay a reward of 0.
   - If $i = d$ (i.e. if the inputs to g_i are input wires), check that the values of z_i^0 and z_i^1 are equal to the corresponding bits of x. Pay reward R to Merlin if this is the case, nothing otherwise.
   - If $i < d$, choose a random bit b, send it to Merlin and invoke $Round(i+1, g_{i+1}^b, z_i^b)$ where g_{i+1}^b is the child gate of g_i whose output is z_i^b.

3.3.1 Efficiency

The protocol runs at most in d rounds. In each round, the Prover sends a constant number of bits representing the values of specific input and output wires; The Verifier sends at most one bit per round, the choice of the child gate. Thus the communication complexity is $O(d)$ bits.

The computation of the Verifier in each round is: (i) computing the result of a gate and checking for bit equality; (ii) sampling a child. Gate operations and equality are $O(1)$ per round. We assume our circuits are T-uniform, which allows the Verifier to select the correct gate in time $T(n)$ [2] Thus the Verifier runs in $O(rd \cdot T(n))$ with $r = O(\log C)$.

3.3.2 Proofs of (stand-alone) Rationality

Theorem 3.3.1. The protocol in Section 3.3 for $r = 1$ is a Rational Proof according to Def. 2.2.1.

We prove the above theorem by showing that for every input x the reward gap $\Delta(x)$ is positive.

[2] We point out that the Prover can provide the Verifier with the requested gate and then the Verifier can use the uniformity of the circuit to check that the Prover has given him the correct gate in time $O(T(n)))$ at each level in $O(T(n))$.

Proof. Let $\widetilde{P}$ a prover that always reports $\tilde{y} \neq y_1 = f(x)$ at Round 1.

Let us proceed by induction on the depth d of the circuit. If $d = 1$ then there is no possibility for $\widetilde{P}$ to cheat successfully, and its reward is 0.

Assume $d > 1$. We can think of the binary circuit $\mathcal{C}$ as composed by two subcircuits $\mathcal{C}_L$ and $\mathcal{C}_R$ and the output gate g_1 such that $f(x) = g_1(\mathcal{C}_L(x), \mathcal{C}_R(x))$. The respective depths d_L, d_R of these subcircuits are such that $0 \leq d_L, d_R \leq d-1$ and $max(d_L, d_R) = d - 1$. After sending $\tilde{y}$, the protocol requires that $\widetilde{P}$ sends output values for $C_L(x)$ and $C_R(x)$; let us denote these claimed values respectively with $\tilde{y}_L$ and $\tilde{y}_R$. Notice that at least one of these alleged values will be different from the respective correct subcircuit output: if it were otherwise, V would reject immediately as $g(\tilde{y}_L, \tilde{y}_R) = f(x) \neq \tilde{y}$. Thus at most one of the two values $\tilde{y}_L$, $\tilde{y}_R$ is equal to the output of the corresponding subcircuit. The probability that the $\widetilde{P}$ cheats successfully is:

$$\begin{aligned}
\Pr[\text{V accepts}] &\leq \frac{1}{2} \cdot (\Pr[\text{V accepts on } C_L] + \Pr[\text{V accepts on } C_R]) && (3.1)\\
&\leq \frac{1}{2} \cdot (1 - 2^{-\max(d_L, d_R)}) + \frac{1}{2} && (3.2)\\
&\leq \frac{1}{2} \cdot (1 - 2^{-d+1}) + \frac{1}{2} && (3.3)\\
&= 1 - 2^{-d} && (3.4)
\end{aligned}$$

At line 3.2 we used the inductive hypothesis and the fact that all probabilities are at most 1.

Therefore the expected reward of $\widetilde{P}$ is $\tilde{R} \leq R(1 - 2^{-d})$ and the reward gap is $\Delta(x) = 2^{-d}R$ (see Remark 5 below to explain the equality). □

The following useful corollary follows from the proof above.

Corollary 3.3.2. If the protocol described in Section 3.3 is repeated $r \geq 1$ times a prover can cheat with probability at most $(1 - 2^{-d})^r$.

Remark 5. *We point out that one can always build a prover strategy P^* which always answers incorrectly and achieves exactly the reward $R^* = R(1 - 2^{-d})$. This prover outputs an incorrect $\tilde{y}$*

and then computes one of the subcircuits that results in one of the input values (so that at least one of the inputs is correct). This will allow him to recursively answer with values z_i^0 and z_i^1 where one of the two is correct, and therefore be caught only with probability 2^{-d}.

Remark 6. *In order to have a non-negligible reward gap (see Remark 4) we need to limit ourselves to circuits of depth $d = O(\log n)$.*

3.3.3 Proof of Sequential Composability

General sufficient conditions for sequential composability

Lemma 3.3.3. Let $\mathcal{C}$ be a circuit of depth d. If the (C, ϵ) Unique Inner State Assumption (see Assumption 3.2.6) holds for the function f computed by $\mathcal{C}$, and input distribution $\mathcal{D}$, then the protocol presented above with r repetitions is a $k\epsilon R$-sequentially composable Rational Proof for $\mathcal{C}$ for $\mathcal{D}$ if the following inequality holds

$$(1 - 2^{-d})^r \leq \frac{1}{C}$$

Proof. Let $\gamma = \frac{\tilde{C}}{C}$. Consider $x \in \mathcal{D}$ and prover $\widetilde{P}$ which invests effort $\tilde{C} \leq C$. Under Assumption 3.2.6, $\widetilde{P}$ gives the correct outputs with probability $\gamma + \epsilon$ – assume that in this case $\widetilde{P}$ collects the reward R. If $\widetilde{P}$ gives an incorrect output we know (following Corollary 3.3.2) that he collects the reward R with probability at most $(1 - 2^{-d})^r$ which by hypothesis is less than γ. So either way we have that $\tilde{R} \leq (\gamma + \epsilon)R$ and therefore applying Lemma 3.2.2 concludes the proof. $\square$

The problem with the above Lemma is that it requires a large value of r for the result to be true resulting in an inefficient Verifier. In the following sections we discuss two approaches that will allows us to prove sequential composability even for an efficient Verifier:

- Limiting the class of provers we can handle in our security proof;
- Limiting the class of functions/circuits.

Limiting the strategy of the prover: Non-adaptive Provers

In proving sequential composability it is useful to find a connection between the amount of work done by a dishonest prover and its probability of cheating. The more a dishonest prover works, the higher its probability of cheating. This is true for our protocol, since the more "subcircuits" the prover computes correctly, the higher is probability of convincing the verifier of an incorrect output becomes. The question then is how can a prover who has an "effort budget" to spend, can maximize its probability of success in our protocol.

As we discussed in Remark 5, there is an *adaptive* strategy for the $\widetilde{P}$ to maximize its probability of success: compute one subcircuit correctly at every round of the protocol. We call this strategy "adaptive", because the prover allocates its "effort budget" $\tilde{C}$ on the fly during the execution of the rational proof. Conversely a *non-adaptive* prover $\widetilde{P}$ uses $\tilde{C}$ to compute some subcircuits in $\mathcal{C}$ before starting the protocol. 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).

Is it possible to limit the Prover to a non-adaptive strategy? 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 [22].

Therefore in the rest of this subsection 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.

Consider a prover $\widetilde{P}$ with effort budget $\tilde{C} < C$. A *DFS* (for "depth first search") prover uses its effort budget $\tilde{C}$ to compute a whole subcircuit of size $\tilde{C}$ and maximal depth d_{DFS}. Call this subcircuit $\mathcal{C}_{\mathcal{DFS}}$. $\widetilde{P}$ can answer correctly any verifier's query about a gate in $\mathcal{C}_{\mathcal{DFS}}$. During the interaction with V, the behavior of a DFS prover is as follows:

- At the beginning of the protocol send an arbitrary output value y_1.
- During procedure $Round(i, g_i, y_i)$:
  - If $g_i \in \mathcal{C}_{\mathcal{DFS}}$ then $\widetilde{P}$ sends the two correct inputs z_i^0 and z_i^1.
  - If $g_i \notin \mathcal{C}_{\mathcal{DFS}}$ and neither of g_i's input gate belongs to $\mathcal{C}_{\mathcal{DFS}}$ then $\widetilde{P}$ sends two arbitrary z_i^0 and z_i^1 that are consistent with y_i, i.e. $g_i(z_i^0, z_i^1) = y_i$.
  - $g_i \notin \mathcal{C}_{\mathcal{DFS}}$ and one of g_i's input gates belongs to $\mathcal{C}_{\mathcal{DFS}}$, then $\widetilde{P}$ will send the correct wire known to him and another arbitrary value consistent with y_i as above.

Lemma 3.3.4 (Advantage of a DFS prover)**.** In one repetition of the protocol above, a DFS prover with effort budget $\tilde{C}$ investment has probability of cheating $\tilde{p}_{DFS}$ defined by

$$\tilde{p}_{FS} \leq 1 - 2^{-d_{DFS}}$$

The proof of Lemma 3.3.4 follows easily from the proof of the stand-alone rationality of our protocol (see Theorem 3.3.1).

If a DFS prover focuses on maximizing the depth of a computed subcircuit given a certain investment, *BFS* provers allot their resources to compute all subcircuits rooted at a certain height. A BFS prover with effort budget $\tilde{C}$ computes the value of all gates up to the maximal height possible d_{BFS}. Note that d_{BFS} is a function of the circuit $\mathcal{C}$ and of the effort $\tilde{C}$. Let $\overline{\mathcal{C}}_{BFS}$ be the collection of gates computed by the BFS prover. The interaction of a BFS prover with V throughout the protocol resembles that of the DFS prover outlined above:

- At the beginning of the protocol send an arbitrary output value y_1.
- During procedure $Round(i, g_i, y_i)$:

- If $g_i \in \overline{\mathcal{C}}_{BFS}$ then $\widetilde{P}$ sends the two correct inputs z_i^0 and z_i^1.
- If $g_i \notin \overline{\mathcal{C}}_{BFS}$ and neither of g_i's input gate belongs to $\overline{\mathcal{C}}_{BFS}$ then $\widetilde{P}$ sends two arbitrary z_i^0 and z_i^1 that are consistent with y_i, i.e. $g_i(z_i^0, z_i^1) = y_i$.
- $g_i \notin \overline{\mathcal{C}}_{BFS}$ and both g_i's input gates belong to $\mathcal{C}_{\mathcal{DFS}}$, then $\widetilde{P}$ will send one of the correct wires known to him and another arbitrary value consistent with y_i as above.

As before, it is not hard to see that the probability of successful cheating by a BFS prover can be bounded as follows:

Lemma 3.3.5 (Advantage of a BFS prover)**.** In one repetition of the protocol above, a BFS prover with effort budget $\tilde{C}$ has probability of cheating $\tilde{p}_{BFS}$ bounded by

$$\tilde{p} \leq 1 - 2^{-d_{BFS}}$$

BFS and DFS provers are both special cases of the general non-adaptive strategy which allots its investment $\tilde{C}$ among a general collection of subcircuits $\overline{\mathcal{C}}_{NA}$. The interaction with V of such a prover is analogous to that of a a BFS/DFS prover but with a collection of computed subcircuits not constrained by any specific height. We now try to formally define what the success probability of such a prover is.

Definition 3.3.6 (Path experiment)**.** Consider a circuit $\mathcal{C}$ and a collection $\overline{\mathcal{C}}_{NA}$ of subcircuits of $\mathcal{C}$. Perform the following experiment: starting from the output gate, flip a biased coin and choose the ”left” subcircuit or the ”right” subcircuit at random with probability 1/2. Continue until the random path followed by the experiment reaches a computed gate in $\overline{\mathcal{C}}_{NA}$. Let i be the height of this gate, which is the output of the experiment. Define with Π_i the probability that this experiment outputs i.

The proof of the following Lemma is a generalization of the proof of security of our scheme. Once the ”verification path” chosen by the Verifier enters a fully computed subcircuit at height

i (which happens with probability $\Pi_i^{\overline{\mathcal{C}}_{NA}}$), the probability of success of the Prover is bounded by $(1 - 2^{-i})$

Lemma 3.3.7 (Advantage of a non adaptive prover)**.** In one repetition of the protocol above, a generic prover with effort budget $\tilde{C}$ used to compute a collection $\overline{\mathcal{C}}_{NA}$ of subcircuits, has probability of cheating $\tilde{p}_{\overline{\mathcal{C}}_{NA}}$ bounded by

$$\tilde{p}_{\overline{\mathcal{C}}_{NA}} \leq \sum_{i=0}^{d} \Pi_i (1 - 2^{-i})$$

where Π_i-s are defined as in Definition 3.3.6.

Limiting the class of functions: Regular Circuits

Lemma 3.3.7 still does not produce a clear bound on the probability of success of a cheating prover. The reason is that it is not obvious how to bound the probabilities $\Pi_i^{\overline{\mathcal{C}}_{NA}}$ that arise from the computed subcircuits $\overline{\mathcal{C}}_{NA}$ since those depends in non-trivial ways from the topology of the circuit $\mathcal{C}$.

We now present a type of circuits for which it can be shown that the BFS strategy is optimal. The restriction on the circuit is surprisingly simple: we call them regular circuits. In the next section we show examples of interesting functions that admit regular circuits.

Definition 3.3.8 (Regular circuit)**.** A circuit $\mathcal{C}$ is said to be regular if the following conditions hold:

- $\mathcal{C}$ is layered;
- every gate has fan-in 2;
- the inputs of every gate are the outputs of two distinct gates.

The following lemma states that, in regular circuits, we can bound the advantage of any prover investing $\tilde{C}$ by looking at the advantage of a BFS prover with the same investment.

Lemma 3.3.9 (A bound for provers' advantage in regular circuits)**.** Let $\widetilde{P}$ be a prover investing $\tilde{C}$. Let $\mathcal{C}$ be the circuit being computed and $\delta = d_{BFS}(\mathcal{C}, \tilde{C})$. In one repetition of the protocol above, the advantage of $\widetilde{P}$ is bounded by

$$\tilde{p} \leq \tilde{p}_{BFS} = 1 - 2^{-\delta}$$

Proof. Let $\overline{\mathcal{C}}_{NA}$ be the family of subcircuits compued by $\widetilde{P}$ with effort $\tilde{C}$. As pointed out above the probability of success for $\widetilde{P}$ is

$$\tilde{p} \leq \sum_{i=0}^{d} \Pi_i^{\overline{\mathcal{C}}_{NA}} (1 - 2^{-i})$$

Consider now a prover $\widetilde{P}'$ which uses $\tilde{C}$ effort to compute a different collection of subcircuits $\overline{\mathcal{C}}'_{NA}$ defined as follows:

- Remove a gate from a subcircuit of height j in $\overline{\mathcal{C}}_{NA}$: this produces two subcircuits of height $j-1$. This is true because of the regularity of the circuit: since the inputs of every gate are the outputs of two distinct gates, when removing a gate of height j this will produce two subcircuits of height $j-1$;

- Use that computation to "join" two subcircuits of height k into a single subcircuit of height $k+1$. Again we are using the regularity of the circuit here: since the circuit is layered, the only way to join two subcircuits into a single computed subcircuit is to take two subcircuits of the same height.

What happens to the probability $\tilde{p}'$ of success of $\widetilde{P}'$? Let ℓ be the number of possible paths generated by the experiment above with $\overline{\mathcal{C}}_{NA}$. Then the probability of entering a computed subcircuit at height j decreases by $1/\ell$ and that probability weight goes to entering at height $j-1$. Similarly the probability of entering at height k goes down by $2/\ell$ and that probability weight is shifted to entering at height $k+1$. Therefore

$$\tilde{p}' \leq \sum_{i \neq j, j-1, k, k+1} \Pi_i (1 - 2^{-i}) +$$

$$+(\Pi_j - \frac{1}{\ell})(1 - 2^{-j}) + (\Pi_{j-1} + \frac{1}{\ell})(1 - 2^{-j+1}) +$$

$$+(\Pi_k - \frac{2}{\ell})(1 - 2^{-k}) + (\Pi_{k+1} + \frac{2}{\ell})(1 - 2^{-k-1}) =$$

$$= \tilde{p} + \frac{1}{\ell 2^j} - \frac{1}{\ell 2^{j-1}} + \frac{1}{\ell 2^{k-1}} - \frac{1}{\ell 2^k}$$

$$= \tilde{p} + \frac{2^k - 2^{k+1} + 2^{j+1} - 2^j}{\ell 2^{j+k}} = \tilde{p} + \frac{2^j - 2^k}{\ell 2^{j+k}}$$

Note that $\tilde{p}'$ increases if $j > k$ which means that it's better to take "computation" away from tall computed subcircuits to make them shorter, and use the saved computation to increase the height of shorter computed subtrees, and therefore that the probability is maximized when all the subtrees are of the same height, i.e. by the BFS strategy which has probability of success $\tilde{p}_{BFS} = 1 - 2^{-\delta}$. □

The above Lemma, therefore, yields the following

Theorem 3.3.10. Let $\mathcal{C}$ be a regular circuit of size C. If the (C, ϵ) Unique Inner State Assumption (see Assumption 3.2.6) holds for the function f computed by $\mathcal{C}$, and input distribution $\mathcal{D}$, then the protocol presented above with r repetitions is a $k\epsilon R$-sequentially composable Rational Proof for $\mathcal{C}$ for $\mathcal{D}$ if the prover follows a non-adaptive strategy and the following inequality holds for all $\tilde{C}$

$$(1 - 2^{-\delta})^r \leq \frac{\tilde{C}}{C}$$

where $\delta = d_{BFS}(\mathcal{C}, \tilde{C})$.

Proof. Let $\gamma = \frac{\tilde{C}}{C}$. Consider $x \in \mathcal{D}$ and prover $\widetilde{P}$ which invests effort $\tilde{C} \leq C$. Under Assumption 3.2.6, $\widetilde{P}$ gives the correct outputs with probability $\gamma + \epsilon$ – assume that in this case $\widetilde{P}$ collects the reward R.

If $\widetilde{P}$ gives an incorrect output we can invoke Lemma 3.3.9 and conclude that he collects reward R with probability at most $(1 - 2^{-\delta})^r$ which by hypothesis is less than γ. So either way we have that $\tilde{R} \leq (\gamma + \epsilon)R$ and therefore applying Lemma 3.2.2 concludes the proof. □

3.4 Results for FFT circuits

In this section we apply the previous results to the problem of computing FFT circuits, and by extension to polynomial evaluations.

3.4.1 FFT circuit for computing a single coefficient

The Fast Fourier Transform is an almost ubiquitous computational problem that appears in many applications, including many of the volunteer computations that motivated our work. As described in [17] a circuit to compute the FFT of a vector of n input elements, consists of $\log n$ levels, where each level comprises $n/2$ *butterflies* gates. The output of the circuit is also a vector of n input elements.

Let us focus on the circuit that computes a single element of the output vector: it has $\log n$ levels, and at level i it has $n/2^i$ butterflies gates. Moreover the circuit is regular, according to Definition 3.3.8.

Theorem 3.4.1. Under the (C, ϵ)-unique inner state assumption for input distribution $\mathcal{D}$, the protocol in Section 3.3, when repeated $r = O(1)$ times, yields sequentially composable rational proofs for the FFT, under input distribution $\mathcal{D}$ and assuming non-adaptive prover strategies.

Proof. Since the circuit is regular we can prove sequential composability by invoking Theorem 3.3.10

and proving that for $r = O(1)$, the following inequality holds

$$\tilde{p} = (1 - 2^{-\delta})^r \leq \frac{\tilde{C}}{C}$$

where $\delta = d_{BFS}(\mathcal{C}, \tilde{C})$.

But for any $\tilde{\delta} < d$, the structure of the FFT circuit inmplies that the number of gates below height $\tilde{\delta}$ is $\tilde{C}_{\tilde{\delta}} = \Theta(C(1 - 2^{-\tilde{\delta}}))$. Thus the inequality above can be satisfied with $r = \Theta(1)$. □

3.4.2 Mixed Strategies for Verification

One of the typical uses of the FFT is to change representation for polynomials. Given a polynomial $P(x)$ of degree $n-1$ we can represent it as a vector of n coefficients $[a_0, \dots, a_{n-1}]$ or as a vector of n points $[P(\omega_0), \dots, P(\omega_{n-1})]$. If ω_i are the complext n-root of unity, the FFT is the algorithm that goes from one representation to the other in $O(n \log n)$ time, rather than the obvious $O(n^2)$.

In this section we consider the following problem: given two polynomial P, Q of degree $n-1$ in point representation, compute the inner product of the coefficients of P, Q. A fan-in two circuit computing this function could be built as follows:

- two parallel FFT subcircuits computing the coefficient representation of P, Q ($\log n$-depth and $n \log n$) size total for the 2 circuits) ;
- a subcircuit where at the first level the i-degree coefficients are multiplied with each other, and then all these products are added by a binary tree of additions $O(\log n)$-depth and $O(n)$ size);

Note that this circuit is regular, and has depth $2\log n + 1$ and size $n \log n + n + 1$

Consider a prover $\widetilde{P}$ who pays $\tilde{C} < n \log n$ effort. Then, since the BFS strategy is optimal, the probability of convincing the Verifier of a wrong result of the FFT is $(1 - 2^{-\tilde{d}})^r$ where $\tilde{d} = c \log n$

with $c \leq 1$. Note also that $\frac{\tilde{C}}{C} < 1$. Therefore with $r = O(n^c)$ repetitions, the probability of success can be made smaller than $\frac{\tilde{C}}{C}$. The Verifier's complexity is $O(n^c \log n) = o(n \log n)$.

If $\tilde{C} \geq n \log n$ the analysis above fails since $\tilde{d} > \log n$. Here we observe that in order for $\widetilde{P}$ to earn a larger reward than P, it must be that P has run at least $k = O(\log n)$ executions (since it is possible to find $k+1$ inputs such that $(k+1)\tilde{C} \leq kC$ only if $k > \log n$). In this case we can use a "mixed" strategy for verification:

- The Verifier pays the Prover only after k executions. Each execution is verified as above (with n^c repetitions);
- Additionally the Verifier uses the "check by re-execution" (from Section 3.2.4) strategy every k executions (verifiying one execution by recomputing it);
- The Verifier pays R if all the checks are satisfied, 0 otherwise;
- The Verifier's complexity is $O(kn^c \log n + n \log n) = o(kn \log n)$ – the latter being the complexity of computing k instances.

3.5 Sequential Composability for Space Bounded Computation

The intuition behind our definition and Lemma 3.2.3 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 [8] 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 [8] 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 [14] we formalized the notion of Sequential Composability in Definition 3.2.1 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.

3.5.1 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 3.5.1 (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 3.5.1 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 [14] 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 3.5.1 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 2.3.1 tight.).

Is it possible to limit the Prover to a non-adaptive strategy? As pointed out in [14] 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 [22]. 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 [31]) 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 Merke-tree committment, 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 3.5.2. Let $L \in \mathsf{DTISP}[\mathsf{poly}(n), S(n)]$ and M be a Turing Machine recognizing L. Assume that Assumption 3.5.1 holds for M, under input distribution $\mathcal{D}$ and parameter ϵ. Moreover assume the prover follows a non-adaptive strategy. Then the protocol of Section 2.3 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 3.5.1 we know that the probability that $\widetilde{P}$ can compute more than t correct transitions is ϵ, therefore an upperbound on $\tilde{p}_x$ is $\frac{t}{T} + \epsilon$ and the Theorem follows from Corollary 3.2.3. $\square$

Chapter 4

Fine-Grained Verifiable Computation

4.1 Preliminaries

In this chapter, all arithmetic computations (such as sums, inner product, matrix products, etc.) in this work will be over GF(2) unless specified otherwise.

Definition 4.1.1 (Function Family)**.** A *function family* is a family of (possibly randomized) functions $F = \{f_\lambda\}_{\lambda\in\mathbb{N}}$, where for each λ, f_λ has domain D^f_λ and co-domain R^f_λ. A *class* $\mathcal{C}$ is a collection of function families.

In most of our constructions $D^f_\lambda = \{0,1\}^{d^f_\lambda}$ and $R^f_\lambda = \{0,1\}^{r^f_\lambda}$ for sequences $\{d^f_\lambda\}_\lambda$, $\{d^f_\lambda\}_\lambda$.

In the rest of the paper we will focus on the class of $\mathcal{C} = \mathsf{NC}^1$ of functions for which there is a polynomial $p(\cdot)$ and a constant c such that for each λ, the function f_λ can be computed by a Boolean (randomized) fan-in 2, circuit of size $p(\lambda)$ and depth $c\log(\lambda)$. In the formal statements of our results we will also use the following classes: AC^0, the class of functions of polynomial size and constant depth with $\mathsf{AND}, \mathsf{OR}$ and NOT gates with unbounded fan-in; $\mathsf{AC}^0[2]$, the class of functions of polynomial size and constant depth with $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ and PARITY gates with unbounded fan-in. For a gate g we denote by $\mathsf{type}_C(g)$ the type of the gate g in the circuit C and by $\mathsf{parents}_C(g)$ the list of gates of C whose output is an input to C (such list may potentially contain duplicates).

Given a function f, we can think of its *multiplicative depth* as the degree of the lowest-degree polynomial in GF(2) that evaluates to f. Similarly, we define the multiplicative depth of a circuit as follows:

Definition 4.1.2 (Multiplicative Depth)**.** Let C be a circuit, we define the multiplicative depth of C as $\mathsf{md}(g_{out})$ where g_{out} is its output gate and the function md, from the set of gates to the set of natural numbers is recursively defined as follows:

$$\mathsf{md}(g) := \begin{cases} 1 & \text{if } \mathsf{type}_C(g) = \mathsf{input} \\ \max\{\mathsf{md}(g') : g' \in \mathsf{parents}_C(g)\} & \text{if } \mathsf{type}_C(g) = \mathsf{XOR} \\ \sum_{g' \in \mathsf{parents}_C(g)} \mathsf{md}(g') & \text{if } \mathsf{type}_C(g) \in \{\mathsf{AND}, \mathsf{OR}\} \end{cases}$$

The following two circuit classes will appear in several of our results.

Definition 4.1.3 (Circuits with Constant Multiplicative Depth)**.** We denote by $\mathsf{AC}^0_{\mathrm{CM}}[2]$ the class of circuits in $\mathsf{AC}^0[2]$ with *constant multiplicative depth*.

Definition 4.1.4 (Circuits with Quasi-Constant Multiplicative Depth)**.** For a circuit C we denote by $S_{\omega(1)}(C)$ the set of AND and OR gates in C with non-constant fan-in. We say that C has *quasi-constant multiplicative depth* if $|S_{\omega(1)}(C)| = O(1)$. We shall denote by $\mathsf{AC}^0_{\mathrm{Q}}[2]$ the class of circuits in $\mathsf{AC}^0[2]$ with quasi-constant multiplicative depth.

LIMITED ADVERSARIES. We define adversaries also as families of randomized algorithms $\{A_\lambda\}_\lambda$, one for each security parameter (note that this is a non-uniform notion of security). We denote the class of adversaries we consider as $\mathcal{A}$, and in the rest of the paper we will also restrict $\mathcal{A}$ to NC^1.

INFINITELY-OFTEN SECURITY. We now move to define security against all adversaries $\{A_\lambda\}_\lambda$ that belong to a class $\mathcal{A}$.

Our results achieve an ”infinitely often” notion of security, which states that for all adversaries outside of our permitted class $\mathcal{A}$ our security property holds infinitely often (i.e. for an infinite sequence of security parameters rather than for every sufficiently large security parameter. We inherit this limitation from the techniques of [20].

Definition 4.1.5 (Infinitely-Often Computational Indistinguishability)**.** Let $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ Let $\mathcal{Y} = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles over the same domain family, $\mathcal{A}$ a class of adversaries, and Λ an infinite subset of $\mathbb{N}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are infinitely often computational indistinguishable with respect to set Λ and the class $\mathcal{A}$, denoted by $\mathcal{X} \sim_{\Lambda,\mathcal{A}} \mathcal{Y}$ if there exists a negligible function ν such that for any $\lambda \in \Lambda$ and for any adversary $A = \{A_\lambda\}_\lambda \in \mathcal{A}$

$$|\Pr[A_\lambda(X_\lambda) = 1] - \Pr[A_\lambda(Y_\lambda) = 1]| < \nu(\lambda)$$

When $\mathcal{A} = \mathsf{NC}^1$ we will keep it implicit and use the notation $\mathcal{X} \sim_\Lambda \mathcal{Y}$ and say that $\mathcal{X}$ and $\mathcal{Y}$ are Λ-computationally indistinguishable.

In our proofs we will use the following facts on infinitely-often computationally indistinguishable ensembles. We skip their proof as, except for a few technicalities, it is analogous to the corresponding properties for standard computational indistinguishability[1].

Lemma 4.1.6 (Facts on Λ-Computational Indistinguishability).

- **Transitivity:** Let $m = \mathsf{poly}(\lambda)$ and $\mathcal{X}^{(j)}$ with $j \in \{0, \ldots, m\}$ be ensembles. If for all $j \in [m]$ $\mathcal{X}^{(j-1)} \sim_\Lambda \mathcal{X}^{(j)}$, then $\mathcal{X}^{(0)} \sim_\Lambda \mathcal{X}^{(m)}$.

- **Weaker than statistical indistinguishability:** Let $\mathcal{X}, \mathcal{Y}$ be statistically indistinguishable ensembles. Then $\mathcal{X} \sim_\Lambda \mathcal{Y}$ for any infinite $\Lambda \subseteq \mathbb{N}$

- **Closure under NC^1:** Let $\mathcal{X}, \mathcal{Y}$ be ensembles and $\{f_\lambda\}_{\lambda\in\mathbb{N}} \in \mathsf{NC}^1$. If $\mathcal{X} \sim_\Lambda \mathcal{Y}$ for some Λ then $f_\lambda(\mathcal{X}) \sim_\Lambda f_\lambda(\mathcal{Y})$.

4.1.1 Public-Key Encryption

A public-key encryption scheme

$\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is a triple of algorithms which operate as follow:

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

[1] We refer the reader to [25].

- **Encryption.** The algorithm $c \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}}(\mu)$ takes the public key pk and a single bit message $\mu \in \{0,1\}$ and outputs a ciphertext c. The notation $\mathsf{PKE.Enc}_{\mathsf{pk}}(\mu; r)$ will be used to represent the encryption of a bit μ using randomness r.

- **Decryption.** The algorithm $\mu^* \leftarrow \mathsf{PKE.Dec}_{\mathsf{sk}}(c)$ takes the secret key sk and a ciphertext c and outputs a message $\mu^* \in \{0,1\}$.

Obviously we require that $\mu = \mathsf{PKE.Dec}_{\mathsf{sk}}(\mathsf{PKE.Enc}_{\mathsf{pk}}(\mu))$

Definition 4.1.7 (CPA Security for PKE)**.** A scheme PKE is IND-CPA secure if for an infinite $\Lambda \subseteq \mathbb{N}$ we have

$$(\mathsf{pk}, \mathsf{PKE.Enc}_{\mathsf{pk}}(0)) \sim_{\Lambda} (\mathsf{pk}, \mathsf{PKE.Enc}_{\mathsf{pk}}(1))$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Keygen}(1^{\lambda})$.

Remark 7 (Security for Multiple Messages)**.** Notice that by a standard hybrid argument and Lemma 4.1.6 we can prove that any scheme secure according to Definition 4.1.7 is also secure for multiple messages (i.e. the two sequences of encryptions bit by bit of two bit strings are computationally indistinguishable). We will use this fact in the proofs in Section 4.3, but we do not provide the formal definition for this type of security. We refer the reader to 5.4.2 in [26].

Somewhat Homomorphic Encryption

A public-key encryption scheme is said to be homomorphic if there is an additional algorithm Eval which takes a input the public key pk, the representation of a function $f : \{0,1\}^l \to \{0,1\}$ and a set of l ciphertexts $c_1, \ldots, c_l$, and outputs a ciphertext c_f[2].

We proceed to define the homomorphism property. The next notion of $\mathcal{C}$-homomorphism is sometimes also referred to as "somewhat homomorphism".

[2]Notice that the syntax of Eval can also be extended to return a sequence of encryptions for the case of multi-output functions. We will use this fact in Section 4.2.4. See also Remark 7.

Definition 4.1.8 ($\mathcal{C}$-homomorphism)**.** Let $\mathcal{C}$ be a class of functions (together with their respective representations). An encryption scheme PKE is $\mathcal{C}$-homomorphic (or, homomorphic for the class $\mathcal{C}$) if for every function f_λ where $f_\lambda \in \mathcal{F}\{f_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{C}$ and respective inputs $\mu_1, \ldots, \mu_l \in \{0,1\}$ (where $l = l(\lambda)$), it holds that if $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$ and $c_i \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}}(\mu_i)$ then

$$\Pr[\mathsf{PKE.Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(F, c_1, \ldots, c_l)) \neq F(\mu_1, \ldots, \mu_l)] = \mathsf{neg}(\lambda),$$

As usual we require the scheme to be non-trivial by requiring that the output of Eval is compact:

Definition 4.1.9 (Compactness). A homomorphic encryption scheme PKE is compact if there exists a polynomial s in λ such that the output length of Eval is at most $s(\lambda)$ bits long (regardless of the function f being computed or the number of inputs).

Definition 4.1.10. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ of arithmetic circuits in GF(2). A scheme PKE is leveled $\mathcal{C}$-homomorphic if it takes 1^L as additional input in key generation, and can only evaluate depth-L arithmetic circuits from $\mathcal{C}$. The bound $s(\lambda)$ on the ciphertext must remain independent of L.

4.1.2 Verifiable Computation

In a *Verifiable Computation* scheme a Client uses an untrusted server to compute a function f over an input x. The goal is to prevent the Client from accepting an incorrect value $y' \neq f(x)$. We require that the Client's cost of running this protocol be smaller than the cost of computing the function on his own. The following definition is from [23] which allows the client to run a possibly expensive pre-processing step.

Definition 4.1.11 (Verifiable Computation Scheme)**.** A *verifiable computation scheme* $\mathcal{VC} = (\mathsf{VC.KeyGen}, \mathsf{VC.Pro}$ consists of the four algorithms defined below.

1. $\mathsf{VC.KeyGen}(f, 1^\lambda) \rightarrow (\mathsf{pk_W}, \mathsf{sk_D})$: Based on the security parameter λ, the randomized *key generation* algorithm generates a public key that encodes the target function f, which is used

by the Server to compute f. It also computes a matching secret key, which is kept private by the Client.

2. $\mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x) \rightarrow (q_x, s_x)$: The *problem generation* algorithm uses the secret key $\mathsf{sk_D}$ to encode the function input x as a public query q_x which is given to the Server to compute with, and a secret value s_x which is kept private by the Client.

3. $\mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x) \rightarrow a_x$: Using the Client's public key and the encoded input, the Server *computes* an encoded version of the function's output $y = F(x)$.

4. $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x) \rightarrow y \cup \{\perp\}$: Using the secret key $\mathsf{sk_D}$ and the secret "decoding" s_x, the *verification* algorithm converts the worker's encoded output into the output of the function, e.g., $y = f(x)$ or outputs $\perp$ indicating that a_x does not represent the valid output of f on x.

The scheme should be complete, i.e. an honest Server should (almost) always return the correct value.

Definition 4.1.12 (Completeness)**.** A delegation scheme $\mathcal{VC} = (\mathsf{VC.KeyGen}, \mathsf{VC.ProbGen}, \mathsf{VC.Compute}, \mathsf{VC.Verif}$ has *overwhelming completeness* for a class of functions $\mathcal{C}$ if there is a function $\nu(n) = \mathsf{neg}(\lambda)$ such that for infinitely many values of λ, if $f_\lambda \in \mathcal{F} \in \mathcal{C}$, then for all inputs x the following holds with probability at least $1 - \nu(n)$: $(\mathsf{pk_W}, \mathsf{sk_D}) \leftarrow \mathsf{VC.KeyGen}(f_\lambda, \lambda)$ $(q_x, s_x) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x)$ and $a_x \leftarrow \mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x)$ then $y = f_\lambda(x) \leftarrow \mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$.

To define soundness we consider an adversary who plays the role of a malicious Server who tries to convince the Client of an incorrect output $y \neq f(x)$. The adversary is allowed to run the protocol on inputs of her choice, i.e. see the *queries* q_{x_i} for adversarially chosen x_i's before picking an input x and attempt to cheat on that input. Because we are interested in the parallel complexity of the adversary we distinguish between two parameters l and m. The adversary is allowed to do l rounds of adaptive queries, and in each round she queries m inputs. Jumping ahead, because our adversaries are restricted to NC^1 circuits, we will have to bound l with a constant, but we will be able to keep m polynomially large.

```
Experiment $\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, l, m]$
    $(\mathsf{pk_W}, \mathsf{sk_D}) \leftarrow \mathsf{VC.KeyGen}(f, \lambda)$;
    $\mathcal{I} \leftarrow \emptyset$;
    For $i = 1, \ldots, i = l$;
        $\{x_{(i-1)m}, \ldots x_{im-1}\} \leftarrow A_\lambda(\mathsf{pk_W}, \mathcal{I})$;
        $\{(q_j, s_j) : (q_j, s_j) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x_j), j \in \{(i-1)m, \ldots, im\}\}$
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{x_{(i-1)m}, \ldots x_{im-1}\} \cup \{q_{(i-1)m}, \ldots q_{im-1}\}$;
    $\hat{a} \leftarrow A_\lambda(\mathsf{pk_W}, \mathcal{I})$;
    $\hat{y} \leftarrow \mathsf{VC.Verify}_{\mathsf{sk_D}}(s_{lm}, \hat{a})$
    If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x_{lm})$, output 1, else 0.
```

Remark 8. In the experiment above the adversary "tries to cheat" on the last input presented in the last round of queries (i.e. $x_l m$). This is without loss of generality. In fact, assume the adversary aimed at cheating on an input presented before round l, then with one additional round it could present that same input once more as the last of the batch in that round.

Definition 4.1.13 (Soundness)**.** We say that a verifiable computation scheme is (l, m)-sound against a class $\mathcal{A}$ of adversaries if there exists a negligible function $\mathsf{neg}(\lambda)$, such that for all $A = \{A_\lambda\}_\lambda \in \mathcal{A}$, and for infinitely many λ we have that

$$\Pr[\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, l, m] = 1] \leq \mathsf{neg}(\lambda)$$

Assume the function f we are trying to compute belongs to a class $\mathcal{C}$ which is smaller than $\mathcal{A}$. Then our definition guarantees that the "cost" of cheating is higher than the cost of honestly computing f and engaging in the Verifiable Computation protocol $\mathcal{VC}$. Jumping ahead, our scheme will allow us to compute the class $\mathcal{C} = \mathsf{AC}^0[2]$ against the class of adversaries $\mathcal{A} = \mathsf{NC}^1$.

EFFICIENCY The last thing to consider is the efficiency of a VC protocol. Here we focus on the time complexity of computing the function f. Let n be the number of input bits, and m be the number of output bits, and S be the size of the circuit computing f.

- A verifiable computation scheme $\mathcal{VC}$ is **client-efficient** if circuit sizes of $\mathsf{VC.ProbGen}$ and $\mathsf{VC.Verify}$ are $o(S)$. We say that it is **linear-client** if those sizes are $O(\mathsf{poly}(\lambda)(n+m))$.
- A verifiable computation scheme $\mathcal{VC}$ is **server-efficient** if the circuit size of $\mathsf{VC.Compute}$ is $O(\mathsf{poly}(\lambda)S)$.

We note that the key generation protocol $\mathsf{VC.KeyGen}$ can be expensive, and indeed in our protocol (as in [23, 16, 2]) its cost is the same as computing f – this is OK as $\mathsf{VC.KeyGen}$ is only invoked once per function, and the cost can be amortized over several computations of f.

4.2 Fine-Grained SHE

We start by recalling the public key encryption from [20] which is secure against adversaries in NC^1.

The scheme is described in Figure 4.1. Its security relies on the following result, implicit in [38][3]. We will also use this lemma when proving the security of our construction in Section 4.2.

Lemma 4.2.1 ([38])**.** If $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L/poly}$ then there exist distribution $\mathcal{D}_\lambda^{\mathrm{kg}}$ over $\{0,1\}^{\lambda\times\lambda}$, distribution $\mathcal{D}_\lambda^{f}$ over matrices in $\{0,1\}^{\lambda\times\lambda}$ *of full rank*, and infinite set $\Lambda \subseteq \mathbb{N}$ such that

$$\mathbf{M}^{\mathrm{kg}} \sim_\Lambda \mathbf{M}^{\mathrm{f}}$$

where $\mathbf{M}^{\mathrm{f}} \leftarrow \mathcal{D}_\lambda^{f}$ and $\mathbf{M}^{\mathrm{kg}} \leftarrow \mathcal{D}_\lambda^{\mathrm{kg}}$.

The following result is central to the correctness of the scheme PKE in Figure 4.1 and is implicit in [20].

[3]Stated as Lemma 4.3 in [20].

- $\mathsf{PKE.Keygen}_{\mathsf{sk}}(1^\lambda)$:
  1. Sample $(\mathbf{M}, \mathbf{k}) \leftarrow \mathsf{KSample}(1^\lambda)$;
  2. Output $(\mathsf{pk} = \mathbf{M}, \mathsf{sk} = \mathbf{k})$.
- $\mathsf{PKE.Enc}_{\mathsf{pk}=\mathbf{M}}(\mu))$:
  1. Sample $\mathbf{r} \leftarrow_{\$} \{0,1\}^\lambda$;
  2. Let $t^\intercal = (0\ \ldots 0\ 1) \in \{0,1\}^\lambda$;
  3. Output $\mathbf{c}^\intercal = \mathbf{r}^\intercal \mathbf{M} + \mu \mathbf{t}^\intercal$.
- $\mathsf{PKE.Dec}_{\mathsf{sk}=\mathbf{k}}(\mathbf{c})$:
  1. Output $\langle \mathbf{k}\, , \mathbf{c} \rangle$

Figure 4.1: PKE construction [20]

Lemma 4.2.2 ([20])**.** There exists sampling algorithm $\mathsf{KSample}$ such that $(\mathbf{M}, \mathbf{k}) \leftarrow \mathsf{KSample}(1^\lambda)$, $\mathbf{M}$ is a matrix distributed according to $\mathcal{D}_\lambda^{\mathrm{kg}}$ (as in Lemma 4.2.1), $\mathbf{k}$ is a vector in the kernel of $\mathbf{M}$ and has the form
$\mathbf{k} = (r_1,\, r_2,\, \ldots,\, r_{\lambda-1},\, 1) \in \{0,1\}^\lambda$ where r_i-s are uniformly distributed bits.

Theorem 4.2.3 ([20])**.** Assume $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$. Then, the scheme $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ defined in Figure 4.1 is a Public Key Encryption scheme secure against NC^1 adversaries. All algorithms in the scheme are computable in $\mathsf{AC}^0[2]$.

4.2.1 Leveled Homomorphic Encryption for $\mathsf{AC}^0_{\mathbf{CM}}[2]$ Functions Secure against NC^1

We denote by $\mathbf{x}[i]$ the i-th bit of a vector of bits $\mathbf{x}$. Below, the scheme $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is the one defined in Figure 4.1.

Our SHE scheme is defined by the following four algorithms:

- $\mathsf{HE.Keygen}_{\mathsf{sk}}(1^\lambda, L)$: For key generation, sample $L+1$ key pairs $(\mathbf{M}_0, \mathbf{k}_0), \ldots, \mathbf{M}_0, \mathbf{k}_L) \leftarrow$ $\mathsf{PKE.Keygen}(1^\lambda)$, and compute, for all $\ell \in \{0, \ldots, L-1\}$, $i, j \in [\lambda]$, the value

$$\mathbf{a}_{\ell,i,j} \leftarrow \mathsf{PKE.Enc}_{\mathbf{M}_{\ell+1}}(\mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \in \{0,1\}^\lambda$$

We define $\mathbf{A} := \{a_{\ell,i,j}\}_{\ell,i,j}$ to be the set of all these values. t then outputs the secret key $\mathsf{sk} = \mathbf{k}_L$, and the public key $\mathsf{pk} = (\mathbf{M}_0, \mathbf{A})$. In the following we call $\mathsf{evk} = \mathbf{A}$ the evaluation key.

We point out a property that will be useful later: by the definition above, for all $\ell \in \{0, \ldots, L-1\}$ we have

$$\langle \mathbf{k}_{\ell+1}\,, \mathbf{a}_{\ell+1,i,j} \rangle = \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]\,. \tag{4.1}$$

- $\mathsf{HE.Enc}_{\mathsf{pk}}(\mu))$: Recall that $\mathsf{pk} = \mathbf{M}_0$. To encrypt a message μ we compute $\mathbf{v} \leftarrow \mathsf{PKE.Enc}_{\mathbf{M}_0}(\mu)$. The output ciphertext contains $\mathbf{v}$ in addition to a "level tag", an index in $\{0, \ldots, L\}$ denoting the "multiplicative depth" of the generated ciphertext. The encryption algorithm outputs $c := (\mathbf{v}, 0)$.

- $\mathsf{HE.Dec}_{\mathbf{k}_L}(c)$: To decrypt a ciphertext[4] $c = (\mathbf{v}, L)$ compute $\mathsf{PKE.Dec}_{\mathbf{k}_L}(\mathbf{v})$, i.e.

$$\langle \mathbf{k}_L\,, \mathbf{v} \rangle$$

- $\mathsf{HE.Eval}_{\mathsf{evk}}(f, c_1, \ldots, c_t)$: where $F : \{0,1\}^t \to \{0,1\}$: We require that f is represented as an arithmetic circuit in GF(2) with addition gates of unbounded fan-in and multiplication gates of fan-in 2. We also require the circuit to be *layered*, i.e. the set of gates can be partitioned in subsets (layers) such that wires are always between adjacent layers. Each layer should be

[4]We are only requiring to decrypt ciphertexts that are output by $\mathsf{HE.Eval}(\cdots)$

composed homogeneously either of addition or multiplication gates. Finally, we require that the number of multiplications layers (i.e. the multiplicative depth) of f is L.

We homomorphically evaluate f gate by gate. We will show how to perform multiplication (resp. addition) of two (resp. many) ciphertexts. Carrying out this procedure recursively, we can homomorphically compute any circuit f of multiplicative depth L.

Ciphertext structure during evaluation.

During the homomorphic evaluation a ciphertext will be of the form $c = (\mathbf{v}, \ell)$ where ℓ is the "level tag" mentioned above. At any point of the evaluation we will have that ℓ is between 0 (for fresh ciphertexts at the input layer) and L (at the output layer). We define homomorphic evaluation only among ciphertexts at the same level. Since our circuit is layered we will not have to worry about homomorphic evaluation occurring among ciphertexts at different levels. Consistently with the fact a level tag represents the multiplicative depth of a ciphertext, addition gates will keep the level of ciphertexts unchanged, whereas multiplication gates will increase it by one. Finally, we will keep the invariant that the output of each gate evaluation $c = (\mathbf{v}, \ell)$ is such that

$$\langle \mathbf{k}_\ell , \mathbf{v} \rangle = \mu \tag{4.2}$$

where μ is the correct plaintext output of the gate.

Homomorphic Evaluation of gates:

- *Addition gates.* Homomorphic evaluation of an addition gates on inputs $c_1, \dots, c_t$ where $c_i = (\mathbf{v}_i, \ell)$ is performed by outputting

  $$c_{\text{add}} = (\mathbf{v}_{\text{add}}, \ell) := \Big(\sum_i \mathbf{v}_i, \ell\Big)$$

  Informally, one can see that

  $$\langle \mathbf{k}_\ell\,, \mathbf{v}_{\text{add}} \rangle = \langle \mathbf{k}_\ell\,, \sum_i \mathbf{v}_i \rangle = \sum_i \langle \mathbf{k}_\ell\,, \mathbf{v}_i \rangle = \sum_i \mu_i$$

  where μ_i is the plaintext corresponding to $\mathbf{v}_i$. This satisfies the invariant in Eq. 4.2.

- *Multiplication gates.* We show how to multiply ciphertexts c, c' where $c = (\mathbf{v}, \ell)$ and $c' = (\mathbf{v}', \ell)$ to obtain an output ciphertext $c_{\text{mult}} = (\mathbf{v}_{\text{mult}}, \ell + 1)$.

  The homomorphic multiplication algorithm will set

  $$\mathbf{v}_{\text{mult}} := \sum_{i,j \in [\lambda]} h_{i,j} \cdot \mathbf{a}_{\ell+1,i,j}$$

  where $h_{i,j} = \mathbf{v}[i] \cdot \mathbf{v}'[j]$ for $i, j \in [\lambda]$.

  The final output ciphertext will be

  $$c_{\text{mult}} := (\mathbf{v}_{\text{mult}}, \ell + 1).$$

This satisfies the invariant in Eq. 4.2 as

$$
\begin{aligned}
\langle \mathbf{k}_{\ell+1}\,, \mathbf{v}_{\text{mult}} \rangle &= \langle \mathbf{k}_{\ell+1}\,, \sum_{i,j\in[\lambda]} h_{i,j} \cdot \mathbf{a}_{\ell+1,i,j} \rangle \\
&= \sum_{i,j\in[\lambda]} (h_{i,j} \cdot \langle \mathbf{k}_{\ell+1}\,, \mathbf{a}_{\ell+1,i,j} \rangle) \\
&= \sum_{i,j\in[\lambda]} (h_{i,j} \cdot \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \\
&= \sum_{i,j\in[\lambda]} (\mathbf{v}[i] \cdot \mathbf{v}'[j] \cdot \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \\
&= \Big(\sum_{i\in[\lambda]} \mathbf{v}[i] \cdot \mathbf{k}_\ell[i]\Big) \cdot \Big(\sum_{j\in[\lambda]} \mathbf{v}'[j] \cdot \mathbf{k}_\ell[j]\Big) \\
&= \langle \mathbf{k}_\ell\,, \mathbf{v} \rangle \cdot \langle \mathbf{k}_\ell\,, \mathbf{v}' \rangle \\
&= \mu \cdot \mu'
\end{aligned}
$$

where in the third and fourth equality we used respectively Eq. 4.1 and the definition of $h_{i,j}$, and μ, μ' are the plaintexts corresponding to $\mathbf{v}$ $\mathbf{v}'$ respectively.

4.2.2 Security Analysis

Theorem 4.2.4 (Security)**.** The scheme HE is CPA secure against NC^1 adversaries (Definition 4.1.7) under the assumption $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$.

Proof. We are going to prove that there exists infinite $\Lambda \subseteq \mathbb{N}$ such that $(\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(0)) \sim_\Lambda (\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(1))$.

When using the notations $\mathbf{M}^{\mathsf{f}}$ and $\mathbf{M}^{\text{kg}}$ we will always denote matrices distributed respectively according to $\mathcal{D}^{\mathsf{f}}_\lambda$ and $\mathcal{D}^{\text{kg}}$, where $\mathcal{D}^{\mathsf{f}}_\lambda$ and $\mathcal{D}^{\text{kg}}$ are the distributions defined in Lemma 4.2.1.

We will define the (randomized) encoding procedure $\mathsf{E} : \{0,1\}^{\lambda \times \lambda} \to \{0,1\}\lambda$ defined as

$$\mathsf{E}(\mathbf{M}, b) = \mathbf{r}^\intercal \mathbf{M} + (0 \ \ldots 0 \ b)^\intercal \,,$$

where r is uniformly distributed in $\{0,1\}^\lambda$. The functions we will pass to E will be distributed either according to $\mathbf{M}^{\mathsf{kg}}$ or $\mathbf{M}^{\mathsf{f}}$. Notice that: *(i)* $\mathsf{E}(\mathbf{M}^{\mathsf{kg}}, b)$ is distributed identically to $\mathsf{HE.Enc}_{\mathsf{pk}}(b)$; *(ii)* $\mathsf{E}(\mathbf{M}^{\mathsf{f}}, b)$ corresponds to the uniform distribution over $\{0,1\}^\lambda$ because (by Lemma 4.2.1) $\mathbf{M}^{\mathsf{f}}$ has full rank and hence $\mathbf{r}^\intercal \mathbf{M}^{\mathsf{f}}$ must be uniformly random.

We will denote with $\mathbf{M}_1^{\mathsf{kg}}, \ldots, \mathbf{M}_L^{\mathsf{kg}}$ the matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$ used to construct the evaluation key in $\mathsf{HE.Keygen}$ (see definition). Recall these matrices are distributed according to $\mathcal{D}^{\mathsf{kg}}$ as in Lemma 4.2.1.

We will also define the following vectors:

$$\boldsymbol{\alpha}_\ell^{\mathsf{kg}} := \{\mathsf{E}(\mathbf{M}_{\ell+1}^{\mathsf{kg}}, \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \mid i, j \in [\lambda]\} \qquad \boldsymbol{\alpha}_\ell^{\mathsf{f}} := \{\mathsf{E}(\mathbf{M}_{\ell+1}^{\mathsf{f}}, \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \mid i, j \in [\lambda]\} \,,$$

where $\mathbf{k}_\ell$ is defined as in $\mathsf{HE.Keygen}$ and the matrices in input to E will be clear from the context. Notice that all the elements of $\boldsymbol{\alpha}_\ell^{\mathsf{kg}}$ are encryptions, whereas all the elements of $\boldsymbol{\alpha}_\ell^{\mathsf{f}}$ are uniformly distributed.

We will use a standard hybrid argument. Each of our hybrids is parametrized by a bit b. This bit informally marks whether the hybrid contains an element indistinguishable from an encryption of b.

- $\mathcal{E}^b := (\mathbf{M}_0^{\mathsf{kg}}, \mathsf{E}(\mathbf{M}_0^{\mathsf{kg}}, b), \boldsymbol{\alpha}_1^{\mathsf{kg}}, \ldots, \boldsymbol{\alpha}_L^{\mathsf{kg}})$ where $\mathbf{M}_0^{\mathsf{kg}}$ corresponds to the public key of our scheme. Notice that $\boldsymbol{\alpha}_\ell^{\mathsf{kg}} \equiv \{\mathbf{a}_{\ell,i,j} \mid i, j \in [\lambda]\}$ where $\mathbf{a}_{\ell,i,j}$ is as defined in $\mathsf{HE.Keygen}$. This hybrid corresponds to the distribution $(\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(b))$.

- $\mathcal{H}_0^b := (\mathbf{M}_0^{\mathsf{f}}, \mathsf{E}(\mathbf{M}^{\mathsf{f}}, b), \boldsymbol{\alpha}_1^{\mathsf{kg}}, \ldots, \boldsymbol{\alpha}_L^{\mathsf{kg}})$. The only difference from $\mathcal{E}$ is in the first two components where we replaced the actual public key and ciphertext with a full rank matrix distributed according to $\mathcal{D}^{\mathsf{f}}_{\lambda}$ and a random vector of bits.

- For $\ell \in [L]$ we define

$$\mathcal{H}^b_\ell := (\mathbf{M}^{\mathsf{f}}_0, \mathsf{E}(\mathbf{M}^{\mathsf{f}}, b), \boldsymbol{\alpha}^{\mathsf{f}}_1, \ldots, \boldsymbol{\alpha}^{\mathsf{f}}_\ell, \boldsymbol{\alpha}^{\mathsf{kg}}_{\ell+1}, \ldots, \boldsymbol{\alpha}^{\mathsf{kg}}_L)\,.$$

We will proceed proving that

$$\mathcal{E}^0 \sim_\Lambda \mathcal{H}^0_0 \sim_\Lambda \mathcal{H}^0_1 \sim_\Lambda \ldots \sim_\Lambda \mathcal{H}^0_L \sim_\Lambda \mathcal{H}^1_L \sim_\Lambda \ldots \sim_\Lambda \mathcal{H}^1_1 \sim_\Lambda \mathcal{H}^1_0 \sim_\Lambda \mathcal{E}^1$$

through a series of smaller claims. In the remainder of the proof Λ refers to the set in Lemma 4.2.1.

- $\mathcal{E}^0 \sim_\Lambda \mathcal{H}^0_0$: if this were not the case we would be able to distinguish $\mathbf{M}^{\mathsf{kg}}_0$ from $\mathbf{M}^{\mathsf{f}}_0$ for some of the values in the set Λ thus contradicting Lemma 4.2.1.

- $\mathcal{H}^0_{\ell-1} \sim_\Lambda \mathcal{H}^0_\ell$ for $\ell \in [L]$: assume by contradiction this statement is false for some $\ell \in [L]$. That is

$$(\mathbf{M}^{\mathsf{f}}_0, \mathsf{E}(\mathbf{M}^{\mathsf{f}}_0, b), \boldsymbol{\alpha}^{\mathsf{f}}_1, \ldots, \boldsymbol{\alpha}^{\mathsf{f}}_{\ell-1}, \boldsymbol{\alpha}^{\mathsf{kg}}_\ell, \ldots, \boldsymbol{\alpha}^{\mathsf{kg}}_L) \not\sim_\Lambda (\mathbf{M}^{\mathsf{f}}_0, \mathsf{E}(\mathbf{M}^{\mathsf{f}}_0, b), \boldsymbol{\alpha}^{\mathsf{f}}_1, \ldots, \boldsymbol{\alpha}^{\mathsf{f}}_\ell, \boldsymbol{\alpha}^{\mathsf{kg}}_{\ell+1}, \ldots, \boldsymbol{\alpha}^{\mathsf{kg}}_L)\,.$$

  Recall that, by definition, the elements of $\boldsymbol{\alpha}^{\mathsf{kg}}_\ell$ are all encryptions whereas the elements of $\boldsymbol{\alpha}^{\mathsf{f}}_\ell$ are all randomly distributed values. This contradicts the the semantic security of the scheme PKE (by a standard hybrid argument on the number of ciphertexts).

- $\mathcal{H}^0_L \sim_\Lambda \mathcal{H}^1_L$: the distributions associated to these two hybrids are identical. In fact, notice the only difference between these two hybrids is in the second component: $\mathsf{E}(\mathbf{M}^{\mathsf{f}}, 0)$ in $\mathcal{H}^0_L$ and $\mathsf{E}(\mathbf{M}^{\mathsf{f}}, 1)$ in $\mathcal{H}^1_L$. As observed above $\mathsf{E}(\mathbf{M}^{\mathsf{f}}, b)$ is uniformly distributed, which proves the claim.

All the claims above can be proven analogously for $\mathcal{E}^1, \mathcal{H}^1_0$ and $\mathcal{H}^1_\ell$-s. $\square$

4.2.3 Efficiency and Homomorphic Properties of Our Scheme

Our scheme is secure against adversaries in the class NC^1. This implies that we can run $\mathsf{HE.Eval}$ only on functions f that are in NC^1, otherwise the evaluator would be able to break the semantic security of the scheme. However we have to ensure that the *whole* homomorphic evaluation stays in NC^1. The problem is that homomorphically evaluating f has an overhead with respect to the "plain" evaluation of f. Therefore, we need to determine for which functions f, we can guarantee that $\mathsf{HE.Eval}(F, \dots)$ will stay in NC^1.

In terms of circuit depth, the main overhead when evaluating f homomorphically is given by the multiplication gates (addition, on the other hand, is "for free" — see definition of $\mathsf{HE.Eval}$ above). A single homomorphic multiplication can be performed by a depth two $\mathsf{AC}^0[2]$ circuit, but this requires depth $\Omega(\log(n))$ with a circuit of fan-in two. Therefore, a circuit for f with $\omega(1)$ multiplicative depth would require an evaluation of $\omega(\log(n))$ depth, which would be out of NC^1. On the other hand, observe that for any function f in $\mathsf{AC}^0[2]$ with constant multiplicative depth, the evaluation stays in $\mathsf{AC}^0[2]$. This because there is a constant number (depth) of homomorphic multiplications each requiring an $\mathsf{AC}^0[2]$ computation.

We can now state the following result, derived from the observations above and the fact that the invariant in Eq. 4.2 is preserved throughout homomorphic evaluation.

Theorem 4.2.5. Let $\mathsf{AC}^0_{\mathrm{CM}}[2]$ the family of circuits in $\mathsf{AC}^0[2]$ with constant multiplicative depth (see Definition 4.1.3). The scheme HE is leveled $\mathsf{AC}^0_{\mathrm{CM}}[2]$-homomorphic. Key generation, encryption, decryption and evaluation are all computable in $\mathsf{AC}^0_{\mathrm{CM}}[2]$.

4.2.4 Beyond Constant Multiplicative Depth

In the previous section we saw how our scheme is homomorphic for a class of constant-depth, unbounded fan-in arithmetic circuits in GF(2) with *constant multiplicative depth*, i.e. polynomials in GF(2) of constant degree. We now show how to overcome this limitation by slightly changing our scheme and using techniques from [49] to approximate $\mathsf{AC}^0[2]$ circuits with low-degree polynomials.

Lemma 4.2.6 ([49])**.** Let C be an $\mathsf{AC}^0_{\mathsf{Q}}[2]$ circuit of depth d. Then there exists a randomized circuit $C' \in \mathsf{AC}^0_{\mathsf{CM}}[2]$ such that, for all x,

$$\Pr[C'(x) \neq C(x)] \leq \epsilon\,,$$

where $\epsilon = O(1)$. The circuit C' uses $O(n)$ random bits and its representation can be computed in NC^0 from a representation of C.

Proof. Consider a circuit $C \in \mathsf{AC}^0_{\mathsf{Q}}[2]$ and let $K = O(1)$ be the total number of AND and OR gates with non-constant fan-in. We can replace every OR gate of fan-in $m = \omega(1)$ with a randomized "gadget" that takes in input m additional random bits and computes the function

$$\hat{g}_{\mathsf{OR}}(x_1, \ldots, x_m; r_1, \ldots, r_m) := \sum_{i \in [m]} x_i r_i\,.$$

This function can be implemented in constant multiplicative depth with one XOR gate and m AND gates of fan-in two. Let $\mathbf{x} = (x_1, \ldots, x_m)$ and $\mathbf{r} = (r_1, \ldots, r_m)$. The probabilistic gadget $\hat{g}_{\mathsf{OR}}$ has one-sided error. if $x_i = 0$ (i.e. if $\mathsf{OR}(\mathbf{x}) = 0$) then $\Pr[\hat{g}_{\mathsf{OR}}(\mathbf{x}; \mathbf{r}) = 0] = 1$; otherwise $\Pr[\hat{g}_{\mathsf{OR}}(\mathbf{x}; \mathbf{r}) = 1] = \frac{1}{2}$.

In a similar fashion, we can replace every unbounded fan-in AND gate with a randomized gadget in computing

$$\hat{g}_{\mathsf{AND}}(x_1, \ldots, x_m; r_1, \ldots, r_m) := 1 - \sum_{i \in [m]} (1 - x_i) r_i\,.$$

This gadget can also be implemented in constant-multiplicative depth and has one-sided error $1/2$. Finally, let us observe that $\Pr[C'(x) \neq C(x)] \leq \epsilon$ with ϵ being a constant, because we have only a constant number of gates to be replaced with gadgets for $\hat{g}_{\mathsf{OR}}$ or $\hat{g}_{\mathsf{AND}}$.

We only provide the intuition for why the transformations above can be carried out in NC^0. Assume the encoding of a circuit as a list of gates in the form $(g, t_g, in_1, \ldots, in_m)$ where g and t are respectively the index of the output wire of the gate and its type (possibly of the form "input" or "random input") and the in_i-s are the indices of the input wire of g. The transformation from C to C' needs to simply copy all the items in the list except for the gates of unbounded fan-in. We will assume the encoding conventions of C always puts these gates at the end of the list[5]. For each of such gates the transformation circuit needs to: add appropriate $r_1, \ldots, r_m$ to the list, add m AND gates and one XOR, possibly (if we are transforming an AND gate) add negation gates. All this can be carried out based on wire connections and the type of the gate (a constant-size string) and thus in NC^0. □

In the construction above, we built C' by replacing every gate $g \in \mathsf{S}_{\omega(1)}(C)$ (as in Definition 4.1.4) with a (randomized) gadget G_g. The output of each these gadgets will be useful in order to keep the low complexity of the decryption algorithm in our next homomorphic encryption scheme. We shall use an "expanded" version of C', the multi-output circuit C'_{exp}.

Definition 4.2.7 (Expanded Approximating Function)**.** Let C be a circuit in $\mathsf{AC}^0_{\mathsf{Q}}[2]$ and let C' be a circuit as in the proof of Lemma 4.2.6. We denote by $G_g(\mathbf{x}; \mathbf{r})$ the output of the gadget G_g when C' is evaluated on inputs $(\mathbf{x}; \mathbf{r})$. On input $(\mathbf{x}; \mathbf{r})$, the multi-output circuit C'_{exp} output $C'(\mathbf{x}; \mathbf{r})$ together with the outputs of the $O(1)$ gadgets G_g for each $g \in \mathsf{S}_{\omega(1)}(C)$. Finally, we denote with $\mathsf{GenApproxFun}$ the algorithm computeing a representation of C'_{exp} from a representation of C.

Lemma 4.2.8. There exists a deterministic algorithm $\mathsf{DecodeApprox}$ computable in $\mathsf{AC}^0[2]$ with the following properties. For every circuit C in $\mathsf{AC}^0_{\mathsf{Q}}[2]$ computing the function f, there exists

[5]This allows our NC^0 circuit to to "know" which gates to copy and which ones to transform based on their position only.

$\mathbf{aux}_f \in \{0,1\}^{O(1)}$ such that for all $\mathbf{x} \in \{0,1\}^n$

$$\Pr[\mathsf{DecodeApprox}(C'_{exp}(\mathbf{x};\mathbf{r}^{(1)}),\ldots,C'_{exp}(\mathbf{x};\mathbf{r}^{(s)})) = C(\mathbf{x})] \geq 1 - \mathsf{neg}(s)\,,$$

where C' is an approximating circuit as in Lemma 4.2.6, the probability is taken over the uniformly distributed bit vectors $\mathbf{r}^{(i)}$-s for $i \in [s]$, C'_{exp} is as in Definition 4.2.7. Finally, there exists a function $\mathsf{GenDecodeAux}$ that computes $\mathbf{aux}_f$ from a representation of C in NC^0.

Proof. Before we provide a construction for $\mathsf{DecodeApprox}$, let us observe how we can amplify the error of C'. Consider for example a gadget $\hat{g}_{\mathsf{OR}}$ constructed as in the proof of Lemma 4.2.6, approximating an OR gate in C. If we repeat the execution of the gadget s times, every time using fresh random bit vectors $\mathbf{r}'^{(1)},\ldots,\mathbf{r}'^{(s)}$, then we can correctly compute $\mathsf{OR}(\mathbf{x}')$ with overwhelming probability. Define $h_{\mathsf{OR}}(\mathbf{x}';\mathbf{r}'^{(1)},\ldots,\mathbf{r}'^{(s)}) := \mathsf{OR}(\hat{g}_{\mathsf{OR}}(\mathbf{x};\mathbf{r}'^{(1)}),\ldots,\hat{g}_{\mathsf{OR}}(\mathbf{x}';\mathbf{r}'^{(s)}))$. Clearly $\Pr[h_{\mathsf{OR}}(\mathbf{x}';\mathbf{r}'^{(1)},\ldots,\mathbf{r}'^{(s)}) = \mathsf{OR}(\mathbf{x}')] \geq 1 - 2^{-s}$. In a similar fashion we can define $h_{\mathsf{AND}}(\mathbf{x}';\mathbf{r}'^{(1)},\ldots,\mathbf{r}'^{(s)}) := \mathsf{AND}(\hat{g}_{\mathsf{AND}}(\mathbf{x};\mathbf{r}'^{(1)}),\ldots,\hat{g}_{\mathsf{AND}}(\mathbf{x}';\mathbf{r}'^{(s)}))$. It holds that $\Pr[h_{\mathsf{AND}}(\mathbf{x}';\mathbf{r}'^{(1)},\ldots,\mathbf{r}'^{(s)}) = \mathsf{AND}(\mathbf{x}')] \geq 1 - 2^{-s}$.

If C' were composed by a single gadget $\hat{g}_{\mathsf{OR}}$ (resp. $\hat{g}_{\mathsf{AND}}$) we could just let $\mathsf{DecodeApprox}$ be the same as h_{OR} (resp. h_{AND}) and we would be done. To deal with multiple gadgets, however, we need a more general approach. For sake of presentation, assume there are only gadgets approximating OR gates and let us temporarily ignore $\mathbf{aux}_f$. We can write each of the $C'_{exp}(\mathbf{x};\mathbf{r}^{(j)})$ input to $\mathsf{DecodeApprox}$ as $(z^{(j)}, y_1^{(j)},\ldots,y_K^{(j)})$ where $K := |\mathsf{S}_{\omega(1)}|$, $z^{(j)}$ is the output of $C'(\mathbf{x},\mathbf{r}^{(j)})$ and $y_i^{(j)}$ is the output of the $i-th$ gadget when provided random bits from $\mathbf{r}^{(j)}$. Define y_i^* as $y_i^* := \mathsf{OR}(y_i^{(1)},\ldots,y_i^{(s)})$. We then let the output of $\mathsf{DecodeApprox}$ be z^{j^*} where j^* is such that for all $i \in [K]$ it is the case that $y_i^{j^*} = y_i^*$. By the union bound the probability of $z^{j^*} \neq C(\mathbf{x})$ is upper bounded by $K \cdot 2^{-s}$, which is negligible since $K = O(1)$. To generalize this same approach to the scenario including both OR and AND gadgets we let the string $\mathbf{aux}_f$ include information on the type of gates in $\mathsf{S}_{\omega(1)}$. This way $\mathsf{DecodeApprox}$ can use $\hat{g}_{\mathsf{OR}}$ or $\hat{g}_{\mathsf{AND}}$ accordingly. Clearly the representation of $\mathbf{aux}_f$ can be computed by a representation of C in NC^0. $\square$

Homomorphic Evaluations of $\mathsf{AC}^0_{\mathbf{Q}}[2]$ Circuits

Below is a variation of our homomorphic scheme that can evaluate all circuits in $\mathsf{AC}^0_{\mathrm{Q}}[2]$ in $\mathsf{AC}^0[2]$. This time, in order to evaluate circuit C, we perform several homomorphic evaluations of the randomized circuit C' (as in Lemma 4.2.6). To obtain the plaintext output of C we can decrypt all the ciphertext outputs and take the majority result. Notice that this scheme is still compact. As we use a randomized approach to evaluate f, the scheme HE' will be implicitly parametrized by a soundness parameter s. Intuitively, the probability of a function f being evaluated incorrectly will be upper bounded by 2^{-s}.

For our new scheme we will use the following auxiliary functions:

Definition 4.2.9 (Auxiliary Functions for HE')**.**

Let $f : \{0,1\}^t \to \{0,1\}$ be represented as an arithmetic circuit as in HE and pk a public key for the scheme HE that includes the evaluation key. Let s be a soundness parameter. We denote by f' the expanded randomized function approximating f as in Definition 4.2.7; let $t' = O(t)$ be the number of additional random bits f' will take in input.

- $\mathsf{GenApproxFun}(f)$:
  - Computes and returns the representation of the expanded approximating function f' as in Definition 4.2.7.
- $\mathsf{GenDecodeAux}(f)$:
  - Computes and returns the auxiliary string $\mathbf{aux}_f$ from a representation of f as in Lemma 4.2.8.
- $\mathsf{SampleAuxRandomness}_s(\mathsf{pk}, f')$:

1. We assume f' is the expanded randomized function approximating f as in Definition 4.2.7; let $t' = O(t)$ be the number of additional random bits f' will take in input.
2. Sample $s \cdot t'$ random bits $r_1^{(1)}, \ldots, r_{t'}^{(1)}, \ldots, r_1^{(s)}, \ldots, r_{t'}^{(s)}$;
3. Compute $\hat{\mathbf{r}}_{\text{aux}} := \{\hat{r}_j^{(i)} \mid \hat{r}_j^{(i)} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(r_j^{(i)}), i \in [s], j \in [t']\}$;
4. Output $\hat{\mathbf{r}}_{\text{aux}}$.

- $\mathsf{EvalApprox}_s(\mathsf{pk}, f', c_1, \ldots, c_t, \hat{\mathbf{r}}_{\text{aux}})$:
  1. Let $\hat{\mathbf{r}}_{\text{aux}} = \{\hat{r}_j^{(i)} \mid i \in [s], j \in [t']\}$.
  2. For $i \in [s]$, compute $\mathbf{c}_i^{\text{out}} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}(f', c_1, \ldots c_t, \hat{r}_1^{(i)}, \ldots, \hat{r}_{t'}^{(i)})$;
  3. Output $\mathbf{c} = (\mathbf{c}_1^{\text{out}}, \ldots, \mathbf{c}_s^{\text{out}})$[6].

The new scheme HE' with soundness parameter s follows. Notice that the evaluation function outputs an auxiliary string $\mathbf{aux}_f$ together with the proper ciphertext $\mathbf{c}$. This is necessary to have a correct decoding in decryption phase.

- Key generation and encryption are the same as in HE.
- $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_t)$:
  1. Compute $f' \leftarrow \mathsf{GenApproxFun}(f)$;
  2. Compute $\hat{\mathbf{r}}_{\text{aux}} \leftarrow \mathsf{SampleAuxRandomness}_s(\mathsf{pk}, f')$;
  3. $\mathbf{aux}_f \leftarrow \mathsf{GenDecodeAux}(f)$;
  4. $\mathbf{c} \leftarrow \mathsf{EvalApprox}_s(\mathsf{pk}, f', c_1, \ldots, c_t, \hat{\mathbf{r}}_{\text{aux}})$;

[6]Recall that the output of the expanded approximating function f' is a bit string and thus each $\mathbf{c}_i^{\text{out}}$ encrypts a bit string.

5. Output $(\mathbf{c}, \mathbf{aux}_f)$.

- $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\mathbf{c} = (\mathbf{c}_1^{\text{out}}, \ldots, \mathbf{c}_s^{\text{out}}), \mathbf{aux}_f)$:

  1. Let $\mathbf{y}_i^{\text{out}} \leftarrow \mathsf{HE}.\mathsf{Dec}_{\mathsf{sk}}(\mathbf{c}_i^{\text{out}})$ for $i \in [s]$;

  2. Output $\mathsf{DecodeApprox}_f(\mathbf{y}_1^{\text{out}}, \ldots, \mathbf{y}_s^{\text{out}})$.

Remark 9. Given in input a function f not necessarily of constant multiplicative depth, GenApproxFun returns a function f' of constant multiplicative depth that approximates it. As stated in Lemma 4.2.6, GenApproxFun is computable in NC^0 and so is GenDecodeAux. The function SampleAuxRandomness in $\mathsf{AC}^0_{\text{CM}}[2]$ and EvalApprox makes parallel invocations to HE.Eval which is computable in $\mathsf{AC}^0_{\text{CM}}[2]$ when provided in input a function in $\mathsf{AC}^0_{\text{CM}}[2]$ (Theorem 4.2.5). This fact will be useful when showing the completeness of our verifiable computation schemes in Section 4.3.

Theorem 4.2.10. Let $\mathsf{AC}^0_{\text{Q}}[2]$ the family of circuits in $\mathsf{AC}^0[2]$ with quasi-constant multiplicative depth as in Definition 4.1.4. The scheme HE' above with soundness parameter $s = \Omega(\lambda)$ is leveled $\mathsf{AC}^0_{\text{Q}}[2]$-homomorphic. Key generation, encryption and evaluation can be computed in $\mathsf{AC}^0_{\text{CM}}[2]$. Decryption is computable in $\mathsf{AC}^0[2]$.

4.3 Fine-Grained Verifiable Computation

In this section we describe our private verifiable computation scheme. Our constructions are heavily based on the techniques in [16] to obtain (reusable) verifiable computation from fully homomorphic encryption. In order to guarantee that these techniques also work within NC^1 we prove that: *(i)* the constructions can be computed in low-depth; *(ii)* the reductions in the security proofs can be carried out in low-depth.

THE SCHEME FROM [16]. To derive Verifiable Computation from Homomorphic Encryption,

[16] follows this approach. The Client, in the expensive preprocessing phase, selects a random input r, encrypts it $c_r = E(r)$ and homomorphically compute $c_{f(r)}$ an encryption of $f(r)$. During the online phase, the Client, on input x, computes $c_x = E(x)$ and submits the ciphertexts c_x, c_r in random order to the Server, who homomorphically compute $c_{f(r)} = E(f(r))$ and $c_{f(x)} = E(f(x))$ and returns them to the Client. The Client given the message c_0, c_1 from the Server, checks that $c_b = c_{f(r)}$ (for the appropriate bit b) and if so accepts $y = D(c_{f(x)})$ as $y = f(x)$. The semantic security of E guarantees that this protocol has soundness error $1/2$ (which can be reduced by parallel repetition). This scheme is however one-time, as a malicious server can figure out which one is the test ciphertext $c_{f(r)}$ if it is used again.

To make this scheme "many time secure", [16] uses the paradigm introduced in [23] of running the one-time scheme "under the covers" of a different homomorphic encryption key each time.

4.3.1 A One-time Verification Scheme

Before we present our variant of the one-time construction in [16], we present two auxiliary lemmas that guarantee that our protocols are computable in $\mathsf{AC}^0[2]$. We refer the reader to [33, 45] for the proof Lemma 4.3.1.

Lemma 4.3.1. [33, 45] There are uniform AC^0 circuits $C : \{0,1\}^{\mathsf{poly}(l)} \to [l]^l$ of size $\mathsf{poly}(l)$ and depth $O(1)$ whose output distribution have statistical distance $\leq 2^{-l}$ from the uniform distribution over permutations of $[l]$.

Lemma 4.3.2. There are uniform $\mathsf{AC}^0[2]$ circuits $C : [l]^l \times \{0,1\}^l \to \{0,1\}^l$ of size $O(l^2)$ where $C(\pi, (x_1, \ldots, x_l)) = (\pi(1), \ldots, \pi(l))$ and π is a permutation.

Proof. Let $\mathbf{x} = (x_1, \ldots, x_l)$ the bits to permute and let π be a permutation If π is represented as a permutation matrix with rows $\mathbf{r}_1, \ldots, \mathbf{r}_l$, we can permute $\mathbf{x}$ by simply performing l parallel inner products $\langle \mathbf{x}\,, \mathbf{r}_i \rangle$-s, which is in $\mathsf{AC}^0[2]$. We now describe how to generate the permutation matrix from a binary representations $x_1, \ldots, x_{\mathsf{lg}(l)}$ of the integers in $[l]$. Let $f_i : \{0,1\}^{\mathsf{lg}(l)} \to \{0,1\}^l$ be the function that computes the i-th row of the permutation matrix. We can define f_i as follows:

$$f_i(x_1, \dots, x_{\mathsf{lg}(l)}) := \mathsf{eq}([i-1]_2, (x_1, \dots, x_{\mathsf{lg}(l)})\,,$$

where $[i-1]_2$ is the binary representation of $i-1$ and eq returns 1 if its two inputs (each of lenght $\mathsf{lg}(l)$) are equal. The function f_i is clearly in $\mathsf{AC}^0[2]$. □

In Figure 4.2 we describe an adaptation of the one-time secure delegation scheme from [16]. We make non-black box use of our homomorphic encryption scheme HE' (Section 4.2.4) with soundness parameter $s = \lambda$. Notice that. during the preprocessing phase, we fix the "auxiliary randomness" for $\mathsf{EvalApprox}$ (and thus for $\mathsf{HE}'.\mathsf{Eval}$) once and for all. We will use that same randomness for all the input instances. This choice does not affect the security of the construction. We remind the reader that we will simplify notation by considering the evaluation key of our somewhat homomorphic encryption scheme as part of its public key.

If x is a vector of bits $x_1, \dots, x_n$, below we will denote with $\mathsf{HE}'.\mathsf{Enc}(x)$ the concatenation of the bit by bit ciphertexts $\mathsf{HE}'.\mathsf{Enc}(x_1), \dots, \mathsf{HE}'.\mathsf{Enc}(x_n)$. We denote by $\mathsf{HE}'.\mathsf{Enc}(\bar{0})$ the concatenation of n encryptions of 0, $\mathsf{HE}'.\mathsf{Enc}(0)$.

Remark 10 (On deterministic homomorphic evaluation)**.** As pointed out in [16], one requirement for the approach in Figure 4.2 to work is for the homomorphic evaluation to be deterministic. We point out that once $\hat{\mathbf{r}}_{\text{aux}}$ are fixed once and for all the homomorphic evaluation in $\mathsf{VC}.\mathsf{Compute}$ is deterministic.

Lemma 4.3.3 (Completeness of $\mathcal{VC}$)**.** The verifiable computation scheme $\mathcal{VC}$ in Figure 4.2 has overwhelming completeness (Definition 4.1.12) for the class $\mathsf{AC}^0_{\mathsf{Q}}[2]$.

Proof. The proof is straightforward and stems directly from the homomorphic properties of HE' (Theorem 4.2.10). In fact, by construction and by definition of HE' (Section 4.2.4), the distribution

Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function and $\mathsf{GenApproxFun}, \mathsf{SampleAuxRandomness}$ and $\mathsf{EvalApprox}$ as in Definition 4.2.9.

- $\mathsf{VC.KeyGen}(1^\lambda, f) \to (\mathsf{pk_W}, \mathsf{sk_D})$: We assume function f represented as
  1. Generate a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}'.\mathsf{Keygen}(1^\lambda)$.
  2. Generate the approximating function $f' \leftarrow \mathsf{GenApproxFun}(f)$ and auxiliary string $\mathbf{aux}_f \leftarrow \mathsf{GenDecodeAux}(f)$;
  3. Generate the ciphertext of the auxiliary random input for homomorphic evaluation $\hat{\mathbf{r}}_{\text{aux}} \leftarrow \mathsf{SampleAuxRandomness}_\lambda(\mathsf{pk}, f')$
  4. Compute t independent encryptions $\hat{r}_i = \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}}(\bar{0})$ and the homomorphic evaluations $\hat{w}_i = \hat{f}(\hat{r}_i) = \mathsf{EvalApprox}_s(\mathsf{pk}, f', \hat{r}_i, \hat{\mathbf{r}}_{\text{aux}})$ for $i \in [t]$;
  5. $\mathsf{pk_W} \leftarrow (\mathsf{pk}, f', \hat{\mathbf{r}}_{\text{aux}}), \mathsf{sk_D} \leftarrow (\{(\hat{r}_i, \hat{w}_i)_{i\in[t]}\}, \mathbf{aux}_f)$.
- $\mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x) \to (q_x, s_x)$:
  1. Compute t independent encryptions $\hat{r}_{i+t} = \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}}(x)$ for $i \in [t]$.
  2. Sample a random permutation $\pi \leftarrow_\$ S_{2t}$.
  3. $q_x \leftarrow (\hat{z}_{\pi(1)}, \dots, \hat{z}_{\pi(2t)}) = (\hat{r}_1, \dots, \hat{r}_{2t})$; $s_x \leftarrow \pi$
- $\mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x) \to a_x$:
  1. Compute $\hat{y}_i = \hat{f}(\hat{z}_i) = \mathsf{EvalApprox}_s(\mathsf{pk}, f', \hat{z}_i, \hat{\mathbf{r}}_{\text{aux}})$ for $i \in [2t]$.
  2. $a_x = (\hat{y}_1, \dots, \hat{y}_{2t})$.
- $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$:
  1. Check if $\hat{w}_i = \hat{y}_i$ for all $i \in [t]$.
  2. Check if $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(t+1)}, \mathbf{aux}_f) = \cdots = \mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(2t)}, \mathbf{aux}_f)$.
  3. If either of the two tests above fails, return $\perp$; otherwise return $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(t+1)}, \mathbf{aux}_f)$.

Figure 4.2: One-Time Delegation Scheme

of the $\hat{w}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, \hat{r}_i)$. Analogously, the distribution of $\hat{y}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, \hat{z}_i)$. □

Remark 11 (Efficiency of $\mathcal{VC}$)**.** In the following we consider the verifiable computation of a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ computable by an $\mathsf{AC}^0_{\mathsf{Q}}[2]$ circuit of size S.

- $\mathsf{VC.KeyGen}$ can be computed by an $\mathsf{AC}^0[2]$ circuit of size $O(\mathsf{poly}(\lambda)S)$;
- $\mathsf{VC.ProbGen}$ can be computed by an $\mathsf{AC}^0[2]$ circuit of size $O(\mathsf{poly}(\lambda)(m+n))$;
- $\mathsf{VC.Compute}$ can be computed by an $\mathsf{AC}^0[2]$ circuit of size $O(\mathsf{poly}(\lambda)S)$;
- $\mathsf{VC.Verify}$ can be computed by a $\mathsf{AC}^0[2]$ circuit of size $O(\mathsf{poly}(\lambda)(m+n))$ and whose (constant) depth is independent of the depth of f.

Lemma 4.3.4 (One-time Soundness)**.** Under the assumption that $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$ the scheme in Figure 4.2 is $(1,1)$-sound (one time secure) against NC^1 adversaries whenever t is chosen to be $\omega(\log(\lambda))$.

Proof. We follow the same proof structure as in the proof of Lemma 12 in [16]. We will keep part of the analysis informal, emphasizing why this proof still works for low-depth circuits. We refer the reader to [16] for further details.

The following observation will be crucial in the rest of the proof. Notice that, by construction and by definition of HE' (Section 4.2.4), the distribution of the $\hat{w}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, \hat{r}_i)$. Analogously, the distribution of $\hat{y}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, \hat{z}_i)$.

Consider an NC^1 adversary $\mathcal{A}^*$ that cheats with non-negligible probability in the one-time security experiment $\mathbf{Exp}^{\mathsf{Verif}}_{A}[\mathcal{VC}, f, \lambda, 1, 1]$ (Definition 4.1.13). Let $(\hat{r}_1, \dots, \hat{r}_t)$ be the independent copies of $\mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk_W}}(\bar{0})$ and $(\hat{r}_{t+1}, \dots, \hat{r}_{2t})$ the t independent copies of $\mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk_W}}(x)$ as above. Whenever the verification algorithm accepts, the adversary must have responded correctly on $\hat{r}_1, ..., \hat{r}_t$ and incorrectly (and consistently) on $\hat{r}_{t+1}, \dots, \hat{r}_{2t}$. Our goal is to bound the probability that the adversary succeeds in doing that.

First, notice that the view of the adversary is $(\mathsf{pk}_\mathsf{W}, \hat{r}_1, \ldots, \hat{r}_{2t})$, and identical to $(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^t, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(x)^t)$. By semantic security of the homomorphic encryption scheme, there exists an infinitely large set of parameters Λ such that

$$(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^t, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(x)^t) \sim_\Lambda (\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^{2t})$$

Consider a modified game where the adversary receives $(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^{2t})$. Denote by p the probability that the adversary succeeds in this game. By computational indistinguishability we have

$$\Pr[\mathcal{A}^* \text{ is correct on } (\hat{r}_1, \ldots, \hat{r}_t) \text{ and incorrect on } (\hat{r}_{t+1}, \ldots, \hat{r}_{2t})] \leq p + \mathsf{neg}(\lambda)$$

for all $\lambda \in \Lambda$. This inequality holds because we can test in NC^1 whether $\mathcal{A}^*$ cheats only on $(\hat{r}_{t+1}, \ldots, \hat{r}_{2t})$. Therefore, if the adversary's behavior differed significantly between the two games, one would be able to break the semantic security of the homomorphic scheme. Here we made use of the third fact in Lemma 4.1.6.

We now proceed to upper bound p. Observe that

$$p = \Pr[\mathcal{A}^* \text{ is correct on } (\hat{z}_{\pi(1)}, \ldots, \hat{z}_{\pi(t)}) \text{ and incorrect on } (\hat{z}_{\pi(t+1)}, \ldots, \hat{z}_{\pi(2t)})]$$

where the $\hat{z}_{\pi(i)}$-s are defined as in Figure 4.2. Because of Lemma 4.3.1 that the distribution of π is statistically indistinguishable from that of a uniformly random permutation. Also, observe that the answers $\hat{y}_i$ of the adversary are independent of π. We can then conclude that $p \leq \frac{1}{\binom{2t}{t}} + \mathsf{neg}(t)$, which concludes the security analysis. □

4.3.2 A Reusable Verification Scheme

We now describe how to obtain a reusable verification scheme $\overline{\mathcal{VC}}$ applying the transformation in [16] from one-time sound verification schemes through fully homomorphic encryption. The core

Let $\mathcal{VC}$ be the verifiable computation scheme defined in Figure 4.2. The reusable verifiable computation scheme $\overline{\mathcal{VC}} = (\overline{\mathsf{VC.KeyGen}}, \overline{\mathsf{VC.ProbGen}}, \overline{\mathsf{VC.Compute}}, \overline{\mathsf{VC.Verify}})$ is defined as follows.

- $\overline{\mathsf{VC.KeyGen}}(1^\lambda, f) \rightarrow (\mathsf{pk_W}, \mathsf{sk_D})$: The key generation stage is the same as in $\mathcal{VC}$.
- $\overline{\mathsf{VC.ProbGen}}_{\mathsf{sk_D}}(x) \rightarrow (\overline{q_x}, \overline{s_x})$:
  1. $(q_x, s_x) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x)$;
  2. Compute a fresh pair of keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{HE.Keygen}(1^\lambda)$;
  3. Compute $\hat{q}_x \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}_x}(q_x)$;
  4. $\overline{q_x} \leftarrow (\mathsf{pk}_x, \hat{q}_x)$; $\overline{s_x} \leftarrow (s_x, \mathsf{sk}_x)$
- $\overline{\mathsf{VC.Compute}}_{\mathsf{pk_W}}(\overline{q_x}) \rightarrow \overline{a_x}$:
  1. $\hat{a}_x \leftarrow \mathsf{HE.Eval}_{\mathsf{pk}_x}(\mathsf{VC.Compute}(\cdot, f), \hat{q}_x)$.
  2. $\overline{a_x} \leftarrow \hat{a}_x$.
- $\overline{\mathsf{VC.Verify}}_{\mathsf{sk_D}}(\overline{s_x}, \overline{a_x})$:
  1. $a_x \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}_x}(\hat{a}_x)$.
  2. return $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$.

Figure 4.3: Transformation from one-time $\mathcal{VC}$ scheme to a *reusable* $\mathcal{VC}$ scheme

idea behind the transformation in [16] is to encapsulate all the operations of a one-time verifiable computation scheme through homomorphic encryption. We instantiate this transformation with the one-time verifiable construction $\mathcal{VC}$, described in Figure 4.2, and the simplest of our two somewhat homomorphic encryption schemes, HE (defined in Section 4.2.1).

Corollary 4.3.5 (Completeness of $\overline{\mathcal{VC}}$)**.** The verifiable computation scheme $\overline{\mathcal{VC}}$ in Figure 4.3 has overwhelming completeness (Definition 4.1.12) for the class $\mathsf{AC}^0_{\mathrm{Q}}[2]$.

Proof. The completeness of the scheme above follows directly from the completeness of $\mathcal{VC}$ and the homomorphic properties of HE. Notice that we can use HE.Eval to homomorphically compute VC.Compute as the latter carries out a computation in $\mathsf{AC}^0_{\mathrm{CM}}[2]$ (although it is *approximating* a computation in $\mathsf{AC}^0_{\mathsf{Q}}[2]$). □

Remark 12 (Efficiency of $\overline{\mathcal{VC}}$)**.** The efficiency of $\overline{\mathcal{VC}}$ is analogous to that of $\mathcal{VC}$ with the exception of a circuit size overhead of a factor $O(\lambda)$ on the problem generation and verification algorithms and of $O(\lambda^2)$ for the computation algorithm. All algorithms in $\overline{\mathcal{VC}}$ are computable by constant depth circuit (of unbounded fan-in) and the depth of the verification algorithm is independent of the function F.

Theorem 4.3.6. Under the assumption that $\mathsf{NC}^1 \subsetneq \oplus\mathsf{L}/\mathsf{poly}$ the scheme $\overline{\mathcal{VC}}$ in Figure 4.3 is $(O(1), \mathsf{poly}(\lambda))$-sound (many-times secure) against NC^1 adversaries whenever t is chosen to be $\omega(\log(\lambda))$ in the underlying scheme $\mathcal{VC}$.

Proof. By Lemma 4.3.4 there exists an infinite set $\Lambda \subseteq \mathbb{N}$ of security parameters for which $\mathcal{VC}$ "is secure". By the proof of Lemma 4.3.4, this set is also the set of parameters where the somewhat homomorphic encryption scheme HE "is secure". We will show that for all values in this same set Λ, the probability of success of any NC^1 adversary in $\mathbf{Exp}_A^{\mathsf{Verif}}[\overline{\mathcal{VC}}, f, \lambda, O(1), \mathsf{poly}(\lambda)]$ is negligible.

Assume by contradiction there exists an NC^1 adversary $\mathcal{A}^*$ that achieves non-negligible advantage in $\mathbf{Exp}_A^{\mathsf{Verif}}[\overline{\mathcal{VC}}, f, \lambda, O(1), \mathsf{poly}(\lambda)]$ for some $\lambda \in \Lambda$.

Claim: If $\overline{\mathcal{VC}}$ is not secure for some $\lambda^* \in \Lambda$ then we can break the one-time security of $\mathcal{VC}$. Let $l = O(1)$ be the number of rounds in the many-time soundness experiment for $\overline{\mathcal{VC}}$. Consider the following NC^1 adversary $\mathcal{A}_1$ for the experiment $\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, 1, 1]$:

- $\mathcal{A}_1$ obtains a pair a public key $\mathsf{pk_W}$ and sends it to $\mathcal{A}^*$;
- For all rounds $i \in \{1, \ldots, l-1\}$, $\mathcal{A}_1$ replies to $\mathcal{A}^*$ queries by generating a fresh pair of keys $(\mathsf{pk}, \mathsf{sk})$ and sending back encryptions of $\mathsf{HE.Enc_{pk}}(\bar{0})$;
- At round l, $\mathcal{A}_1$ responds to all input queries but the last one as above. This, by experiment definition, is the input where $\mathcal{A}^*$ will try to cheat; we denote this input by x^*. Now $\mathcal{A}_1$ sends

x^* as the only input query in the one-time security experiment and will receive back q^*. It will then obtain a fresh pair of keys $(\mathsf{pk}^*, \mathsf{sk}^*)$ and send $\mathsf{HE.Enc}_{\mathsf{pk}^*}(q^*)$ to $\mathcal{A}^*$.

- $\mathcal{A}^*$ will respond with $\hat{a}^*$ and $\mathcal{A}_1$ will send $\mathsf{HE.Dec}_{\mathsf{sk}^*}(\hat{a})$ to the challenger for one-time security experiment.

The advantage of $\mathcal{A}_1$ depends on how likely is $\mathcal{A}^*$ can successfully cheat in that interaction. Let p be the advantage of $\mathcal{A}_1$ in the one-time security experiment. Clearly, if p is close to the advantage of $\mathcal{A}^*$ in the many-times security experiment $\mathcal{A}_1$ breaks the security of the one-time scheme.

Claim: the advantage of $\mathcal{A}_1$ is negligibly close to that of $\mathcal{A}^*$ in the many-time security game for security parameter λ^*. We can prove this by relying on the semantic security of the homomorphic encryption and on a hybrid argument.

Let $L = lm$, the total number of input queries in the many-times security experiment. We now define the hybrids $H^{(j)}$ with $j \in \{0, \ldots, L\}$. We define $H^{(0)}$ to be the exactly the many-time security experiment. For $j \in [L]$ we define $H^{(j)}$ to be an experiment where we respond to input queries with $\mathsf{HE.Enc}_{\mathsf{pk}_f}(\bar{0})$ where pk_f is a fresh public key up to input query j and behaves the many-time security experiment from input query $j + 1$ on. Notice that $H^{(L)}$ corresponds to the interaction with $\mathcal{A}_1$ above.

Denote by $A^{(j)}$ the output distribution of $\mathcal{A}^*$ when interacting with $H^{(j)}$. Intuitively, if the advantage of the $\mathcal{A}_1$ in the one-time experiment is significantly different from the advantage of $\mathcal{A}^*$ in the many-times security games, then $A^{(0)}$ and $A^{(L)}$ are not Λ-computationally indistinguishable.

Therefore (by Lemma 4.1.6), there exists $j \in [L]$ such that $A^{(j-1)} \not\sim_\Lambda A^{(j)}$.

Claim: If there exists $j \in [L]$ such that $A^{(j-1)} \not\sim_\Lambda A^{(j)}$ then we can break the semantic security of HE. Consider the following NC^1 adversary $\mathcal{A}_{\text{CPA}}$ which receives in input a “challenge” public key pk^*. $\mathcal{A}_{\text{CPA}}$ will interact with $\mathcal{A}^*$ simulating $H^{(j)}$ until receiving input query x_j. At this point it will compute q_j from $\mathsf{VC.ProbGen}(x_j)$ and send to the CPA challenger (see Remark 7) q_j and $\bar{0}$, receiving back an encryption c^* of either message under the public key pk^*. $\mathcal{A}_{\text{CPA}}$ will now send (pk^*, c^*) to $\mathcal{A}^*$ and continue simulating $H^{(j)}$ till the end of the experiment. The adversary $\mathcal{A}_{\text{CPA}}$ will check whether $\mathcal{A}^*$ cheated successfully at the end of the experiment and output (in the multiple-message CPA experiment) 1 if that is the case and 0 otherwise. This would allow $\mathcal{A}_{\text{CPA}}$ to have a noticeable advantage in the experiment thus breaking the semantic security of HE. $\square$

Appendix A

Appendix A

In this chapter we show that a fine-grained interactive proof yields a sequentially composable rational proof.

Definition (Fine-Grained Interactive Proof)**.** Let $\mathcal{C}$ be a complexity class. An interactive protocol (P, V) is a fine-grained interactive proof with respect to $\mathcal{C}$ if it has perfect completeness and for all $\widetilde{P} \in \mathcal{C}$ and all inputs x

$$\Pr[\mathsf{out}(\widetilde{P}, V)(x) \neq f(x)] \leq \mathsf{neg}(|x|)$$

Theorem. Let (P, V) be a fine-grained interactive proof with respect to class $\mathcal{C}$ for the function f. Let c be a cost function such that for all $\widetilde{P}$ and for all inputs x $c(\widetilde{P}, x) \leq c(f) \implies \widetilde{P} \in \mathcal{C}$, then (P, V) is a $(\mathsf{neg}(|x|, \mathsf{poly}(|x|))$-sequentially composable rational proof for f.

Proof. Let $\widetilde{P}$ be a prover such that $c(\widetilde{P}, x) < c(f)$ and such that $\Pr[\mathsf{out}(\widetilde{P}, V) \neq f(x)] = 1$. Consider the interaction between $\widetilde{P}$ and V. And let V reward $\widetilde{P}$ with $R = \mathsf{poly}(|x|)$ if V accepts and with 0 otherwise. The probability of $\widetilde{P}$ cheating successfully is negligible (as it must be in the class $\mathcal{C}$ by hypothesis). The result follows by Corollary 3.2.3. □

Bibliography

[1] B Applebaum, Y Ishai, and E Kushilevitz. "Cryptography in NC0". In: *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*. 2004, pp. 166–175.

[2] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "From secrecy to soundness: Efficient verification via secure computation". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 152–163.

[3] Yonatan Aumann and Yehuda Lindell. "Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries". In: *J. Cryptology* 23.2 (2010), pp. 281–343.

[4] Pablo Daniel Azar and Silvio Micali. "Rational proofs". In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM. 2012, pp. 1017–1028.

[5] Pablo Daniel Azar and Silvio Micali. "Super-efficient rational proofs". In: *Proceedings of the fourteenth ACM conference on Electronic commerce*. ACM. 2013, pp. 29–30.

[6] László Babai. "Trading group theory for randomness". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. ACM. 1985, pp. 421–429.

[7] Marshall Ball et al. "Average-Case Fine-Grained Hardness." In: *Electronic Colloquium on Computational Complexity (ECCC)*. Vol. 24. 2017, p. 39.

[8] Mira Belenkiy et al. "Incentivizing outsourced computation". In: *Proceedings of the ACM SIGCOMM 2008 Workshop on Economics of Networked Systems, NetEcon 2008, Seattle, WA, USA, August 22, 2008*. 2008, pp. 85–90.

[9] Joan Boyar and Rene C Peralta. "A depth-16 circuit for the AES S-box". In: *IACR Cryptology ePrint Archive* 2011.IACR Cryptology ePrint Archive (2011).

[10] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient fully homomorphic encryption from (standard) LWE". In: *SIAM Journal on Computing* 43.2 (2014), pp. 831–871.

[11] Christian Cachin and Ueli Maurer. "Unconditional security against memory-bounded adversaries". In: *Advances in Cryptology — CRYPTO '97*. Ed. by Burton S. Kaliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 292–306. ISBN: 978-3-540-69528-8.

[12] Matteo Campanelli and Rosario Gennaro. "Efficient Rational Proofs for Space Bounded Computations". In: *International Conference on Decision and Game Theory for Security*. Springer. 2017, pp. 53–73.

[13] Matteo Campanelli and Rosario Gennaro. *Fine-Grained Secure Computation*. Cryptology ePrint Archive, Report 2018/297. `https://eprint.iacr.org/2018/297`. 2018.

[14] Matteo Campanelli and Rosario Gennaro. “Sequentially composable rational proofs”. In: *International Conference on Decision and Game Theory for Security*. Springer. 2015, pp. 270–288.

[15] Jing Chen, Samuel McCauley, and Shikha Singh. “Rational proofs with multiple provers”. In: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. ACM. 2016, pp. 237–248.

[16] Kai-Min Chung, Yael Tauman Kalai, and Salil P Vadhan. “Improved Delegation of Computation Using Fully Homomorphic Encryption.” In: *CRYPTO*. Vol. 6223. Springer. 2010, pp. 483–501.

[17] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[18] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Practical verified computation with streaming interactive proofs”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM. 2012, pp. 90–112.

[19] Sergio De Agostino and Riccardo Silvestri. “Bounded size dictionary compression: SC_k-completeness and NC algorithms”. In: *Information and Computation* 180.2 (2003), pp. 101–112.

[20] Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. “Fine-grained Cryptography”. In: *Annual Cryptology Conference*. Springer. 2016, pp. 533–562.

[21] Cynthia Dwork and Moni Naor. “Pricing via processing or combatting junk mail”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 139–147.

[22] Cynthia Dwork, Moni Naor, and Amit Sahai. “Concurrent zero-knowledge”. In: *J. ACM* 51.6 (2004), pp. 851–898.

[23] Rosario Gennaro, Craig Gentry, and Bryan Parno. “Non-interactive verifiable computing: Outsourcing computation to untrusted workers”. In: *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 465–482.

[24] Rosario Gennaro et al. “Quadratic Span Programs and Succinct NIZKs without PCPs.” In: *EUROCRYPT*. Vol. 7881. Springer. 2013, pp. 626–645.

[25] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.

[26] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge university press, 2009.

[27] Oded Goldreich and Guy N Rothblum. “Simple doubly-efficient interactive proof systems for locally-characterizable sets”. In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[28] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. “Delegating computation: interactive proofs for muggles”. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 2008, pp. 113–122.

[29] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof systems”. In: *SIAM Journal on computing* 18.1 (1989), pp. 186–208.

[30] Shafi Goldwasser et al. “Verifying and decoding in constant depth”. In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, pp. 440–449.

[31] Siyao Guo et al. “Rational arguments: single round delegation with sublinear verification”. In: *Proceedings of the 5th conference on Innovations in theoretical computer science*. ACM. 2014, pp. 523–540.

[32] Siyao Guo et al. “Rational sumchecks”. In: *Theory of Cryptography*. Springer, 2016, pp. 319–351.

[33] Torben Hagerup. “Fast Parallel Generation of Random Permutations”. In: *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*. New York, NY, USA: Springer-Verlag New York, Inc., 1991, pp. 405–416. ISBN: 0-387-54233-7. URL: `http://dl.acm.org/citation.cfm?id=111713.111744`.

[34] Joseph Y Halpern and Rafael Pass. “I don’t want to think about it now: Decision theory with costly computation”. In: *arXiv preprint arXiv:1106.2657* (2011).

[35] Johan Hastad. “One-way permutations in NC0”. In: *Information Processing Letters* 26.3 (1987), pp. 153–155.

[36] Russell Impagliazzo. “A personal view of average-case complexity”. In: *Structure in Complexity Theory Conference, 1995., Proceedings of Tenth Annual IEEE*. IEEE. 1995, pp. 134–147.

[37] Yuval Ishai and Eyal Kushilevitz. “Randomizing polynomials: A new representation with applications to round-efficient secure computation”. In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE. 2000, pp. 294–304.

[38] Yuval Ishai and Eyal Kushilevitz. “Randomizing polynomials: A new representation with applications to round-efficient secure computation”. In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE. 2000, pp. 294–304.

[39] David S Johnson. “The NP-completeness column: The many limits on approximation”. In: *ACM Transactions on Algorithms (TALG)* 2.3 (2006), pp. 473–489.

[40] Yael Tauman Kalai, Ran Raz, and Ron D Rothblum. “How to delegate computations: the power of no-signaling proofs”. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM. 2014, pp. 485–494.

[41] Richard M Karp, Eli Upfal, and Avi Wigderson. “Constructing a perfect matching is in random NC”. In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. ACM. 1985, pp. 22–32.

[42] Subhash Khot and Ashok Kumar Ponnuswami. “Better inapproximability results for max-clique, chromatic number and min-3lin-deletion”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 226–237.

[43] Loi Luu et al. “Demystifying incentives in the consensus computer”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 706–719.

[44] Konstantin Makarychev, Rajsekar Manokaran, and Maxim Sviridenko. “Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 594–604.

[45] Yossi Matias and Uzi Vishkin. “Converting High Probability into Nearly-constant Time - with Applications to Parallel Hashing”. In: *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*. STOC ’91. New York, NY, USA: ACM, 1991, pp. 307–316. ISBN: 0-89791-397-3. DOI: `10.1145/103418.103453`. URL: `http://doi.acm.org/10.1145/103418.103453`.

[46] Ralph C Merkle. “Secure communications over insecure channels”. In: *Communications of the ACM* 21.4 (1978), pp. 294–299.

[47] Noam Nisan. “Pseudorandom generators for space-bounded computation”. In: *Combinatorica* 12.4 (1992), pp. 449–461.

[48] Periklis A Papakonstantinou. “Constructions, lower bounds, and new directions in Cryptography and Computational Complexity”. PhD thesis. University of Toronto, 2010.

[49] Alexander A Razborov. “Lower bounds on the size of bounded depth circuits over a complete basis with logical addition”. In: *Mathematical Notes of the Academy of Sciences of the USSR* 41.4 (1987), pp. 333–338.

[50] Omer Reingold, Ron Rothblum, and Guy Rothblum. “Constant-Round Interactive Proofs for Delegating Computation”. In: *Proceedings of the Forty-Eighth Annual ACM on Symposium on Theory of Computing*. ACM. 2016, To appear.

[51] Guy N Rothblum. “Delegating computation reliably: paradigms and constructions”. PhD thesis. Massachusetts Institute of Technology, 2009.

[52] Tomas Sander, Adam Young, and Moti Yung. “Non-Interactive CryptoComputing For NC1”. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society. 1999, p. 554.

[53] Michael Walfish and Andrew J Blumberg. “Verifying computations without reexecuting them”. In: *Communications of the ACM* 58.2 (2015), pp. 74–84.