Delegating RAM Computations

Yael Tauman Kalai* Omer Paneth†

October 6, 2015

Abstract

In the setting of cloud computing a user wishes to delegate its data, as well as computations over this data, to a cloud provider. Each computation may read and modify the data, and these modifications should persist between computations. Minding the computational resources of the cloud, delegated computations are modeled as RAM programs. In particular, the delegated computations' running time may be sub-linear, or even exponentially smaller than the memory size.

We construct a two-message protocol for delegating RAM computations to an untrusted cloud. In our protocol, the user saves a short digest of the delegated data. For every delegated computation, the cloud returns, in addition to the computation's output, the digest of the modified data, and a proof that the output and digest were computed correctly. When delegating a T-time RAM computation M with security parameter k, the cloud runs in time $\mathsf{T} \cdot \mathrm{poly}(k)$ and the user in time $\mathrm{poly}(|M|, \log \mathsf{T}, k)$. Our protocol is secure assuming super-polynomial hardness of the Learning with Error (LWE) assumption. Security holds even when the delegated computations are chosen adaptively as a function of the data and output of previous computations.

We note that RAM delegation schemes are an improved variant of memory delegation schemes [Chung et al. CRYPTO 2011]. In memory delegation, computations are modeled as Turing machines, and therefore, the cloud's work always grows with the size of the delegated data.

*Microsoft Research. Email: yael@microsoft.com.

†Boston University. Email: omer@bu.edu. Supported by the Simons award for graduate students in theoretical computer science and an NSF Algorithmic foundations grant 1218461.

Contents

1 Introduction

In recent years, with the growing popularity of cloud computing platforms, more and more users store data and run computations on the cloud. This raises many concerns. As cryptographers, our first concern is that of secrecy: users may wish to hide their confidential data and computations from the cloud. But perhaps a more fundamental concern is that of *integrity*: ensuring that the cloud is doing what it is suppose to do. In this paper we focus on the latter.

We ask the following question: how can a cloud provider convince a user that a delegated computation was performed correctly? We believe that the adoption of cloud computing services depends on the existence of such mechanisms. Indeed, even if not every computation is explicitly checked, the mere ability to check computations may be desirable.

RAM delegation. we model the above problem as follows. Initially the user owns some memory D containing the data it wishes to delegate. In order to verify the correctness of future computations over this memory, the user must save some short digest of the memory D. We therefore allow the user to pre-process the memory once, before delegating it, and compute a digest d. We also allow the cloud to pre-process the memory before storing it. During this pre-processing the cloud can compute auxiliary information that will be stored together with the memory and used to construct proofs efficiently.

To compute on the memory, the user specifies a program M and sends its description to the cloud. We model the program M as a RAM program. We believe that this is the most realistic choice when the outsourced memory is very large and the computation may not access it all.[1] The cloud sends back to the user the output y of the program M when executed on the memory D. The user can delegate multiple computations sequentially where each computation may modify the memory. We require that the state of the memory persists between computations. Therefore, after every computation, the cloud sends back to the user, together with the output y, the new digest $\mathsf{d}_{\mathsf{new}}$ coresponding to the new digest of the memory.

The cloud also provides a proof that the output y and the a new digest $\mathsf{d}_{\mathsf{new}}$ are correct with respect to the program M and the digest d of the original memory. We require that this proof proceeds in two messages, namely, together with the program M, the user sends a challange ch, and together with y and $\mathsf{d}_{\mathsf{new}}$, the cloud sends a proof pf. Thus, the proof of correctness does not require additional rounds of interaction. We refer to such a protocol as a *two-message delegation scheme for RAM computations.*

1.1 Our Results

We construct a two-message delegation scheme for RAM computations based on the Learning with Errors (LWE) assumption.

Efficiency. For security parameter k and for initial memory of size n such that $n < 2^k$, the user's and the cloud's pre-processing time is $n \cdot \mathrm{poly}(k)$, and the digest is of size $\mathrm{poly}(k)$. If the running time of the delegated RAM program is T (we assume that $\mathsf{T} < 2^k$), then the running time of the cloud is $\mathsf{T} \cdot \mathrm{poly}(k)$. The communication complexity of the proof, and the time it takes the user to generate a challenge and verify a proof are $\mathrm{poly}(k)$, and are independent of the computation time. Namely, our delegation scheme increases the user's and the cloud's computation by a factor of $\mathrm{poly}(k)$, independent of T, compared to delegating the computation insecurely.

Adaptive soundness. The soundness of our scheme holds even if the adversary (acting as the cloud) can choose the program to be delegated adaptively depending on the memory and on the outcome of previously delegated computations. This feature is especially important in applications where the pre-processing step is performed once and then used and reused to delegate many computations over time.

[1] For example, consider the setting where the user wishes to simply retrieve an element from the outsourced database D. We would like the runtime of the user in this case to be proportional to $\log |D|$, as opposed to proportional to D.

We emphasize that our protocol may not be sound if the adversary chooses the program adaptively depending on the user's challenge ch. Such a stronger notion of adaptive soundness can be obtained based on sub-exponential hardness assumptions following a standard complexity-leveraging argument.

Public pre-process step. In a two-message delegation scheme for RAM computations the user must pre-process the memory before delegating it. We emphasize that in our delegation scheme the pre-processing step is *public* – it does not require any secret randomness. In particular, the user is not required to keep any secret state between computations. This feature also allows a single execution of the pre-processing step to serve multiple users, as long as they all trust the generated memory digest.[2]

Another feature of our scheme is that the efficiency of the pre-processing step only depends on the *initial* memory size and does not depend on the amount of memory required to execute future computations. In particular, if there is no initial memory to delegate, the pre-processing step can be skipped.[3]

Informal Theorem 1 (See Theorem 8)**.** *There exists a two-message delegation scheme for RAM computations, with efficiency, adaptive soundness and public pre-processing, as described above, assuming the existence of a collision resistant hash family that is sub-exponentially secure and assuming that the LWE problem (with security parameter k) is hard to break in time quasi-polynomial in T, where T is an upper bound on the running time of the delegated computations.*

We note that the existence of a sub-exponentially secure collision resistant hash family follows from the sub-exponential hardness of the LWE problem.

On the necessity of cryptographic assumptions. Since the user does not store its memory locally, and only stores a short digest, we cannot hope to get information-theoretic soundness. An all powerful malicious cloud can always cheat by finding a fake memory D' with the same digest as the original memory, and perform computations using the fake memory. Therefore, the soundness of our scheme must rely on some hardness assumption (such as the hardness of finding digest collisions).

On delegation with secrecy. Our delegation protocol does not achieve secrecy. That is, it does not hide the user's data and computations from the cloud. One method for achieving secrecy is to execute the entire delegation protocol under fully-homomorphic encryption. However, this method is not applicable when delegating RAM computations, since it increases the cloud's running time proportionally to the size of the entire memory.

1.2 Previous Work

We compare our result with previous results on delegating computation in various models based on various computational assumptions.

1.2.1 Delegating Non-Deterministic Computations

Previous works constructed delegation schemes for non-deterministic computations in the random oracle model or based on strong "knowledge" assumptions. As we observe in this work (see Section 1.3), any delegation scheme for non-deterministic computations, combined with a collision-resistant hash function, can be used to delegate RAM computations.

The random oracle model. Based on the interactive arguments of Kilian [Kil92], Micali [Mic94] gave the first construction of a non-interactive delegation scheme in the random oracle model. Micali's

[2] When different users delegate different computations that may *change* the memory, there should be an external mechanism to synchronize these computations, and make sure that every computation is verified with respect to the most recent digest of the memory.

[3] In fact, we can always replace the pre-processing step with an initial delegation round where the user delegates a program that initializes the memory.

scheme supports non-deterministic computations and can therefore be used to delegate RAM computations assuming also the existence of a collision-resistant hash family.[4] The main advantage of Micali's scheme over the scheme presented in this work is that it is completely non-interactive (it requires one message rather than two). In particular, Micali's scheme is also *publicly verifiable*. However, our scheme can be proven secure in the standard model based on standard cryptographic assumptions.

Knowledge assumptions. In a sequence of recent works, non-interactive (one message) delegation schemes in the *common reference string* (CRS) model, were constructs based on strong and non-standard "knowledge" assumptions such as variants of the Knowledge of Exponent assumption [Gro10, Lip12, DFH12, GGPR13, BCI$^+$13, BCCT13, BCC$^+$14]. These schemes support non-deterministic computations and can therefore be used to delegate RAM computations. Some of the above schemes are also publicly verifiable (the user does need any secret trapdoor on the CRS). The main advantage of our scheme is that it can be based on standard cryptographic assumptions.

1.2.2 Indistinguishability Obfuscation

Several recent results construct non-interactive (one message) delegation schemes for RAM computations in the CRS model based on indistinguishability obfuscation [GHRW14, BGL$^+$15, CHJV15, CH15, CCC$^+$15]. Next we compare our scheme to the obfuscation based schemes.

The advantage of their schemes is that they achieve secrecy. In fact, they construct stronger objects such as garbling and obfuscation schemes for RAM computations. In addition, their schemes are publicly verifiable. The advantages of our scheme, compared to the obfuscation based schemes, are the following:

Assumptions. Our scheme is based on the hardness of the LWE problem – a standard and well studied cryptographic assumption. In particular, the LWE problem is known to be as hard as certain worst-case lattice problems.

Adaptivity. In our scheme security holds even against an adaptive adversary that chooses the delegated computations as a function of the delegated memory. In contrast, the obfuscation based schemes only have static security. That is, in the security proof all future delegated computations must be fixed before the memory is delegated. We note that using complexity leveraging and sub-exponential hardness assumptions it is possible to prove that obfuscation based schemes are secure against a *bounded* number of adaptively chosen computations, where the bound on the number of computations depends on the size of the CRS.

Public pre-processing. In our scheme the pre-processing step is completely public while in the obfuscation based schemes the pre-processing step requires private randomness. However, as mentioned above, the obfuscation based schemes are publicly verifiable.

1.2.3 Learning with Errors

We review existing delegation protocols based on the hardness of the LWE problem. These protocols are less efficient then our delegation protocols for RAM computations.

Deterministic Turing machine delegation. The work of [KRR14] gives a two-message delegation scheme for deterministic Turing machine computations based on the quasi-polynomial hardness of the LWE problem. The main differences between delegation of RAM computations and delegation of deterministic Turing machine computations are as follows:

[4]The solution described in Section 1.3 makes non-black-box use of the collision-resistant hash function, and therefore we cannot replace the hash function with the random oracle.

1. In deterministic Turing machine delegation, the user needs to save the entire memory (thought of as the input to the computation), while in RAM delegation, the user only needs to save a short digest of the memory.

2. In deterministic Turing machine delegation, the cloud's running time depends on the running time of the computation when described as a Turing machine, rather than a RAM program. In particular, the cloud's running time always grows with the memory size, even if the delegated computation does not access the entire memory.

We mention that our scheme has better asymptotic efficiency than the scheme of [KRR14] even for Turing machine computations. In our scheme The could's running time depends only linearly on the running time of the delegated computation and not quadratically as in [KRR14].

Memory delegation. As mentioned in [KRR14], the techniques of Chung et al. [CKLR11] can be used to convert the [KRR14] scheme into a memory delegation scheme that overcomes the first difference above, but not the second one.

Fully-homomorphic signatures. The work of Gorbunov et al. [GVW15] on fully-homomorphic signatures gives a non-interactive, publicly verifiable protocol in the CRS model, overcoming both differences above. However, while their protocol has small communication, the user's work is still proportional to computation's running time. Additionally, their protocol does not support computations that write to the memory.

Proofs of proximity. Finally, we mention a recent line of works on proofs of proximity [RVW13, GR15, KR15, GGR15]. These proofs can be verified much faster than the size of the memory, however, unlike in RAM delegation, in their model the user does not get to pre-process the memory. Instead the user has oracle access to the memory during proof verification. In proofs of proximity the user is only convinced that the computation output is consistent with some memory that is close to the real memory. Additionally, in proofs of proximity the verification takes time at least $\Omega(\sqrt{n})$ where n is the memory size [KR15].

1.3 Technical Overview

We start with a high level description of our scheme.

Pre-processing. In the pre-processing step, the user computes a hash-tree [Mer87] over the memory D and saves the root of this tree as the digest d. The cloud also pre-processes the delegated memory D by computing the same hash-tree and stores the entire tree. The hash-tree allows the cloud to efficiently access the memory in an "authenticated" way. Specifically, the cloud performs the following operations:

1. Read a bit from memory.

2. Write a bit to memory, update the hash tree, and obtain a new digest.

The cloud can then compute a short certificate (in the form of an authenticated path), authenticating the value of the bit read or the value of the updated digest. The time required to access the memory and compute the certificate depends only logarithmically on the memory size.

Emulated computations and their transcript. When the user delegates a computation given by a RAM program M, the cloud starts by emulating the execution of M on the memory D as described in [BEG$^+$91]. Whenever M accesses the memory, the cloud performs an authenticated memory access via the hash tree. When the emulation of M terminates, the cloud obtains the program output y and the updated memory digest $\mathsf{d}_{\mathsf{new}}$. The cloud also compiles a *transcript* of the memory accessed during the computation. This transcript contains an ordered list of M's memory accesses. For every memory access, the transcript contains the memory location, the bit that was read or written, the new memory

digest (in case the memory changed), and the certificate of authenticity. This transcript allows to "re-execute" the computation of the program M and obtain y and $\mathsf{d}_{\mathsf{new}}$, without accessing the memory D directly. Moreover, it is computationally hard to find a valid transcript (containing only valid certificates) that yeilds the wrong output or digest $(y', \mathsf{d}'_{\mathsf{new}}) \neq (y, \mathsf{d}_{\mathsf{new}})$. For security parameter k and a RAM program M executing in time $\mathsf{T} \leq 2^k$, the time to generate the transcript and to re-execute the program based on the transcript is $\mathsf{T} \cdot \mathrm{poly}(k)$.

Proof of correctness. After emulating the execution of M, the cloud sends the output y and the new digest $\mathsf{d}_{\mathsf{new}}$ to the user. The cloud also proves to the user that it knows a valid computation transcript which is consistent with y and $\mathsf{d}_{\mathsf{new}}$. More formally, we consider a non-deterministic Turing machine TVer that accepts an input tuple $(M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}})$ if and only if there exists a valid transcript Trans with respect to d such that the emulation of the program M with Trans produces the output y and the digest $\mathsf{d}_{\mathsf{new}}$.

Proving knowledge of a witness Trans that makes TVer accept $(M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}})$ requires a delegation scheme supporting *non-deterministic computations*. The problem with this approach is that currently, two-message delegation schemes for non-deterministic computations are only known in the random oracle model or based on strong knowledge assumptions (see Section 1.2). However, it turns out that for the specific computation TVer, we can construct a two-message delegation scheme based on standard cryptographic assumptions.

Re-purposing the KRR proof system. Our solution is based on the delegation scheme of Kalai, Raz and Rothblum [KRR14]. While in general, their proof system only supports deterministic computations, we extend their security proof so it also applies to non-deterministic computations of a certain form.

We start with a brief overview of the [KRR14] proof system and explain why it does not support general non-deterministic computations. Then we describe the extended security proof and the type of non-deterministic computations it does support.

The [KRR14] proof system can be used to prove that a deterministic Turing machine M is accepting. We describe the proof system and its analysis in two steps. In the first step, M is translated into a 3-SAT formula ϕ that is satisfiable if and only if M is accepting. The cloud then convinces the user that the formula ϕ satisfies a relaxed notion of satisfiability called *local satisfiability*. In the second step, the specific structure of the formula ϕ is exploited to prove that if ϕ is locally satisfiable it must also be satisfiable.

Local satisfiability. Unlike full-fledged satisfiability, the notion of local satisfiability only considers assignments to ℓ variables at a time, where ℓ is a locality parameter that may be much smaller than the total number of variables in the formula. Formally, we say that a 3-SAT formula ϕ is ℓ-locally satisfiable if for every set Q of ℓ variables there exists a distribution D_Q over assignments to the variables in Q such that the following conditions are satisfied:

Everywhere local consistency. For every set Q of ℓ variables, a random assignment in D_Q satisfies all local constraints in ϕ over the variables in Q with high probability.

No-signaling. For every set Q of ℓ variables and for every subset $Q' \subseteq Q$, the distribution of an assignment sample from D_Q restricted to the variables in Q' is independent of the other variables in $Q \setminus Q'$.

From local satisfiability to full-fledged satisfiability. In the [KRR14] proof system, ℓ is a fixed polynomial in the security parameter, independent of the size of the formula ϕ (the communication complexity of the proof grows with ℓ). In this setting, local satisfiability does not generally imply full-fledged satisfiability. However, the analysis of [KRR14] exploits the specific structure of ϕ to go from local satisfiability to full-fledged satisfiability. The proof of this step crucially relies on the fact that the formula ϕ describes a deterministic computation. We show how to extend this proof for non-deterministic computations of a specific form.

Roughly, we require that (computationally) there exists a unique "correct" witness that can be verified *locally*. Namely, for any proposed witness (that can be found efficiently) and any bit of this proposed witness, it is possible to verify that the value of this bit agrees with the correct witness in time that is independent of the running time of the entire computation.

More on the analysis of KRR. We describe the argument of [KRR14] and explain why it fails for non-deterministic computations. To go from local satisfiability to full-fledged satisfiability, the proof of [KRR14] relies on the fact that the formula ϕ describing an accepting deterministic computation has a unique satisfying assignment. We call this the *correct* assignment to ϕ. The rest of the proof uses the fact that the variables of ϕ can be partitioned into "layers" such that variables in the i-th layer correspond to the computation's state immediately before the i-th computation step. The proof proceeds by induction over the layers. In the inductive step we assume that local assignments to any ℓ variables in the i-th layer are correct (agrees with the correct assignment) with high probability and prove that the same holds for the $(i+1)$-st layer. Indeed, if the local assignment to some set of ℓ variables in the $(i+1)$-st layer is correct with a significantly lower probability, the special structure of ϕ and the no-signaling property of the assignments can be used to argue that there must exist a set of ℓ variables whose assignment violates ϕ's local constraints with some significant probability.

Non-deterministic computations. The above argument does not extend to non-deterministic computations, since the notion of a "correct" assignment is not well defined in this setting. Moreover, even if there is a unique witness that makes the computation accept, and we consider the correct assignment defined by this witness, the above argument still fails. The issue is that even if every local assignment to any set of variables in the i-th layer is correct, there could still be more than one assignment to variables in the $(i+1)$-st layer satisfying all of ϕ's local constraints.

We show how to overcome this problem for non-deterministic computations where (computationally) there exists a unique "correct" witness that can be verified *locally*, as described above. Consider for example the computation of the Turing machine TVer on input $(M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}})$ where d is the digest of the initial memory D. The (computationally) unique witness for this computation is a transcript of the program execution that can be verified locally – one step at a time.

In more details, let Trans be the correct transcript defined by the execution of M on memory D. Let ϕ be the formula describing the computation of $\mathsf{TVer}(M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}})$. We prove that any acceptng local assignment to variables of ϕ must agree with the global correct assignment to ϕ defined by the execution of TVer with the (well defined) transcript Trans. As in the case of deterministic computations, we partition ϕ's variables into layers. In the i-th inductive step we assume that local assignments to any ℓ variables in the i-th layer are correct with high probability. If the local assignment to some set of ℓ variables in the $(i+1)$-th layer is correct with a significantly lower probability then we prove that the assignment must describe an incorrect transcript. Since both the correct transcript and the incorrect one contain valid certificates, we can use these certificates to break the security of the hash tree.

Multi-prover arguments. The presentation of the construction in [KRR14], as well as the presentation in the body of this work, goes through the intermediate step of constructing a no-signaling multi-prover proof-system. In more details, [KRR14] first construct a no-signaling multi-prover interactive proof for local-satisfiability. They then leverage local-satisfiability to prove full-fledged satisfiability, resulting in a no-signaling multi-prover interactive proof (with unconditional soundness) for deterministic computations. Finally, they transform any no-signaling multi-prover interactive proof into a delegation scheme assuming fully-homomorphic encryption.

Our construction follows the same blueprint. We first construct a no-signaling multi-prover interactive argument for RAM computations, and then transform it into a delegation scheme. Unlike in [KRR14], the soundness of our multi-prover arguments is conditional on the existence of collision-resistent hashing. We note that for RAM delegation, computational assumptions are necessary even in the multi-prover model.

2 Tools and Definitions

2.1 Notation

For sets B, S, we denote by B^S the set of vectors of elements in B indexed by the elements of S. That is, every vector $\mathbf{a} \in B^S$ is of the form $\mathbf{a} = (\mathsf{a}_i \in B)_{i \in S}$. For a vector $\mathbf{a} \in B^S$ and a subset $Q \subseteq S$, we denote by $\mathbf{a}[Q] \in B^Q$ the vector that contains only the elements in $\mathbf{a}$ with indices in Q, that is, $\mathbf{a}[Q] = (\mathsf{a}_i)_{i \in Q}$.

2.2 RAM computation

We consider the standard model of RAM computation where a program M can access an initial memory string $D \in \{0,1\}^n$. For an input x, we denoted by $M^D(x)$ an execution of the program M with input x and initial memory D. For a bit $y \in \{0,1\}$ and for a string $D_{\mathsf{new}} \in \{0,1\}^n$ we also use the notation $y \leftarrow M^{(D \to D_{\mathsf{new}})}(x)$ to denote that y is the output of the program M on input x and initial memory D, and D_{new} is the final memory string after the execution. For simplicity we think only of RAM programs that output a single bit.[5] The computation of M is carried out one step at a time by a CPU algorithm STEP. STEP is a polynomial-time algorithm that takes as input a description of a program M, an input x, a state of size $O(\log n)$, and a bit that was supposedly read from memory, and it outputs a quadruple

$$(\mathsf{state}_{\mathsf{new}}, i^{\mathsf{r}}, i^{\mathsf{w}}, b^{\mathsf{w}}) \leftarrow \mathsf{STEP}(M, x, \mathsf{state}, b^{\mathsf{r}}) \ ,$$

where $\mathsf{state}_{\mathsf{new}}$ is the updated state, i^{r} denotes the location in memory to be read next, the location i^{w} denotes the location in memory to write to next, and the bit b^{w} denotes the bit to be written in location i^{w}. The execution $M^D(x)$ proceeds as follows. The program starts with some empty initial state state_1. By convention we set the first memory location read by the program to be $i^{\mathsf{r}}_1 = 1$. Starting from $j = 1$, the j-th execution step proceeds as follows:

1. Read from memory the bit $b^{\mathsf{r}}_j \leftarrow D[i^{\mathsf{r}}_j]$.
2. Compute $(\mathsf{state}_{j+1}, i^{\mathsf{r}}_{j+1}, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}) \leftarrow \mathsf{STEP}(M, x, \mathsf{state}_j, b^{\mathsf{r}}_j)$.
3. Write a bit to memory $D[i^{\mathsf{w}}_{j+1}] \leftarrow b^{\mathsf{w}}_{j+1}$. (If $i^{\mathsf{w}}_{j+1} = \perp$ no writing is performed in this step.)

The execution terminates when the program STEP outputs a special terminating state. We assume that the terminating state includes the value of the output bit y. Note that after the last step was executed and an output has been produced, the memory is written to one last time. We say that a machine M is *read only*, if for every $(x, \mathsf{state}, b^{\mathsf{r}})$, $\mathsf{STEP}(M, x, \mathsf{state}, b^{\mathsf{r}})$ outputs $(\mathsf{state}_{\mathsf{new}}, i^{\mathsf{r}}, i^{\mathsf{w}}, b^{\mathsf{w}})$ where $i^{\mathsf{w}} = \perp$.

Remark 2.1 (Space complexity of STEP)**.** *We assume without loss of generality that the RAM program M reads the input x once and copies it to memory. Therefore the space complexity of the algorithm* STEP *is* $\mathrm{polylog}(n)$.

2.3 Hash Tree

Let $D \in \{0,1\}^n$ be a string. Let k be a security parameter such that $n < 2^k$.

A hash-tree scheme consists of algorithms:

$$(\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite}) \ ,$$

with the following syntax and efficiency:

[5] A program that outputs multipul bits can be simulated by executing several programs in parallel, or by writing the output directly to the memory.

- $\mathsf{HT.Gen}(1^k) \to \mathsf{key}$:
  A randomized polynomial-time algorithm that outputs a hash key, denoted by key.

- $\mathsf{HT.Hash}(\mathsf{key}, D) \to (\mathsf{tree}, \mathsf{rt})$:
  A deterministic polynomial-time algorithm that outputs a hash tree denoted by tree, and a hash root rt of size $\mathrm{poly}(k)$ (we assume that both strings tree and rt include key).

- $\mathsf{HT.Read}^{\mathsf{tree}}(i^{\mathsf{r}}) \to (b^{\mathsf{r}}, \mathsf{pf})$:
  A deterministic read-only RAM program that accesses the initial memory string tree, runs in time $\mathrm{poly}(k)$, and outputs a bit, denoted by b^{r}, and a proof, denoted by pf.

- $\mathsf{HT.Write}^{\mathsf{tree}}(i^{\mathsf{w}}, b^{\mathsf{w}}) \to (\mathsf{rt}_{\mathsf{new}}, \mathsf{pf})$:
  A deterministic RAM program that accesses the initial memory string tree, runs in time $\mathrm{poly}(k)$, and outputs a new hash root, denoted by $\mathsf{rt}_{\mathsf{new}}$, and a proof, denoted by pf.

- $\mathsf{HT.VerRead}(\mathsf{rt}, i^{\mathsf{r}}, b^{\mathsf{r}}, \mathsf{pf}) \to b$:
  A deterministic polynomial-time algorithm that outputs an acceptance bit b.

- $\mathsf{HT.VerWrite}(\mathsf{rt}, i^{\mathsf{w}}, b^{\mathsf{w}}, \mathsf{rt}_{\mathsf{new}}, \mathsf{pf}) \to b$:
  A deterministic polynomial-time algorithm that outputs an acceptance bit b.

Definition 2.2 (Hash-Tree)**.** *A hash-tree scheme*

$$(\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite}) \ ,$$

satisfies the following properties.

- *Completeness of Read. For every $k \in \mathbb{N}$ and for every $D \in \{0,1\}^n$ such that $n \leq 2^k$, and for every $i^{\mathsf{r}} \in [n]$*

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (b^{\mathsf{r}}, \mathsf{pf}) \leftarrow \mathsf{HT.Read}^{\mathsf{tree}}(i^{\mathsf{r}}); \\ 1 \leftarrow \mathsf{HT.VerRead}(\mathsf{rt}, i^{\mathsf{r}}, b^{\mathsf{r}}, \mathsf{pf}); \\ D[i^{\mathsf{r}}] = b^{\mathsf{r}} \end{array}\right] = 1 \ .$$

- *Completeness of Write. For every $k \in \mathbb{N}$ and for every $D \in \{0,1\}^n$ such that $n \leq 2^k$, for every $i^{\mathsf{w}} \in [n], b^{\mathsf{w}} \in \{0,1\}$, and for $D_{\mathsf{new}} \in \{0,1\}^n$ that is equal to the string D except that $D_{\mathsf{new}}[i^{\mathsf{w}}] = b^{\mathsf{w}}$*

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (\mathsf{tree}_{\mathsf{new}}, \mathsf{rt}_{\mathsf{new}}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D_{\mathsf{new}}); \\ (\mathsf{rt}'_{\mathsf{new}}, \mathsf{pf}) \leftarrow \mathsf{HT.Write}^{\mathsf{tree}}(i^{\mathsf{w}}, b^{\mathsf{w}}); \\ 1 \leftarrow \mathsf{HT.VerWrite}(\mathsf{rt}, i^{\mathsf{w}}, b^{\mathsf{w}}, \mathsf{rt}'_{\mathsf{new}}, \mathsf{pf}); \\ \mathsf{rt}'_{\mathsf{new}} = \mathsf{rt}_{\mathsf{new}} \end{array}\right] = 1 \ .$$

- *Soundness of Read. For every polynomial size adversary Adv there exists a negligible function μ such that for every $k \in \mathbb{N}$*

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (D, i^{\mathsf{r}}, b^{\mathsf{r}\prime}, \mathsf{pf}') \leftarrow \mathsf{Adv}(\mathsf{key}); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (b^{\mathsf{r}}, \mathsf{pf}) \leftarrow \mathsf{HT.Read}^{\mathsf{tree}}(i^{\mathsf{r}}); \\ (b^{\mathsf{r}\prime}, \mathsf{pf}') \neq (b^{\mathsf{r}}, \mathsf{pf}); \\ 1 \leftarrow \mathsf{HT.VerRead}(\mathsf{rt}, i^{\mathsf{r}}, b^{\mathsf{r}\prime}, \mathsf{pf}'); \end{array}\right] \leq \mu(k) \ .$$

- *Soundness of Write. For every poly-size adversary* Adv *there exists a negligible function* μ *such that for every* $k \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (D, i^{\mathsf{w}}, b^{\mathsf{w}}, \mathsf{rt}'_{\mathsf{new}}, \mathsf{pf}') \leftarrow \mathsf{Adv}(\mathsf{key}); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (\mathsf{rt}_{\mathsf{new}}, \mathsf{pf}) \leftarrow \mathsf{HT.Write}^{\mathsf{tree}}(i^{\mathsf{w}}, b^{\mathsf{w}}); \\ (\mathsf{rt}'_{\mathsf{new}}, \mathsf{pf}') \neq (\mathsf{rt}_{\mathsf{new}}, \mathsf{pf}); \\ 1 \leftarrow \mathsf{HT.VerWrite}(\mathsf{rt}, i^{\mathsf{w}}, b^{\mathsf{w}}, \mathsf{rt}'_{\mathsf{new}}, \mathsf{pf}'); \end{array}\right] \leq \mu(k) \ .$$

We say that the hash-tree scheme is (S, ϵ)*-secure, for a function* $S(k)$ *and a negligible function* $\epsilon(k)$*, if for every constant* $c > 0$*, the soundness of read and soundness of write properties hold for every adversary of size* $S(k)^c$ *with probability at most* $\epsilon(k)^c$*. We say that the hash-tree scheme has sub-exponential security if it is* $(2^{k^\delta}, 2^{-k^\delta})$*-secure for some constant* $\delta > 0$*.*

Remark 2.3 (Unique proofs in Definition 2.2)**.** *In the soundness properties of Definition 2.2 we make the strong requirement that it is hard to find two different proofs for any statement (even a correct one). This strong requirement simplifies the proof of Theorem 5, however the proof can be modified to rely on a weaker soundness requirement.*

Theorem 2 ([Mer87])**.** *A hash-tree scheme satisfying Definition 2.2 can be constructed from any family of collision-resistant hash functions. Moreover, the hash-tree scheme is sub-exponentially secure if the underlying collision-resistant hash family is sub-exponentially secure.*

2.4 Fully Homomorphic Encryption

A public-key fully homomorphic encryption (FHE) scheme consists of algorithms:

$$(\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec}) \ ,$$

with the following syntax and efficiency:

- $\mathsf{FHE.Gen}(1^k) \to (\mathsf{pk}, \mathsf{sk})$:
  A randomized polynomial-time algorithm that outputs a public and secret key pair.
- $\mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{m}) \to \mathsf{c}$:
  A randomized polynomial-time algorithm that outputs a ciphertext c encrypting the message m.
- $\mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{c}, C) \to \tilde{\mathsf{c}}$:
  A deterministic algorithm that evaluated a circuit C over the ciphertext c, and outputs an evaluated ciphertext $\tilde{\mathsf{c}}$ of length $\mathrm{poly}(k, n)$, where n is the output length of C (in particular, $|\tilde{\mathsf{c}}|$ is independent of $|C|$).
- $\mathsf{FHE.Dec}(\mathsf{sk}, \tilde{\mathsf{c}}) \to \mathsf{m}$:
  A deterministic polynomial-time algorithm that decrypts an evaluated ciphertext and outputs a message m.

Remark 2.4 (On denoting ciphertexts)**.** *We often denote by* $\hat{\mathsf{m}}$ *a ciphertext or an evaluated ciphertext that decrypts to the message* m

Definition 2.5 (Fully Homomorphic Encryption)**.** *A public-key fully homomorphic encryption scheme*

$$(\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec}) \ ,$$

satisfies the following properties.

- *Completeness. For every $k \in \mathbb{N}$, for every $\mathsf{m} \in \{0,1\}^*$ and for every circuit C taking inputs of length $|\mathsf{m}|$*

$$\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^k);\\ \mathsf{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk},\mathsf{m});\\ \tilde{\mathsf{c}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk},\mathsf{c},C);\\ C(\mathsf{m}) = \mathsf{FHE.Dec}(\mathsf{sk},\tilde{\mathsf{c}})\end{array}\right] = 1 \ .$$

- *Security. For every polynomial p and every polynomial size distinguisher D, there exists a negligible function μ such that for every security parameter $k \in \mathbb{N}$ and every pair of messages $\mathsf{m}_0, \mathsf{m}_1 \in \{0,1\}^{p(k)}$:*

$$\Pr\left[\begin{array}{l}b \leftarrow \{0,1\};\\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^k);\\ \mathsf{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk},\mathsf{m}_b);\\ b = D(\mathsf{pk},\mathsf{c})\end{array}\right] \leq \frac{1}{2} + \mu(k) \ .$$

We say that the encryption scheme is (S,ϵ)-secure, for a function $S(k)$ and a negligible function $\epsilon(k)$, if for every constant $c > 0$, the security property holds for every adversary of size $S(k)^c$ with distinguishing gap at most $\epsilon(k)^c$. We say that the encryption is quasi-polynomially secure if it is $(2^{\log^\delta(k)}, 2^{-\log^\delta(k)})$-secure for some constant $\delta > 1$.

2.5 Delegation for RAM Computations

Let M be a T-time RAM program, let $x \in \{0,1\}^m$ be an input to the program, and let $D \in \{0,1\}^n$ be some initial memory string. Let k be a security parameter such that $|M|, \mathsf{T}(m), n < 2^k$. A two-message delegation scheme for RAM computations consists of algorithms:

$$(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier}) \ ,$$

with the following syntax and efficiency:

- $\mathsf{ParamGen}(1^k) \to \mathsf{pp}$:
  A randomized polynomial-time algorithm that outputs public parameters pp.

- $\mathsf{MemGen}(\mathsf{pp}, D) \to (\mathsf{dt}, \mathsf{d})$:
  A deterministic polynomial-time algorithm that outputs the processed memory dt, and a digest of the memory d of size $\mathrm{poly}(k)$.

- $\mathsf{QueryGen}(1^k) \to (\mathsf{q}, \mathsf{st})$:
  A randomized polynomial-time algorithm that outputs a query q and a secret state st.

- $\mathsf{Output}^{\mathsf{dt}}(1^{\mathsf{T}(m)}, n, M, x) \to (y, \mathsf{d}_{\mathsf{new}}, \mathsf{Trans})$:
  A deterministic RAM program running in time $\mathsf{T}(m)\cdot\mathrm{poly}(k)$ that accesses the processed memory dt, and outputs the output bit y, and a new digest $\mathsf{d}_{\mathsf{new}}$ of size $\mathrm{poly}(k)$ and a computation transcript Trans.

- $\mathsf{Prover}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}) \to \mathsf{pf}$:
  A deterministic algorithm running in time $\mathsf{T}(m) \cdot \mathrm{poly}(k)$ that outputs a proof pf of size $\mathrm{poly}(k)$.

- $\mathsf{Verifier}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, \mathsf{pf}) \to b$:
  A deterministic algorithm running in time $m \cdot \mathrm{poly}(k)$ that outputs an acceptance bit b.

Remark 2.6 (Statement-independent queries)**.** *In the above, the queries generated by the algorithm* QueryGen *are independent of the program, the input and the memory digest. We could consider a more liberal definition that allows such a dependency, however, in our construction this is not needed.*

Remark 2.7 (Prover efficiency)**.** *In the above we require that the output of the computation, the new memory digest, and the proof are computed in time that is linear in the running time of the RAM computation. We could consider a more liberal definition allowing for arbitrary polynomial overhead, however, we achieve the stronger notion of linear overhead.*

Remark 2.8 (Verifier efficiency)**.** *We note that the dependence of the verification time on the input length m can be improved. In particular, in our construction, given oracle access to a* low-degree extension *encoding of the input x, the verifier's running time is* $\mathrm{poly}(k)$.

Remark 2.9 (The Output algorithm)**.** *In the above interface we separated the prover computation into two algorithms. The first algorithm,* Output*, accesses the memory, carries out the computation, and produces the output as well as a* transcript *of the computation. This transcript may include all the memory accessed during the RAM computation or any other information. We only restrict the size of the transcript to be related to the running time of the RAM computation. The second algorithm,* Prover*, is given the transcript and the challenge query and outputs the proof. This separation ensures that the memory locations accessed by the prover are independent of the challenge query. This property is used in the transformation in Section 5.*

Definition 2.10 (Two-Message Argument for RAM computations)**.** *A two-message delegation scheme* $(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Prover}, \mathsf{Verifier})$ *for RAM computations satisfies the following properties.*

- *Completeness. For every security parameter $k \in \mathbb{N}$, every T-time RAM program M, every input $x \in \{0,1\}^m$, every $D \in \{0,1\}^n$, and every (y, D_{new}) such that $\mathsf{T}(m), n \leq 2^k$ and $y \leftarrow M^{(D \to D_{\mathsf{new}})}(x)$*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ (y, \mathsf{d}_{\mathsf{new}}, \mathsf{Trans}) \leftarrow \mathsf{Output}^{\mathsf{dt} \to \mathsf{dt}_{\mathsf{new}}}(1^{\mathsf{T}(m)}, n, M, x); \\ \mathsf{pf} \leftarrow \mathsf{Prover}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, \mathsf{pf}); \\ (\mathsf{dt}'_{\mathsf{new}}, \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (\mathsf{dt}'_{\mathsf{new}}, \mathsf{d}'_{\mathsf{new}}) = (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \end{array}\right] = 1 \ .$$

- *Soundness. For every pair of polynomial-size adversaries $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ there exists a negligible function μ such that for every $k \in \mathbb{N}$*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y', \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y', \mathsf{d}'_{\mathsf{new}}); \\ (\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ \mathsf{pf} \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, \mathsf{q}); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y', \mathsf{d}'_{\mathsf{new}}), \mathsf{st}, \mathsf{pf}) \end{array}\right] \leq \mu(k) \ .$$

We say that the delegation scheme is (S, ϵ)-secure, for a function $S(k)$ and a negligible function $\epsilon(k)$, if for every constant $c > 0$, the soundness property holds for every pair of adversaries of size $S(k)^c$ with probability at most $\epsilon(k)^c$.

2.6 Multi-Prover Arguments for RAM Computations

Let ℓ be a polynomial, M be a T-time RAM program, let $x \in \{0,1\}^m$ be an input to the program, and let $D \in \{0,1\}^n$ be some initial memory string. Let k be a security parameter such that $|M|, \mathsf{T}(m), n < 2^k$. An *$\ell$-prover argument* for RAM computations consists of algorithms:

$$(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier}) \ ,$$

with the following syntax and efficiency:

- $\mathsf{ParamGen}(1^k) \to \mathsf{pp}$:
  A randomized polynomial-time algorithm that outputs public parameters pp.
- $\mathsf{MemGen}(\mathsf{pp}, D) \to (\mathsf{dt}, \mathsf{d})$:
  A deterministic polynomial-time algorithm that outputs the processed memory dt and a digest of the memory d of size $\mathrm{poly}(k)$.
- $\mathsf{QueryGen}(1^k) \to ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st})$:
  A randomized polynomial-time algorithm that outputs a set of $\ell = \ell(k)$ queries $(\mathsf{q}_1, \ldots, \mathsf{q}_\ell)$, and a secret state st.
- $\mathsf{Output}^{\mathsf{dt}}(1^{\mathsf{T}(m)}, n, M, x) \to (y, \mathsf{d}_{\mathsf{new}}, \mathsf{Trans})$:
  A deterministic RAM program running in time $\mathsf{T}(m) \cdot \mathrm{poly}(k)$ that accesses the processed memory dt, and outputs the output bit y, a new digest $\mathsf{d}_{\mathsf{new}}$ of size $\mathrm{poly}(k)$, and a computation transcript Trans.
- $\mathsf{Prover}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}) \to \mathsf{a}$:
  A deterministic algorithm running in time $\mathsf{T}(m) \cdot \mathrm{poly}(k)$ that outputs an answer a of size $\mathrm{poly}(k)$ to a single query q.
- $\mathsf{Verifier}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \to b$:
  A deterministic algorithm running in time $m \cdot \mathrm{poly}(k)$ that outputs an acceptance bit b.

Remark 2.11 (Statement-independent queries)**.** *In the above, the queries generated by the algorithm* QueryGen *are independent of the program, the input and the memory digest. We could consider a more liberal definition that allows such a dependency, however, in our construction this is not needed.*

Remark 2.12 (Prover efficiency)**.** *In the above we require that the output of the computation, the new memory digest, and the proof are computed in time that is linear in the running time of the RAM computation. We could consider a more liberal definition allowing for arbitrary polynomial overhead, however, we achieve the stronger notion of linear overhead.*

Remark 2.13 (Verification efficiency)**.** *We note that the dependence of the verification time on the input length m can be improved. In particular, in our construction, given oracle access to a* low-degree extension *encoding of the input x, the verifier's running time is* $\mathrm{poly}(k)$.

Remark 2.14 (The Output algorithm)**.** *In the above interface we separated the prover computation into two algorithms. The first algorithm,* Output*, accesses the memory, carries out the computation, and produces the output as well as a* transcript *of the computation. This transcript may include all the memory accessed during the RAM computation or any other information. We only restrict the size of*

the transcript to be related to the running time of the RAM computation. The second algorithm, Prover*, is given the transcript and a challenge query and outputs an answer. This separation ensures that the memory locations accessed by the prover are independent of the challenge queries. This property is used in the transformation in Section 5.*

Definition 2.15 (Multi-Prover Argument for RAM computations)**.** *Let $\ell = \ell(k)$ be a polynomial in the security parameter. An ℓ-prover argument system* $(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier})$ *for RAM computations satisfies the following properties.*

- *Completeness. For every security parameter $k \in \mathbb{N}$, every T-time RAM program M, every input $x \in \{0,1\}^m$, every $D \in \{0,1\}^n$, and every (y, D_{new}), such that $\mathsf{T}(m), n \leq 2^k$ and $y \leftarrow M^{(D \to D_{\mathsf{new}})}(x)$*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ (y, \mathsf{d}_{\mathsf{new}}, \mathsf{Trans}) \leftarrow \mathsf{Output}^{\mathsf{dt} \to \mathsf{dt}_{\mathsf{new}}}(1^{\mathsf{T}(m)}, n, M, x); \\ \forall i \in [\ell] : \mathsf{a}_i \leftarrow \mathsf{Prover}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}_i); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)); \\ (\mathsf{dt}'_{\mathsf{new}}, \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (\mathsf{dt}'_{\mathsf{new}}, \mathsf{d}'_{\mathsf{new}}) = (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \end{array}\right] = 1 \ .$$

- *Soundness. For every pair of polynomial-size adversaries $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ there exists a negligible function μ such that for every $k \in \mathbb{N}$ and for $\ell = \ell(k)$*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y', \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y', \mathsf{d}'_{\mathsf{new}}); \\ ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ \forall i \in [\ell] : \mathsf{a}_i \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, \mathsf{q}_i); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y', \mathsf{d}'_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] \leq \mu(k) \ .$$

  We say that the argument system is (S, ϵ)-secure, for a function $S(k)$ and a negligible function $\epsilon(k)$, if for every constant $c > 0$, the soundness property holds for every pair of adversaries of size $S(k)^c$ with probability at most $\epsilon(k)^c$.

2.7 No-Signaling Multi-Prover Arguments for RAM Computations

No signaling multi-prover arguments are multi-prover arguments, where the cheating provers are given extra power. In multi-prover arguments (or proofs), each prover answers its own query *locally*, without knowing anything about the queries that were sent to the other provers.

In the no-signaling model we allow the *malicious* provers' answers to depend on all the queries, as long as for any subset $Q \subset [\ell]$ and for every two query vectors $\mathbf{q}^1 = (\mathsf{q}_1^1, \ldots, \mathsf{q}_\ell^1)$ and $\mathbf{q}^2 = (\mathsf{q}_1^2, \ldots, \mathsf{q}_\ell^2)$, such that $\mathbf{q}^1[Q] = \mathbf{q}^2[Q]$, the corresponding vectors of answers $\mathbf{a}^1, \mathbf{a}^2$ (as random variables) satisfy that $\mathbf{a}^1[Q]$ and $\mathbf{a}^2[Q]$ are identically distributed. Intuitively, this means that the answers of the provers in the set Q do not contain information about the queries to the provers outside Q, except for the information that is already found in the queries to the provers in Q.

Definition 2.16. *For a set B and for $\ell \in \mathbb{N}$, we say that a pair of vectors of correlated random variables*

$$\mathbf{q} = (\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathbf{a} = (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \in B^{[\ell]} \ .$$

is no-signaling *if for every subset $Q \subset [\ell]$ and every two vectors $\mathbf{q}^1, \mathbf{q}^2$ in the support of $\mathbf{q}$ such that $\mathbf{q}^1[Q] = \mathbf{q}^2[Q]$, the random variables $\mathbf{a}[Q]$ conditioned on $\mathbf{q} = \mathbf{q}^1$ and $\mathbf{a}[Q]$ conditioned on $\mathbf{q} = \mathbf{q}^2$ are identically distributed.*

If these random are not identical, but rather, the statistical distance between them is at most δ, we say that the pair $(\mathbf{q}, \mathbf{a})$ is δ-no-signaling.

Definition 2.17. *An ℓ-prover argument system* $(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier})$ *for RAM computations is said to be sound against δ-no-signaling strategies (or provers) if the following (more general) soundness property is satisfied:*

For every pair of polynomial-size adversaries $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ satisfying a δ-no-signaling condition (specified below), there exists a negligible function μ such that for every $k \in \mathbb{N}$ and for $\ell = \ell(k)$:

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y', \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y', \mathsf{d}'_{\mathsf{new}}); \\ ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, (\mathsf{q}_1, \ldots, \mathsf{q}_\ell)); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y', \mathsf{d}'_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] \leq \mu(k) \ ,$$

where $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ satisfy the δ-no-signaling condition if the random variables $(\mathsf{q}_1, \ldots, \mathsf{q}_\ell)$ and $(\mathsf{a}_1, \ldots, \mathsf{a}_\ell)$ are δ-no-signaling.

We say that the argument system is (S, ϵ)-secure against δ-no-signaling strategies, for a function $S(k)$ and a negligible function $\epsilon(k)$, if for every constant $c > 0$, the soundness property holds with probability at most $\epsilon(k)^c$ for every pair of adversaries of size $S(k)^c$ satisfying the δ-no-signaling condition.

3 Local Satisfiability

In this section we introduce the notion of *local satisfiability* for formulas, and state a result of [KRR14] providing a no-signaling multi-prover argument for the *local* satisfiability of any *non-deterministic* Turing machine computation.

We start by describing, for every non-deterministic Turing machine M and input x, a formula $\varphi_{M,x}$ of a specific structure that is satisfiable if and only if M accepts x. Then we define the notion of local satisfiability for formulas. Finally we state a result of [KRR14] providing a no-signaling multi-prover argument for the local satisfiability of formulas of the form $\varphi_{M,x}$.

3.1 A Formula Describing Non-Deterministic Computations

The machine M. Let M be a T-time S-space non-deterministic Turing machine. We can think of M as a two-input machine, such that M accepts the input x if and only if there exists a witness w such that $M(x, w)$ accepts. In what follows, we consider a machine M and an input x such that $|x|$ is smaller than the machine's space S. Therefore, we can assume that M copies the entire input x to its work tape. However, the witness w we consider may be such that $|w|$ is much larger than S and therefore w must be given on a separate read-only read-once witness tape.

The machine's state. For $i \in [\mathsf{T}]$ let $\mathsf{st}_i \in \{0,1\}^{O(\mathsf{S})}$ denote the state of the computation $M(x,w)$ immediately before the i-th step. The state st_i includes:

- the machine's state.
- the entire content of the work tape, including the reading head's location.
- the reading head's location j on the witness tape, and the witness bit w_j.

Note that st_i does not include the entire content of the witness tape which may be much longer than S.

The following theorem states that the decision of whether a non-deterministic Turing machine M accepts an inputs x can be converted into a 3-CNF formula $\varphi_{M,x}$ of a specific structure. Loosely speaking, the variables of $\varphi_{M,x}$ correspond to the entire tableau of the computation of $M(x,w)$, and the formula verifies the consistency of all the states of this computation. Thus, $\varphi_{M,x}$ can be separated into sub-formulas, where each sub-formula verifies the consistency of two adjacent states of the computation. This intuition is formalized in the following theorem.

Theorem 3. *For any T-time S-space non-deterministic Turing machine M and any input x there exists a 3-CNF Boolean formula $\varphi_{M,x}$ of size $O(\mathsf{T} \cdot \mathsf{S})$ such that the following holds:*

1. *$\varphi_{M,x}$ is satisfiable if and only if M accepts x. Moreover, given a witness for the fact that M accepts x there is an efficient way to find a satisfying assignment to $\varphi_{M,x}$.*
2. *The formula $\varphi_{M,x}$ can be written as*

$$\varphi_{M,x} = \bigwedge_{i \in [\mathsf{T}-1]} \varphi^i_{M,x} \ ,$$

and the set of the input variables of $\varphi_{M,x}$, denoted by V, can be divided into subsets

$$V = \bigcup_{i \in [\mathsf{T}]} V_i,$$

such that each formula $\varphi^i_{M,x}$ is over the variables $V_i \cup V_{i+1}$, and each $V_i \subseteq V$ is of size $\mathsf{S}' = O(\mathsf{S})$.

3. *There exists an efficient algorithm State such that given an assignment to the variables V_i, outputs a state st_i of the computation of $M(x)$ immediately before the i-th step,*

$$\mathsf{st}_i = \mathsf{State}(\mathsf{a}[V_i]) \ .$$

The algorithm State satisfies the following properties:

- *For every $i \in [\mathsf{T}-1]$ and for every assignment $\mathsf{a} \in \{0,1\}^{V_i \cup V_{i+1}}$, if $\varphi^i_{M,x}(\mathsf{a}) = 1$ then the states*

$$\mathsf{st}_i = \mathsf{State}(\mathsf{a}[V_i]) \quad , \quad \mathsf{st}_{i+1} = \mathsf{State}(\mathsf{a}[V_{i+1}])$$

are consistent with the program M.

- *For every assignment $\mathsf{a} \in \{0,1\}^{V_1 \cup V_2}$, if $\varphi^1_{M,x}(\mathsf{a}) = 1$ then the state*

$$\mathsf{st}_1 = \mathsf{State}(\mathsf{a}[V_1])$$

is the initial state of the machine M with the input x.

- *For every assignment $\mathsf{a} \in \{0,1\}^{V_{\mathsf{T}-1} \cup V_{\mathsf{T}}}$, if $\varphi^{\mathsf{T}-1}_{M,x}(\mathsf{a}) = 1$ then the state*

$$\mathsf{st}_{\mathsf{T}} = \mathsf{State}(\mathsf{a}[V_{\mathsf{T}}])$$

is an accepting state.

Remark 3.1 (On the formula size)**.** *It is well known that there exists a formula of size only $\tilde{O}(\mathsf{T})$ (independent of S) that is satisfiable if and only if M accepts x. Such a formula can be obtained by first making the machine M oblivious [PF79]. However such a formula will not have the desired structure described in Theorem 3.*

3.2 Definition of Local Satisfiability

In this section we define the notion of local satisfiability for formulas.

Definition 3.2 (Local Assignment Generator)**.** *Let φ be a 3-CNF formula over a set of variables V. An (ℓ, ϵ, δ)-local assignment generator* Assign *for φ is a probabilistic algorithm running in time $\mathrm{poly}(|V|)$ that takes as input a set of at most ℓ queries $Q \subseteq V, |Q| \leq \ell$, and outputs an assignment $\mathbf{a} \in \{0,1\}^Q$, such that the following two properties hold.*

- **Everywhere Local Consistency.** *For every set $Q \subseteq V, |Q| \leq \ell$, with probability $1 - \epsilon$ over a draw*
$$\mathbf{a} \leftarrow \mathsf{Assign}(Q) \ ,$$
the assignment is locally consistent with the formula φ. That is, for every variables $q_1, q_2, q_3 \in Q$, every clause in φ over the variables q_1, q_2, q_3 is satisfied by the assignment $\mathbf{a}[\{q_1, q_2, q_3\}]$.
- **No-signaling.** *For every (all powerful) distinguisher D and every sets $Q' \subseteq Q \subseteq V, |Q| \leq \ell$:*
$$\left| \Pr_{\mathbf{a} \leftarrow \mathsf{Assign}(Q)} \left[D(\mathbf{a}[Q']) = 1 \right] - \Pr_{\mathbf{a}' \leftarrow \mathsf{Assign}(Q')} \left[D(\mathbf{a}') = 1 \right] \right| \leq \delta \ .$$

Remark 3.3 (On ordered queries)**.** *In [PR14], the notion of local satisfiability is formalized using an ordered vector of queries. In Definition 3.2 however, the queries are given as an unordered set. We note that these formulations are equivalent.*

3.3 No-Signaling Multi-Prover Arguments for Local Satisfiability

To obtain our results we use a multi-prover proof system satisfying a no-signaling local soundness property (see Theorem 4 below). Such a proof system was constructed in [KRR14].

Let k be the security parameter and let $\ell = \ell(k)$ be a polynomial. Let M be a non-deterministic Turing machine running in time T and space S, let $x \in \{0,1\}^m$ be an input to M such that $\mathsf{T}(m) < 2^k$ and let w be a witness. We consider an ℓ-prover proof system $(\mathsf{LS.QueryGen}, \mathsf{LS.Prover}, \mathsf{LS.Verifier})$ with the following syntax and efficiency:

- $\mathsf{LS.QueryGen}(1^k) \to ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st})$:
A randomized polynomial-time algorithm that outputs a set of $\ell = \ell(k)$ queries $(\mathsf{q}_1, \ldots, \mathsf{q}_\ell)$, and a secret state st.
- $\mathsf{LS.Prover}(1^{\mathsf{T}(m)}, M, x, w, \mathsf{q}) \to \mathsf{a}$:
A deterministic algorithm running in time $\mathsf{T}(m) \cdot \mathsf{S}(m) \cdot \mathrm{poly}(k)$ that outputs an answer a to a single query q where $|\mathsf{a}| = O(\log(k))$.
- $\mathsf{LS.Verifier}(M, x, \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \to b$:
A deterministic algorithm running in time $m \cdot \mathrm{poly}(k)$, that outputs an acceptance bit b.

The completeness and no-signaling local soundness properties of the above proof system are given by Theorem 4 proved in [KRR14].[6]

[6]The proof of Theorem 4 follows by combining Lemma 14.1, Lemma 6.1, and Lemma 7.29 in [KRR14] together with the fact that all the claims and lemmas in Sections 7.1-7.5 hold for arbitrary setting of parameters, and in particular for any ϵ and δ.

Theorem 4 ([KRR14]). *There exists a polynomial ℓ_0, such that for every polynomial ℓ' and for $\ell = \ell_0 \cdot \ell'$ there exists an ℓ-prover proof system* $(\mathsf{LS.QueryGen}, \mathsf{LS.Prover}, \mathsf{LS.Verifier})$ *that satisfies the following properties.*

- *Completeness. For every* T*-time (two input) Turing machine M, every input $x \in \{0,1\}^m$ and witness w such that $M(x,w) = 1$, every $k \in \mathbb{N}$ such that $\mathsf{T}(m) < 2^k$, and for $\ell = \ell(k)$,*

$$\Pr\left[\begin{array}{l} ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{LS.QueryGen}(1^k); \\ \forall i \in [\ell] : \mathsf{a}_i \leftarrow \mathsf{LS.Prover}(1^{\mathsf{T}(m)}, M, x, w, \mathsf{q}_i); \\ 1 \leftarrow \mathsf{LS.Verifier}(M, x, \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] = 1 \ .$$

- *No-Signaling Local Soundness. There exists a probabilistic polynomial-time oracle machine* Assign *such that the following holds. For every* T*-time (two input) Turing machine M, every input $x \in \{0,1\}^m$, every security parameter $k \in \mathbb{N}$ such that $\mathsf{T}(m) < 2^k$ and $\ell = \ell(k)$, every $\epsilon = \epsilon(k)$, every $\delta = \delta(k)$, and every δ-no-signaling cheating prover* Prover^* *such that*

$$\Pr\left[\begin{array}{l} ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{LS.QueryGen}(1^k); \\ (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \leftarrow \mathsf{Prover}^*(\mathsf{q}_1, \ldots, \mathsf{q}_\ell); \\ 1 \leftarrow \mathsf{LS.Verifier}(M, x, \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] \geq \epsilon,$$

  $\mathsf{Assign}^{\mathsf{Prover}^*}$ *is an $(\ell', \delta', \epsilon')$-local assignment generator for the 3-CNF formula $\varphi_{M,x}$ given by Theorem 3, with*

$$\delta' = \frac{\delta \cdot 2^{k \cdot \mathrm{polylog}(\mathsf{T}(m))}}{\epsilon} \quad , \quad \epsilon' = \frac{\delta \cdot \mathrm{polylog}(\mathsf{T}(m))}{\epsilon} \ .$$

  As before, we say that Prover^* *is δ-no-signaling if the random variables $(\mathsf{q}_1, \ldots, \mathsf{q}_\ell)$ and $(\mathsf{a}_1, \ldots, \mathsf{a}_\ell)$ are δ-no-signaling.*

4 No-Signaling Multi-Prover Arguments for RAM Computations

4.1 Verifying RAM computations via Local Satisfiability

In this section we translate any RAM computation into a non-deterministic Turing machine such that the RAM computation is correct if and only if the Turing machine's computation is locally satisfiable. Consider an execution of a RAM program M that on input x and initial memory string D outputs y and results in memory D_{new} within time T. Consider a hash-tree of the initial memory D rooted at rt and a hash-tree of the final memory D_{new} rooted at $\mathsf{rt}_{\mathsf{new}}$.

We describe a Turing machine TVer that takes as input tuples of the form $(M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}})$, together with a corresponding witness, which is a *trancript* of the RAM computation. We start by describing the algorithm TGen which generates the transcript. Roughly, the transcript contains a hash-tree proof of consistency for every memory access made by M (the precise structure of the transcript is desctribed below). We then describe the algorithm TVer. The running time of TVer and TGen is proportinal to the running time of the RAM computation (up to polynomial factors in the security parameter) and is independent of the size of the memory. In terms of soundness we argue that for any incorrect $(y', \mathsf{rt}'_{\mathsf{new}}) \neq (y, \mathsf{rt}_{\mathsf{new}})$, any cheating prover that passes the no-signaling local soundness criterion for the computation of TVer with the instance $(M, x, \mathsf{T}, \mathsf{rt}, y', \mathsf{rt}'_{\mathsf{new}})$ can be used to break the soundness of the hash tree.

Let M be a RAM program, $x \in \{0,1\}^m$ be an input, and $D \in \{0,1\}^n$ be an initial memory string. Let

$$(\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite})$$

be a hash-tree scheme and let

$$\mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k) \ ,$$
$$(\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D) \ .$$

The transcript generation program TGen**.** We start by describing a program TGen that creates the transcript of the computation $M^D(x)$. Let

$$\mathsf{TGen}^{(\mathsf{tree}\to\mathsf{tree}_{\mathsf{new}})}(1^k, 1^{\mathsf{T}}, n, M, x) \to (y, \mathsf{rt}_{\mathsf{new}}, \mathsf{Trans})$$

be the following RAM program. TGen emulates the execution of $M^D(x)$ step by step. The emulation begins with the initial memory containing the hash tree $\mathsf{tree}_1 = \mathsf{tree}$ with the initial root $\mathsf{rt}_1 = \mathsf{rt}$, the empty initial state state_1 and the read location $i^{\mathsf{r}}_1 = 1$. Starting from $j = 1$, the j-th emulation step proceeds as follows:

1. Read from the hash tree the bit:

$$(b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j) \leftarrow \mathsf{HT.Read}^{\mathsf{tree}_j}(i^{\mathsf{r}}_j) \ .$$

2. Compute $(\mathsf{state}_{j+1}, i^{\mathsf{r}}_{j+1}, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}) \leftarrow \mathsf{STEP}(M, x, \mathsf{state}_j, b^{\mathsf{r}}_j)$.

3. If $i^{\mathsf{w}}_{j+1} \neq \perp$, write a bit to the hash tree:

$$(\mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}) \leftarrow \mathsf{HT.Write}^{(\mathsf{tree}_j\to\mathsf{tree}_{j+1})}(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}) \ .$$

The program M terminates after T emulation steps were completed with the terminating state $\mathsf{state}_{\mathsf{T}+1}$, which contains the output bit y. TGen then outputs y, $\mathsf{rt}_{\mathsf{new}} = \mathsf{rt}_{\mathsf{T}+1}$ and the transcript:

$$\mathsf{Trans} = \left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j\right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}\right)\right)_{j\in[\mathsf{T}]} \ .$$

The running time of the program TGen is $\mathsf{T} \cdot \mathrm{poly}(k)$.

The transcript verification program TVer**.** Let

$$\mathsf{TVer}((M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}}), \mathsf{Trans}) \to b$$

be the following Turing machine. TVer verifies the emulation of $M^D(x)$ based on the transcript:

$$\mathsf{Trans} = \left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j\right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}\right)\right)_{j\in[\mathsf{T}']} \ ,$$

produced by TGen. The program first verifies that $\mathsf{T}' = \mathsf{T}$. Then, starting from the initial root $\mathsf{rt}_1 = \mathsf{rt}$, the empty initial state state_1, the read location $\widetilde{i^{\mathsf{r}}_1} = i^{\mathsf{r}}_1 = 1$, and from $j = 1$, the j-th verification step proceeds as follows:

1. Verify that $\widetilde{i^{\mathsf{r}}_j} = i^{\mathsf{r}}_j$ and that

$$1 \leftarrow \mathsf{HT.VerRead}(\mathsf{rt}_j, i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j) \ .$$

2. Compute $(\mathsf{state}_{j+1}, \widetilde{i^{\mathsf{r}}_{j+1}}, \widetilde{i^{\mathsf{w}}_{j+1}}, \widetilde{b^{\mathsf{w}}_{j+1}}) \leftarrow \mathsf{STEP}(M, x, \mathsf{state}_j, b^{\mathsf{r}}_j)$.

3. Verify that $(\widetilde{i^{\mathsf{w}}_{j+1}}, \widetilde{b^{\mathsf{w}}_{j+1}}) = (i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1})$.

4. If $i^{\mathsf{w}}_{j+1} = \perp$ verify that $\mathsf{rt}_j = \mathsf{rt}_{j+1}$. Else, verify that

$$1 \leftarrow \mathsf{HT.VerWrite}(\mathsf{rt}_j, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}) \ .$$

5. If $j = \mathsf{T}$ verify that $\mathsf{rt}_{\mathsf{T}+1} = \mathsf{rt}_{\mathsf{new}}$ and that $\mathsf{state}_{\mathsf{T}+1}$ is terminating and includes the output y.

The program outputs 1 if and only if all the verifications where successful. The running time of the program TVer is $\mathsf{T} \cdot \mathrm{poly}(k)$ and its space complexity is $\mathrm{poly}(k) \cdot \mathrm{polylog}(n) = \mathrm{poly}(k)$ (see Remark 2.1).

Additional structure of TVer. In order to prove Theorem 5 below, we make additional assumptions on the structure of the Turing machine TVer. Intuitively, we assume that the execution of the machine can be divided into *blocks* where the computation in the j-th block is executing the j-th verification step. This assumption is satisfied by some "natural" implementation of TVer.

Formally, let $b = b(k) \leq \mathrm{poly}(k)$ be the block size. For every input $\tilde{x} = (M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}})$ and for every transcript

$$\mathsf{Trans} = \left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j \right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1} \right) \right)_{j \in [\mathsf{T}]} \ ,$$

(not neccesarily such that $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$ accepts) let $\mathsf{T}' = \mathsf{T} \cdot b$ be the running time of $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$. For $i \in [\mathsf{T}']$ let st_i be the state of the computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$ immediately before the i-th step. For $j \in [\mathsf{T}]$ the j-th computation block contains the states

$$\{\mathsf{st}_i : (j-1) \cdot b < i \leq j \cdot b\} \ .$$

Rejecting states. When one of the tests performed by TVer fails, the machine transitions into a rejecting state. Once TVer is in a rejecting state, we require that all its future states are rejecting. (indeed, if one of the tests performed by TVer fails, the machine always rejects). Formally, we require that there exists an efficiently computable predicate Reject such that

1. For every $i \in [\mathsf{T}' - 1]$ if $\mathsf{Reject}(\mathsf{st}_i) = 1$ then $\mathsf{Reject}(\mathsf{st}_{i+1}) = 1$.
2. The computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$ rejects if and only if $\mathsf{Reject}(\mathsf{st}_{\mathsf{T}'}) = 1$.

Additional requirements on the structure of TVer. Using the notion of blocks and rejecting states we formally define additional requirements on the structure of TVer.

3. For every $j \in [\mathsf{T}]$, the j-th computation block reads only the j-th entry of the transcript

$$\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j \right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1} \right) \ .$$

4. If during the computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$ one of the tests in Steps 1,3,4,5 performed in the j-th verification step fails, then at least one of the states in the j-th computation block will be a rejecting. That is, $\mathsf{Reject}(\mathsf{st}_i) = 1$ for some $(j-1) \cdot b < i \leq j \cdot b$.

Theorem 5. *The machines TGen and TVer satisfy the following properties:*

- *Completeness. For every $k \in \mathbb{N}$, every T-time RAM program M, every input $x \in \{0,1\}^m$, every initial memory $D \in \{0,1\}^n$ and every (y, D_{new}) such that $\mathsf{T}(m), n \leq 2^k$ and $y \leftarrow M^{(D \to D_{\mathsf{new}})}(x)$*

$$\Pr \left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (\mathsf{tree}_{\mathsf{new}}, \mathsf{rt}_{\mathsf{new}}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D_{\mathsf{new}}); \\ (y', \mathsf{rt}'_{\mathsf{new}}, \mathsf{Trans}) \leftarrow \mathsf{TGen}^{(\mathsf{tree} \to \mathsf{tree}_{\mathsf{new}})}(1^k, 1^{\mathsf{T}(m)}, n, M, x); \\ 1 \leftarrow \mathsf{TVer}((M, x, \mathsf{T}(m), \mathsf{rt}, y', \mathsf{rt}'_{\mathsf{new}}), \mathsf{Trans}); \\ (y', \mathsf{rt}'_{\mathsf{new}}) = (y, \mathsf{rt}_{\mathsf{new}}) \end{array} \right] = 1 \ .$$

- *Soundness*

  Assume HT *is an* (S,ϵ)*-secure hash-tree scheme for a function* $S(k)$ *and a negligible function* $\epsilon(k)$*. There exists a polynomial* ℓ' *such that for every constant* $c>0$ *and every pair of adversaries* $(\mathsf{Adv}_1,\mathsf{Adv}_2)$ *of size* $S(k)^c$*, there exist constants* $c_1,c_2>0$ *such that for every large enough* $k\in\mathbb{N}$

$$\Pr\left[\begin{array}{l}\mathsf{key}\leftarrow\mathsf{HT.Gen}(1^k);\\ (M,x,1^{\mathsf{T}},D,y',\mathsf{rt}'_{\mathsf{new}})\leftarrow\mathsf{Adv}_1(1^k,\mathsf{key});\\ y\leftarrow M^{(D\rightarrow D_{\mathsf{new}})}(x);\\ (\mathsf{tree},\mathsf{rt})\leftarrow\mathsf{HT.Hash}(\mathsf{key},D);\\ (\mathsf{tree}_{\mathsf{new}},\mathsf{rt}_{\mathsf{new}})\leftarrow\mathsf{HT.Hash}(\mathsf{key},D_{\mathsf{new}});\\ (y,\mathsf{rt}_{\mathsf{new}})\neq(y',\mathsf{rt}'_{\mathsf{new}});\\ \mathsf{CHEAT}\end{array}\right]\leq\epsilon(k)^{c_2},$$

  where CHEAT *is the event that* $\mathsf{Adv}_2(\mathsf{key},\cdot)$ *is an* $(\ell'(k),S(k)^{-c_1},S(k)^{-c_1})$*-local assignment generator for the 3-CNF formula* $\varphi_{\mathsf{TVer},\tilde{x}}$ *where* $\tilde{x}=(M,x,\mathsf{T},\mathsf{rt},y',\mathsf{rt}'_{\mathsf{new}})$ *and* $\varphi_{\mathsf{TVer},\tilde{x}}$ *is as defined in Theorem 3.*

Proof. Completeness follows by construction. We focus on proving the soundness. Assume towards contradiction that for every ℓ' there exists a constant c and a pair of adversaries $(\mathsf{Adv}_1,\mathsf{Adv}_2)$ of size at most $S(k)^c$, such that for every constants $c_1,c_2>0$ and for infinitely many values of $k\in\mathbb{N}$:

$$\Pr\left[\begin{array}{l}\mathsf{key}\leftarrow\mathsf{HT.Gen}(1^k);\\ (M,x,1^{\mathsf{T}},D,y',\mathsf{rt}'_{\mathsf{new}})\leftarrow\mathsf{Adv}_1(1^k,\mathsf{key});\\ y\leftarrow M^{(D\rightarrow D_{\mathsf{new}})}(x);\\ (\mathsf{tree},\mathsf{rt})\leftarrow\mathsf{HT.Hash}(\mathsf{key},D);\\ (\mathsf{tree}_{\mathsf{new}},\mathsf{rt}_{\mathsf{new}})\leftarrow\mathsf{HT.Hash}(\mathsf{key},D_{\mathsf{new}});\\ (y,\mathsf{rt}_{\mathsf{new}})\neq(y',\mathsf{rt}'_{\mathsf{new}});\\ \mathsf{CHEAT}\end{array}\right]>\epsilon(k)^{c_2},\tag{1}$$

where CHEAT is the event that $\mathsf{Adv}_2(\mathsf{key},\cdot)$ is an $(\ell'(k),S(k)^{-c_1},S(k)^{-c_1})$-local assignment generator for the 3-CNF formula $\varphi_{\mathsf{TVer},\tilde{x}}$ where $\tilde{x}=(M,x,\mathsf{T},\mathsf{rt},y',\mathsf{rt}'_{\mathsf{new}})$. Fix $\ell'=2\cdot b\cdot\mathsf{S}'$ where $b=b(k)$ is the block size as defined by the structure of the machine TVer (see Section 4.1), where $\mathsf{S}'=O(\mathsf{S})$ is defined by the structure of the formula $\varphi_{\mathsf{TVer},\tilde{x}}$ (see Theorem 3), and where $\mathsf{S}=\mathrm{poly}(k)$ is the space complexity of the machine TVer (see Section 4.1). Indeed $\ell'=\mathrm{poly}(k)$. Fix the constant $c_1=3\cdot c$. (The choice of ℓ' and c_1 will become clear later in the proof.) Note that $\mathsf{T}\leq S(k)^c$. We use $(\mathsf{Adv}_1,\mathsf{Adv}_2)$ to construct an adversary Adv that breaks the (S,ϵ)-security of the hash-tree scheme (either the soundness of read or write).

Given key, Adv first emulates $\mathsf{Adv}_1(1^k,\mathsf{key})$ and obtains an input $\tilde{x}=(M,x,1^{\mathsf{T}},D,y',\mathsf{rt}'_{\mathsf{new}})$. W.l.o.g. we can assume that Adv_1 is deterministic. We say that $\mathsf{key}\leftarrow\mathsf{HT.Gen}(1^k)$ is *bad* if

$$\begin{aligned}y&=M^{(D\rightarrow D_{\mathsf{new}})}(x)\\ (\mathsf{tree},\mathsf{rt})&=\mathsf{HT.Hash}(\mathsf{key},D)\\ (\mathsf{tree}_{\mathsf{new}},\mathsf{rt}_{\mathsf{new}})&=\mathsf{HT.Hash}(\mathsf{key},D_{\mathsf{new}})\end{aligned}$$

,

are such that $(y,\mathsf{rt}_{\mathsf{new}})\neq(y',\mathsf{rt}'_{\mathsf{new}})$ and the event CHEAT holds. By (1), for infinitely many $k\in\mathbb{N}$,

$$\Pr_{\mathsf{key}\leftarrow\mathsf{HT.Gen}(1^k)}[\mathsf{key\ is\ bad}]>\epsilon(k)^{c_2}\ .\tag{2}$$

For the rest of the proof we fix a bad key and prove that $\mathsf{Adv}(\mathsf{key})$ succeeds in breaking the soundness of either read or write, with some constant probability. Adv computes:

$$(y,\mathsf{rt}_{\mathsf{new}},\mathsf{Trans})\leftarrow\mathsf{TGen}^{(\mathsf{tree}\rightarrow\mathsf{tree}_{\mathsf{new}})}(1^k,1^{\mathsf{T}},n,M,x)\ ,$$

where $n = |D|$ and where

$$\mathsf{Trans} = \left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j\right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}\right)\right)_{j\in[\mathsf{T}]} \ .$$

Let $\mathsf{T}' = \mathsf{T} \cdot b$ be the running time of $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$. By Theorem 3, the formula $\varphi_{\mathsf{TVer},\tilde{x}}$ can be divided into sub-formulas

$$\varphi_{\mathsf{TVer},\tilde{x}} = \bigwedge_{i\in[\mathsf{T}'-1]} \varphi^i_{\mathsf{TVer},\tilde{x}} \ .$$

Following the notation of Theorem 3, let V be the set of all input variables to $\varphi_{\mathsf{TVer},\tilde{x}}$ and let $\{V_i\}_{i\in[\mathsf{T}']}$ be sets such that the formula $\varphi^i_{\mathsf{TVer},\tilde{x}}$ is over the input variables $V_i \cup V_{i+1}$.

Recall that the execution of TVer can be divided into T blocks, where the the j-th block is executing the j-th verification step. For every block $j \in [\mathsf{T}]$, we denote by B_j the set of input variables describing the states of the j-th block in the computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$. Namely,

$$B_j = \bigcup_{i=(j-1)\cdot b+1}^{j\cdot b} V_i \ .$$

We also define the set $B_0 = V_1$.

Let $\mathbf{a}^* \in \{0,1\}^V$ be the *correct* assignment defined by the computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$. For a block $j \in \{0, 1, \ldots, \mathsf{T}\}$, we say that an assignment $\mathbf{a} \in \{0,1\}^{B_j}$ is correct if it describes the same states as the correct assignment. That is, for every i such that $V_i \subseteq B_j$,

$$\mathsf{State}(\mathbf{a}[V_i]) = \mathsf{State}(\mathbf{a}^*[V_i]) \ .$$

We say that the assignment $\mathbf{a} \in \{0,1\}^{B_j}$ is *rejecting* if one of the states it describes is rejecting (see Section 4.1). That is, there exists $V_i \subseteq B_j$ such that $\mathsf{Reject}(\mathsf{State}(\mathbf{a}[V_i])) = 1$. Recall that the assignment $\mathbf{a}$ is *locally consistent* if for every variables $q_1, q_2, q_3 \in B_j$, every clause in $\varphi_{\mathsf{TVer},\tilde{x}}$ over variables q_1, q_2, q_3 is satisfied by the assignment $\mathbf{a}[\{q_1, q_2, q_3\}]$.

For every $j \in [\mathsf{T}]$, Adv computes the assignment:

$$\mathbf{a}^j \leftarrow \mathsf{Adv}_2(\mathsf{key}, B_{j-1} \cup B_j) \ .$$

Note that the set $B_{j-1} \cup B_j$ is of size at most $\ell' = 2 \cdot b \cdot \mathsf{S}'$.

Next, Adv finds a *good* index $j \in [\mathsf{T}]$ such that:

- The assignment $\mathbf{a}^j$ is locally consistent.
- The assignment $\mathbf{a}^j[B_{j-1}]$ is correct.
- The assignment $\mathbf{a}^j[B_j]$ is not correct.
- The assignment $\mathbf{a}^j[B_j]$ is not rejecting.

If no good j exists, Adv samples new assignments:

$$\mathbf{a}^j \leftarrow \mathsf{Adv}_2(\mathsf{key}, B_{j-1} \cup B_j) \ ,$$

for every $j \in [\mathsf{T}]$ until a good index j is found. If no such index is found after $S(k)^c$ iterations, Adv aborts. The following claim bounds the probability that Adv aborts.

Claim 4.1. *Conditioned on the event* CHEAT*,* Adv *aborts with probability at most* 0.9*.*

Before proving the claim, we complete the description of the adversary Adv. We show that conditioned on finding a good j, Adv breaks the hash-tree soundness of either read or write with probability 1. By Claim 4.1 and by Equation (2), this happens with probability $> \epsilon(k)^{c_2}$ for every constant c_2. Since Adv is of size $\mathrm{poly}(S(k))$ we get a contradiction to the (S, ϵ)-security of the hash-tree scheme.

In the rest of the proof we describe how Adv breaks the hash-tree soundness using the assignment $\mathbf{a}^j$ where j is good. For a block $j \in [\mathsf{T}]$, and an assignment $\mathbf{a} \in \{0,1\}^{B_j}$ we denote by $\mathsf{Transcript}(\mathbf{a})$ the transcript bits read from the witness tape in the states

$$\{\mathsf{st}_i = \mathsf{State}(\mathbf{a}[V_j]) : V_i \subseteq B_j\} \ .$$

Recall that by Requirement 3 on the structure of TVer,

$$\left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j\right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}\right)\right) = \mathsf{Transcript}(\mathbf{a}^*[B_j]) \ .$$

(Recall that $\mathbf{a}^*$is the correct assignment defined by the computation $\mathsf{TVer}(\tilde{x}, \mathsf{Trans})$.) We also consider the transcript entry defined by the assignment $\mathbf{a}^j[B_j]$

$$\left(\left(i^{\mathsf{r}\prime}_j, b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j\right), \left(i^{\mathsf{w}}_{j+1}{}', b^{\mathsf{w}}_{j+1}{}', \mathsf{rt}'_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}{}'\right)\right) = \mathsf{Transcript}(\mathbf{a}^j[B_j]) \ .$$

The fact that j is good implies that $\mathbf{a}^j$ is locally consistent, $\mathbf{a}^j[B_{j-1}]$ is correct, but $\mathbf{a}^j[B_j]$ is incorrect. This, together with the fact that the variables B_{j-1} and B_j describe two consecutive blocks, implies that it must be the case that

$$\left(\left(i^{\mathsf{r}\prime}_j, b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j\right), \left(i^{\mathsf{w}}_{j+1}{}', b^{\mathsf{w}}_{j+1}{}', \mathsf{rt}'_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}{}'\right)\right) \neq \left(\left(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j\right), \left(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}\right)\right) \ . \quad (3)$$

Since $\mathbf{a}^j[B_{j-1}]$ is correct and since $\mathbf{a}^j[B_j]$ is not rejecting, by Requirement 4 on the structure of TVer, the test in Step 1 in the j-th verification step of TVer succeeds and therefore $i^{\mathsf{r}}_j = i^{\mathsf{r}\prime}_j$, and

$$1 \leftarrow \mathsf{HT.VerRead}(\mathsf{rt}_j, i^{\mathsf{r}}_j, b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j) \ .$$

By the definition of Trans

$$1 \leftarrow \mathsf{HT.VerRead}(\mathsf{rt}_j, i^{\mathsf{r}}_j, b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j) \ .$$

If $(b^{\mathsf{r}}_j, \mathsf{pf}^{\mathsf{r}}_j) \neq (b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j)$, Adv outputs $(D_j, i^{\mathsf{r}}_j, b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j)$ where D_j is the memory state after j steps of the RAM computation $M^D(x)$. In this case Adv breaks the hash-tree soundness of read.

If $(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j) = (b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j)$, since $\mathbf{a}^j[B_{j-1}]$ is correct and since $\mathbf{a}^j[B_j]$ is not rejecting, by Requirement 4 on the structure of TVer, the tests in Steps 3 and 4 in the j-th verification step of TVer succeed, and therefore $(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}) = (i^{\mathsf{w}}_{j+1}{}', b^{\mathsf{w}}_{j+1}{}')$, and

$$1 \leftarrow \mathsf{HT.VerWrite}(\mathsf{rt}_j, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}'_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}{}') \ .$$

By the definitions of Trans

$$1 \leftarrow \mathsf{HT.VerWrite}(\mathsf{rt}_j, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}) \ .$$

Equation (3), together with the fact that $i^{\mathsf{r}}_j = i^{\mathsf{r}\prime}_j$, and $(i^{\mathsf{r}}_j, b^{\mathsf{r}}_j) = (b^{\mathsf{r}\prime}_j, \mathsf{pf}^{\mathsf{r}\prime}_j)$, and $(i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}) = (i^{\mathsf{w}}_{j+1}{}', b^{\mathsf{w}}_{j+1}{}')$, implies that $(\mathsf{rt}_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}) \neq (\mathsf{rt}'_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}{}')$. In this case, Adv outputs

$$(D_j, i^{\mathsf{w}}_{j+1}, b^{\mathsf{w}}_{j+1}, \mathsf{rt}'_{j+1}, \mathsf{pf}^{\mathsf{w}}_{j+1}{}'),$$

and breaks the hash-tree soundness of write.

We conclude the proof of Theorem 5 with the proof of Claim 4.1.

Proof of Claim 4.1. Recall that Adv samples

$$\mathbf{a}^j \leftarrow \mathsf{Adv}_2(\mathsf{key}, B_{j-1} \cup B_j) \ ,$$

at most $S(k)^c$ times. In order to prove that Adv aborts with probability at most 0.9 it is sufficient to prove that in every one of the $S(k)^c$ iterations,

$$\Pr[\exists j \in [\mathsf{T}] : j \text{ is good } | \mathsf{CHEAT}] \geq 0.9 \cdot S(k)^{-c} \ ,$$

where the probability is over the sampled assignments $\left\{\mathbf{a}^j\right\}_{j\in[\mathsf{T}]}$.

Conditioned on CHEAT, $\mathsf{Adv}_2(\mathsf{key}, \cdot)$ is an $(\ell'(k), S(k)^{-c_1}, S(k)^{-c_1})$-local assignment generator for $\varphi_{\mathsf{TVer},\tilde{x}}$. For the rest of the proof we condition on CHEAT. The claim statement follows from the sequence of simple claims below.

Claim 4.2. *For every $j \in [\mathsf{T}]$*

$$\Pr[\mathbf{a}^j \textit{ is locally consistent}] \geq 1 - S(k)^{-c_1} \ .$$

Proof. The claim follows directly from the everywhere local consistency property of Adv_2. □

Claim 4.3. *For every $j \in [\mathsf{T}]$*

$$\Pr[\mathbf{a}^j[B_j] \textit{ is rejecting}] \leq (2(\mathsf{T} - j) + 1) \cdot S(k)^{-c_1} \ .$$

Proof. The proof is by induction, starting with $j = \mathsf{T}$.

By the structure of the formula $\varphi_{\mathsf{TVer},\tilde{x}}$ (Theorem 3), if $\mathbf{a}^{\mathsf{T}}$ is locally consistent then the state $\mathsf{State}(\mathbf{a}^{\mathsf{T}}[V_{\mathsf{T}'}])$ is accepting. Therefore, by Requirement 1 on the structure of TVer, for every i such that $V_i \subseteq B_{\mathsf{T}}$, the state $\mathsf{State}(\mathbf{a}^{\mathsf{T}}[V_i])$ is also accepting and the hence the assignment $\mathbf{a}^{\mathsf{T}}[B_{\mathsf{T}}]$ is not rejecting. This, together with Claim 4.2, implies that the claim holds for $j = \mathsf{T}$.

Next we argue that if the claim holds for $j > 2$ it also holds for $j - 1$. If $\mathbf{a}^j$ is locally consistent and $\mathbf{a}^j[B_j]$ is not rejecting then by Requirement 1 on the structure of TVer, for every i such that $V_i \subseteq B_{j-1}$, the state $\mathsf{State}(\mathbf{a}^{\mathsf{T}}[V_i])$ is accepting and the hence the assignment $\mathbf{a}^j[B_{j-1}]$ is also not rejecting. Therefore, by the induction hypothesis and Claim 4.2

$$\Pr[\mathbf{a}^j[B_{j-1}] \text{ is rejecting}] \leq (2(\mathsf{T} - j) + 2) \cdot S(k)^{-c_1} \ .$$

By the no-signaling property of Adv_2 it follows that:

$$\Pr[\mathbf{a}^{j-1}[B_{j-1}] \text{ is rejecting}] \leq (2(\mathsf{T} - j) + 3) \cdot S(k)^{-c_1} = (2(\mathsf{T} - (j-1)) + 1) \cdot S(k)^{-c_1} \ .$$

□

Claim 4.4. *If $\mathbf{a}^1$ is locally consistent then $\mathbf{a}^1[B_0]$ is correct.*

Proof. Recall that $B_0 = V_1$. if $\mathbf{a}^1$ is locally consistent then

$$1 = \varphi^1_{\mathsf{TVer},\tilde{x}}(\mathbf{a}^1[V_1 \cup V_2]) \ .$$

We also have that

$$1 = \varphi^1_{\mathsf{TVer},\tilde{x}}(\mathbf{a}^*[V_1 \cup V_2]) \ .$$

By the structure of the formula $\varphi_{\mathsf{TVer},\tilde{x}}$ (Theorem 3), $\mathsf{State}(\mathbf{a}^1[V_1])$ and $\mathsf{State}(\mathbf{a}^*[V_1])$ are both equal to the initial state of TVer with the input $\tilde{x}$, and therefore $\mathbf{a}^1[V_1]$ is correct. □

Claim 4.5. *If $\mathbf{a}^{\mathsf{T}}$ is locally consistent then $\mathbf{a}^{\mathsf{T}}[B_{\mathsf{T}}]$ is not correct.*

Proof. If $\mathbf{a}^{\mathsf{T}}$ is locally consistent, then

$$1 = \varphi^{\mathsf{T}'-1}_{\mathsf{TVer},\tilde{x}}(\mathbf{a}^{\mathsf{T}}[V_{\mathsf{T}'-1} \cup V_{\mathsf{T}'}]) \ ,$$

and by the structure of the formula $\varphi_{\mathsf{TVer},\tilde{x}}$ (Theorem 3) the state $\mathsf{State}(\mathbf{a}^{\mathsf{T}}[V_{\mathsf{T}'}])$ is accepting. However, since $(y, \mathsf{rt}_{\mathsf{new}}) \neq (y', \mathsf{rt}'_{\mathsf{new}})$, by Requirement 2 on the structure of TVer the test in Step 5 of the execution of $\mathsf{TVer}(\tilde{x}, \widetilde{\mathsf{Trans}})$ fails and the state $\mathsf{State}(\mathbf{a}^{\mathsf{T}}[V_{\mathsf{T}'}])$ is rejecting. It follows that $\mathbf{a}^{\mathsf{T}}[B_{\mathsf{T}}]$ is incorrect. □

Claim 4.6. *For every* $j \in [\mathsf{T}-1]$

$$\left|\Pr[\mathbf{a}^j[B_j] \textit{ is correct}] - \Pr[\mathbf{a}^{j+1}[B_j] \textit{ is correct}]\right| \leq S(k)^{-c_1} \ .$$

Proof. The claim follows directly from the no-signaling property of Adv_2. □

By Claims 4.2, 4.4, 4.5, 4.6, there exists $j \in [\mathsf{T}]$ such that

$$\left|\Pr[\mathbf{a}^j[B_j] \text{ is correct}] - \Pr[\mathbf{a}^j[B_{j-1}] \text{ is correct}]\right| \geq \frac{1}{\mathsf{T}} - 2 \cdot S(k)^{-c_1} \ .$$

Since $\mathsf{T} \leq S(k)^c$ and $c_1 = 3 \cdot c$

$$\left|\Pr[\mathbf{a}^j[B_j] \text{ is correct}] - \Pr[\mathbf{a}^j[B_{j-1}] \text{ is correct}]\right| \geq 0.9 \cdot S(k)^{-c} \ .$$

Following also Claims 4.2, and Claim 4.3

$$\Pr[\mathbf{a}^j \text{ is locally consistent } \wedge \mathbf{a}^j[B_j] \text{ is not rejecting}] \geq 1 - (2\mathsf{T}+2)S(k)^{-c_1}) \geq 1 - 0.1 \cdot S(k)^{-c} \ .$$

Overall we conclude that

$$\Pr[j \text{ is good}] \geq 0.5 \cdot S(k)^{-c} \ .$$

□

□

4.2 The Protocol

In this section we describe our no-signaling multi-prover argument for RAM computations. The construction uses the following components.

- A hash-tree scheme $(\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite})$, given by Theorem 2.
- The ℓ-prover proof system $(\mathsf{LS.QueryGen}, \mathsf{LS.Prover}, \mathsf{LS.Verifier})$ for local satisfiability given by Theorem 4 in Section 3.3, where $\ell = \ell' \cdot \ell_0$, and ℓ' is the polynomial given by Theorem 5 and ℓ_0 is the polynomial given by Theorem 4.
- The transcript generation and verification programs $\mathsf{TGen}, \mathsf{TVer}$ described in Section 4.1. We only rely on the following facts
  - The programs $\mathsf{TGen}, \mathsf{TVer}$ satisfy Theorem 5.
  - For security parameter k and for a T-time computation, the running time of the transcript generation program TGen is $\mathsf{T} \cdot \mathrm{poly}(k)$. The running time of the transcript verification program TVer (on the transcript generated by TGen) is $\mathsf{T} \cdot \mathrm{poly}(k)$ and its space complexity is $\mathrm{poly}(k)$.

The multi-prover argument is given by the following procedures:

- $\mathsf{ParamGen}(1^k)$ generates a key for the hash-tree:

$$\mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k) \ ,$$

  and outputs $\mathsf{pp} = \mathsf{key}$.

- $\mathsf{MemGen}(\mathsf{pp}, D)$, given $\mathsf{pp} = \mathsf{key}$, computes a hash-tree for the memory D:

$$(\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D) \ ,$$

  and outputs $(\mathsf{dt}, \mathsf{d}) = (\mathsf{tree}, \mathsf{rt})$.

- $\mathsf{QueryGen}(1^k)$ executes the query generation algorithm of the local-satisfiability proof system:

$$((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{LS.QueryGen}(1^k) \ ,$$

  and outputs $((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st})$.

- $\mathsf{Output}^{\mathsf{dt}}(1^{\mathsf{T}}, n, M, x)$, given access to the memory $\mathsf{dt} = \mathsf{tree}$, executes the transcript generation program:

$$(y, \mathsf{rt}_{\mathsf{new}}, \mathsf{Trans}) \leftarrow \mathsf{TGen}^{(\mathsf{tree} \to \mathsf{tree}_{\mathsf{new}})}(1^k, 1^{\mathsf{T}}, n, M, x) \ ,$$

  and outputs $(y, \mathsf{d}_{\mathsf{new}}, \mathsf{Trans}) = (y, \mathsf{rt}_{\mathsf{new}}, \mathsf{Trans})$.

- $\mathsf{Prover}((M, x, \mathsf{T}, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q})$, where $(\mathsf{d}, \mathsf{d}_{\mathsf{new}}) = (\mathsf{rt}, \mathsf{rt}_{\mathsf{new}})$, does the following:

  1. Let $\mathsf{T}' = \mathsf{T} \cdot \mathrm{poly}(k)$ and $\mathsf{S}' = \mathrm{poly}(k)$ be the time and space complexity of the computation

$$\mathsf{TVer}((M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}}), \mathsf{Trans}) \ .$$

  2. Execute the local-satisfiability prover for the above computation:

$$\mathsf{a} \leftarrow \mathsf{LS.Prover}(1^{\mathsf{T}'}, \mathsf{TVer}, (M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}) \ .$$

  3. Output a.

- $\mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell))$, where $(\mathsf{d}, \mathsf{d}_{\mathsf{new}}) = (\mathsf{rt}, \mathsf{rt}_{\mathsf{new}})$, executes the local-satisfiability verifier:

$$b \leftarrow \mathsf{LS.Verifier}(\mathsf{TVer}, (M, x, \mathsf{T}, \mathsf{rt}, y, \mathsf{rt}_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \ ,$$

  and outputs b.

Theorem 6. *Assume* HT *is an* (S, ϵ)*-secure hash-tree scheme for a function* $S(k)$ *and a negligible function* $\epsilon(k)$*. Then* $(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier})$ *is an* ℓ*-prover argument system for RAM computations that is* (S, ϵ)*-secure against* δ*-no-signaling provers for* $\delta(k) = 2^{-k \cdot \mathrm{polylog}(S(k))}$.

Proof. The syntax, efficiency and completeness properties of the protocol follow directly from those of the hash-tree scheme, the proof system for local satisfiability, and the transcript generation and verification programs, as well as from Theorem 5.

Soundness follows by combining Theorem 4 and Theorem 5, as follows. Assume towards contradiction that there exists a pair of adversaries $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ of size $\mathrm{poly}(S(k))$ satisfying the δ-no-signaling

condition for $\delta(k) = 2^{-k \cdot \mathrm{polylog}(S(k))}$, and there exists a constant $c > 0$, such that for infinitely many values of $k \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y', \mathsf{d}'_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y', \mathsf{d}'_{\mathsf{new}}); \\ ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, (\mathsf{q}_1, \ldots, \mathsf{q}_\ell)); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y', \mathsf{d}'_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)); \end{array}\right] \geq \epsilon(k)^c \ .$$

By construction we therefore have that for infinitely many values of $k \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y', \mathsf{rt}'_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{key}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (\mathsf{tree}_{\mathsf{new}}, \mathsf{rt}_{\mathsf{new}}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D_{\mathsf{new}}); \\ (y, \mathsf{rt}_{\mathsf{new}}) \neq (y', \mathsf{rt}'_{\mathsf{new}}); \\ ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{LS.QueryGen}(1^k); \\ (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{key}, (\mathsf{q}_1, \ldots, \mathsf{q}_\ell)); \\ 1 \leftarrow \mathsf{LS.Verifier}(\mathsf{TVer}, ((M, x, \mathsf{T}, \mathsf{rt}, y', \mathsf{rt}'_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] \geq \epsilon(k)^c \ .$$

Fix any such $k \in \mathbb{N}$. W.l.o.g. we can assume that Adv_1 is deterministic. It follows that with probability at least $\frac{1}{2} \cdot \epsilon(k)^c$ over $\mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k)$

$$\begin{aligned} (M, x, 1^{\mathsf{T}}, D, y', \mathsf{rt}'_{\mathsf{new}}) &\leftarrow \mathsf{Adv}_1(1^k, \mathsf{key}) \\ y &\leftarrow M^{(D \to D_{\mathsf{new}})}(x) \\ (\mathsf{tree}, \mathsf{rt}) &\leftarrow \mathsf{HT.Hash}(\mathsf{key}, D) \\ (\mathsf{tree}_{\mathsf{new}}, \mathsf{rt}_{\mathsf{new}}) &\leftarrow \mathsf{HT.Hash}(\mathsf{key}, D_{\mathsf{new}}) \\ (y, \mathsf{rt}_{\mathsf{new}}) &\neq (y', \mathsf{rt}'_{\mathsf{new}}) \ , \end{aligned}$$

as well as,

$$\Pr\left[\begin{array}{l} ((\mathsf{q}_1, \ldots, \mathsf{q}_\ell), \mathsf{st}) \leftarrow \mathsf{LS.QueryGen}(1^k); \\ (\mathsf{a}_1, \ldots, \mathsf{a}_\ell) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{key}, (\mathsf{q}_1, \ldots, \mathsf{q}_\ell)); \\ 1 \leftarrow \mathsf{LS.Verifier}(\mathsf{TVer}, (M, x, \mathsf{T}, \mathsf{rt}, y', \mathsf{rt}'_{\mathsf{new}}), \mathsf{st}, (\mathsf{a}_1, \ldots, \mathsf{a}_\ell)) \end{array}\right] \geq \frac{1}{2} \cdot \epsilon(k)^c \ . \tag{4}$$

Let Assign be the polynomial-time oracle machine given by Theorem 4. It follows that $\mathsf{Assign}^{\mathsf{Adv}_2(1^k, \mathsf{key}, \cdot)}$ is an $(\ell', \delta', \epsilon')$-local assignment generator for the 3-CNF formula $\varphi_{\mathsf{TVer}, (M, x, \mathsf{T}, \mathsf{rt}, y', \mathsf{rt}'_{\mathsf{new}})}$ where:

$$\delta'(k) = \epsilon(k)^{-c} \cdot \delta(k) \cdot 2^{k \cdot \log^{c_1}(\mathsf{T})} \quad , \quad \epsilon'(k) = \epsilon(k)^{-c} \cdot \delta(k) \cdot \log^{c_1}(\mathsf{T}) \ ,$$

for some constant $c_1 \in \mathbb{N}$. Recall that $\delta(k) = 2^{-k \cdot \mathrm{polylog}(S(k))}$. Let c_2 be a constant such that $\delta(k) \leq 2^{-k \cdot \log^{c_2}(S(k))}$ and $c_2 > (c_1 + 1)$, and let

$$\delta''(k) = \epsilon(k)^{-c} \cdot \delta(k) \cdot 2^{k \cdot \log^{c_2}(\mathsf{T})} \quad , \quad \epsilon''(k) = \epsilon(k)^{-c} \cdot \delta(k) \cdot \log^{c_2}(\mathsf{T}) \ .$$

We have that $\epsilon(k) \geq 2^{-k}$ and that $\mathsf{T} \leq |\mathsf{Adv}_1| = \mathrm{poly}(S(k))$, and therefore

$$\delta''(k), \epsilon''(k) \leq \mathrm{negl}(S(k)) \ .$$

Since (4) holds with probability at least $\frac{1}{2} \cdot \epsilon(k)^c$ over $\mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k)$

$$\Pr\left[\begin{array}{l} \mathsf{key} \leftarrow \mathsf{HT.Gen}(1^k); \\ (\mathsf{tree}, \mathsf{rt}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D); \\ (\mathsf{tree}_{\mathsf{new}}, \mathsf{rt}_{\mathsf{new}}) \leftarrow \mathsf{HT.Hash}(\mathsf{key}, D_{\mathsf{new}}); \\ \mathsf{CHEAT} \end{array}\right] \geq \frac{1}{2} \cdot \epsilon(k)^c,$$

where CHEAT is the event that $\mathsf{Assign}^{\mathsf{Adv}_2(1^k,\mathsf{pp},\cdot)}$ is an $(\ell'(k), \delta''(k), \epsilon''(k))$-local assignment generator for the 3-CNF formula $\varphi_{\mathsf{TVer},(M,x,\mathsf{T},\mathsf{rt},y',\mathsf{rt}'_{\mathsf{new}})}$ and $(y, \mathsf{rt}_{\mathsf{new}}) \neq (y', \mathsf{rt}'_{\mathsf{new}})$. Since $\mathsf{Assign}^{\mathsf{Adv}_2(1^k,\mathsf{pp},\cdot)}$ is of size $\mathrm{poly}(S(k))$, we get a contradiction to the statement of Theorem 5.

□

5 From No-Signaling Multi-Prover Arguments to Delegation

In this section we show how to transform any no-signaling multi-prover argument scheme for RAM computations into a two-message delegation scheme for RAM computations using a fully-homomorphic encryption scheme. The transformation follows the transformation of [KRR14] form no-signaling multi-prover interactive proofs to two-message delegation.

Theorem 7. *Assume there exists a fully homomorphic encryption scheme that is (S, ϵ)-secure. Assume there exists an ℓ-prover argument system for RAM computations that is (S', ϵ')-secure against δ-no-signaling strategies where the provers' answers are of size d (where d, just like ℓ and δ, is a function of the security parameter of the argument system). If there exists a polynomially bounded function $k' = k'(k)$ such that for every large enough k*

$$S'(k') > k \quad , \quad S(k) > \max(k, 2^{\ell(k') \cdot d(k')}) \quad , \quad \epsilon(k) \leq \frac{\delta(k')}{\ell(k')} \ ,$$

then there exists an (S'', ϵ'')-secure two-message delegation scheme for RAM computations where

$$S''(k) = S'(k'(k)) \quad , \quad \epsilon''(k) = \epsilon'(k'(k)) \ .$$

By combining Theorem 6, together with Theorem 7 with $k'(k) = \mathrm{polylog}(k)$ we obtain our main result, formally described in Theorem 8 below.

Theorem 8. *Assuming there exist a hash-tree scheme with sub-exponential security and an FHE scheme with quasi-polynomial security, then there exists a two-message delegation scheme for RAM computations.*

Remark 5.1. *We note that in the above theorem we could have replaced the use of an FHE scheme with any computational PIR scheme, at the price of increasing the runtime of the prover by $\mathrm{poly}(k') \cdot 2^d$. Since in the ℓ-prover argument given by Theorem 4, $2^d = \mathsf{T} \cdot \mathrm{poly}(k)$, this increase is harmless. In what follows we use an FHE scheme for the sake of notational convenience.*

Remark 5.2. *We can also combine Theorem 6 with Theorem 7 with $k'(k) = O(k)$ and conclude that there exists a two-message delegation scheme for RAM computations assuming the existence of (polynomially) secure hash-tree scheme and FHE scheme with sub-exponential security. However, Theorem 8 gives a stronger result since a hash-tree scheme can be constructed from an FHE scheme (or even a PIR scheme) with the same level of security [IKO05].*

The rest of this section is devoted to the proof of Theorem 7.

Proof of Theorem 7. Let

$$(\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval}) \ ,$$

be an (S, ϵ)-secure FHE scheme. Let

$$(\mathsf{ParamGen}', \mathsf{MemGen}', \mathsf{QueryGen}', \mathsf{Output}', \mathsf{Prover}', \mathsf{Verifier}') \ ,$$

be an ℓ-prover argument system for RAM computations that is (S', ϵ')-secure against δ-no-signaling strategies where the provers' answers are of size d. Let $k' = k'(k)$ be a polynomially bounded function such that

$$S'(k') > k \quad , \quad S(k) > \max(k, 2^{\ell(k') \cdot d(k')}) \quad , \quad \epsilon(k) \leq \frac{\delta(k')}{\ell(k')} \ .$$

Let S'', ϵ'' be the functions

$$S''(k) = S'(k'(k)) \quad , \quad \epsilon''(k) = \epsilon'(k'(k)) \ .$$

We construct an (S'', ϵ'')-secure two-message delegation scheme

$$(\mathsf{ParamGen}, \mathsf{MemGen}, \mathsf{QueryGen}, \mathsf{Output}, \mathsf{Prover}, \mathsf{Verifier}) \ ,$$

for RAM computations as follows. Let k be the security parameter of the two-message delegation scheme. The delegation scheme emulates an execution of the ℓ-prover argument system with security parameter $k' = k'(k)$. Let $\ell' = \ell(k')$ be the number of provers in this execution.

- $\mathsf{ParamGen}(1^k)$ executes the algorithm $\mathsf{ParamGen}'(1^{k'})$ of the ℓ-prover argument system with security parameter $k' = k'(k)$.
- $\mathsf{MemGen}(\mathsf{pp}, D)$ executes the algorithm $\mathsf{MemGen}'(\mathsf{pp}, D)$ of the ℓ-prover argument system.
- $\mathsf{QueryGen}(1^k)$ first executes the query generation algorithm of the ℓ-prover argument system with security parameter $k' = k'(k)$

  $$(\mathsf{q}'_1, \ldots, \mathsf{q}'_{\ell'}, \mathsf{st}') \leftarrow \mathsf{QueryGen}'(1^{k'}) \ .$$

  Then, for every $i \in [\ell']$, $\mathsf{QueryGen}$ computes:

  $$\begin{aligned} (\mathsf{pk}_i, \mathsf{sk}_i) &\leftarrow \mathsf{FHE.Gen}(1^k) \ , \\ \hat{\mathsf{q}}_i &\leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathsf{q}'_i) \ , \end{aligned}$$

  and outputs the query and state $(\mathsf{q}, \mathsf{st})$, where

  $$\begin{aligned} \mathsf{q} &= ((\mathsf{pk}_1, \hat{\mathsf{q}}_1), \ldots, (\mathsf{pk}_{\ell'}, \hat{\mathsf{q}}_{\ell'})) \ , \\ \mathsf{st} &= (\mathsf{st}', \mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell'}) \ . \end{aligned}$$

- $\mathsf{Output}^{\mathsf{dt}}(1^{\mathsf{T}}, n, M, x)$ executes the algorithm $\mathsf{Output}'(1^{\mathsf{T}}, n, M, x)$ of the ℓ-prover argument system, and answers all the oracle queries made by Output' using the oracle dt.
- $\mathsf{Prover}((M, x, \mathsf{T}, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q})$ is given the query

  $$\mathsf{q} = ((\mathsf{pk}_1, \hat{\mathsf{q}}_1), \ldots, (\mathsf{pk}_{\ell'}, \hat{\mathsf{q}}_{\ell'})) \ .$$

  For every $i \in [\ell']$, Prover computes

  $$\hat{\mathsf{a}}_i = \mathsf{FHE.Eval}(\mathsf{pk}_i, C, \hat{\mathsf{q}}_i) \ ,$$

where C is a circit that given a query q' to one of the provers in the ℓ-prover argument system, computes the prover's answer

$$C(\mathsf{q}') = \mathsf{Prover}'((M, x, \mathsf{T}, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{Trans}, \mathsf{q}')) \ .$$

Prover then outputs

$$\mathsf{pf} = (\hat{\mathsf{a}}_1, \ldots, \hat{\mathsf{a}}_{\ell'}) \ .$$

- $\mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}), \mathsf{st}, \mathsf{pf})$ is given the state and the proof

$$\mathsf{st} = (\mathsf{st}', \mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell'}) \ ,$$
$$\mathsf{pf} = (\hat{\mathsf{a}}_1, \ldots, \hat{\mathsf{a}}_{\ell'}) \ .$$

For every $i \in [\ell']$, Verifier computes

$$\mathsf{a}'_i = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\hat{\mathsf{a}}_i) \ .$$

Verifier then executes the verification algorithm of the ℓ-prover argument system

$$\mathsf{Verifier}'(M, x, \mathsf{T}(m), \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}, \mathsf{st}', (\mathsf{a}'_1, \ldots, \mathsf{a}'_{\ell'})) \ ,$$

and outputs the same as $\mathsf{Verifier}'$.

The completeness of the delegation scheme above follows from the perfect completeness of the underlying ℓ-prover argument system, and from the perfect completeness of the FHE scheme. Next we prove the soundness of the scheme.

Assume toward contradiction that there exists a constant $c > 0$ and pair of adversaries $(\mathsf{Adv}_1, \mathsf{Adv}_2)$ of size $S''(k)^c$ such that for infinitely many $k \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}(1^k); \\ (M, x, 1^{\mathsf{T}}, D, y^*, \mathsf{d}^*_{\mathsf{new}}) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y^*, \mathsf{d}^*_{\mathsf{new}}); \\ (\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{QueryGen}(1^k); \\ \mathsf{pf}^* \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, \mathsf{q}); \\ 1 \leftarrow \mathsf{Verifier}((M, x, \mathsf{T}, \mathsf{d}, y^*, \mathsf{d}^*_{\mathsf{new}}), \mathsf{st}, \mathsf{pf}^*) \end{array}\right] \geq \epsilon''(k)^c \ . \tag{5}$$

We reach a contradiction to the (S', ϵ')-security of the ℓ-prover argument system by showing that there exists a constant $c' > 0$ and a pair of adversaries $(\mathsf{Adv}^{\mathsf{NS}}_1, \mathsf{Adv}^{\mathsf{NS}}_2)$ of size $S'(k')^{c'}$ satisfying the δ-no-signaling condition, such that for infinitely many $k' \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ParamGen}'(1^{k'}); \\ (M, x, 1^{\mathsf{T}}, D, y^*, \mathsf{d}^*_{\mathsf{new}}) \leftarrow \mathsf{Adv}^{\mathsf{NS}}_1(1^{k'}, \mathsf{pp}); \\ y \leftarrow M^{(D \to D_{\mathsf{new}})}(x); \\ (\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{MemGen}'(\mathsf{pp}, D); \\ (\mathsf{dt}_{\mathsf{new}}, \mathsf{d}_{\mathsf{new}}) \leftarrow \mathsf{MemGen}'(\mathsf{pp}, D_{\mathsf{new}}); \\ (y, \mathsf{d}_{\mathsf{new}}) \neq (y^*, \mathsf{d}^*_{\mathsf{new}}); \\ ((\mathsf{q}'_1, \ldots, \mathsf{q}'_{\ell'}), \mathsf{st}') \leftarrow \mathsf{QueryGen}'(1^{k'}); \\ (\mathsf{a}'_1, \ldots, \mathsf{a}'_{\ell'}) \leftarrow \mathsf{Adv}^{\mathsf{NS}}_2(1^{k'}, \mathsf{pp}, (\mathsf{q}'_1, \ldots, \mathsf{q}'_{\ell'})); \\ 1 \leftarrow \mathsf{Verifier}'((M, x, \mathsf{T}, \mathsf{d}, y^*, \mathsf{d}^*_{\mathsf{new}}), \mathsf{st}', (\mathsf{a}'_1, \ldots, \mathsf{a}'_{\ell'})) \end{array}\right] \geq \epsilon'(k')^{c'} \ .$$

The adversary $\mathsf{Adv}_1^{\mathsf{NS}}$, given security parameter $1^{k'}$, executes $\mathsf{Adv}_1(1^k)$ where k is such that $k' = k'(k)$ and (5) holds. If there is more than one such k we arbitrarily choose one. If there is no such k, $\mathsf{Adv}_1^{\mathsf{NS}}$ aborts. We can assume w.l.o.g that the function k' is non-decreasing and therefore, since (5) holds for infinitely many $k \in \mathbb{N}$, there are infinitely many $k' \in \mathbb{N}$ for which $\mathsf{Adv}_1^{\mathsf{NS}}$ does not abort. The size of the adversary $\mathsf{Adv}_1^{\mathsf{NS}}$ is $\mathrm{poly}(S''(k)) = \mathrm{poly}(S'(k'))$.[7]

The adversary $\mathsf{Adv}_2^{\mathsf{NS}}$, given as input a tuple $(1^{k'}, \mathsf{pp}, (\mathsf{q}'_1, \ldots, \mathsf{q}'_{\ell'}))$, does as follows:

1. Abort if $\mathsf{Adv}_1^{\mathsf{NS}}(1^{k'})$ aborts. Otherwise, let $k \in \mathbb{N}$ be the security parameter chosen by $\mathsf{Adv}_1^{\mathsf{NS}}(1^{k'})$.
2. For every $i \in [\ell']$, let
$$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{FHE.Gen}(1^k) \ ,$$
$$\hat{\mathsf{q}}_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathsf{q}'_i) \ .$$
3. Emulate Adv_2 and obtain
$$(\hat{\mathsf{a}}_1, \ldots, \hat{\mathsf{a}}_\ell) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, ((\mathsf{pk}_1, \hat{\mathsf{q}}_1), \ldots, (\mathsf{pk}_{\ell'}, \hat{\mathsf{q}}_{\ell'}))) \ .$$
4. For every $i \in [\ell']$, let
$$\mathsf{a}'_i = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\hat{\mathsf{a}}_i) \ .$$
5. Output $(\mathsf{a}'_1, \ldots, \mathsf{a}'_{\ell'})$.

The size of the adversary $\mathsf{Adv}_2^{\mathsf{NS}}$ is $\mathrm{poly}(S''(k), k) = \mathrm{poly}(S'(k'))$. We first note that the view of the adversary Adv_2 emulated by $\mathsf{Adv}_2^{\mathsf{NS}}$ is distributed exactly like the view of Adv_2 in (5), and therefore, the proof generated by $\mathsf{Adv}_2^{\mathsf{NS}}$ is accepted with probability at least $\epsilon''(k)^c = \epsilon'(k')^c$.

We next argue that $(\mathsf{Adv}_1^{\mathsf{NS}}, \mathsf{Adv}_2^{\mathsf{NS}})$ satisfy the δ-no-signaling condition. Namely, we argue that the correlated distributions $(\mathsf{q}'_1, \ldots, \mathsf{q}'_{\ell'})$ and $(\mathsf{a}'_1, \ldots, \mathsf{a}'_{\ell'})$ are δ-no-signaling. To this end, assume towards contradiction that there exist parameters pp, two query vectors $\mathbf{q}$ and $\mathbf{q}'$, a set $S \subset [\ell']$ such that $\mathbf{q}_S = \mathbf{q}'_S$ and a distinguisher D such that

$$\left| \Pr_{\mathbf{a} \leftarrow \mathsf{Adv}_2^{\mathsf{NS}}(1^k, \mathsf{pp}, \mathbf{q})}[D(\mathbf{a}_S) = 1] - \Pr_{\mathbf{a}' \leftarrow \mathsf{Adv}_2^{\mathsf{NS}}(1^k, \mathsf{pp}, \mathbf{q}')}[D(\mathbf{a}'_S) = 1] \right| > \delta(k') \ . \tag{6}$$

Since D takes as input bit strings of length at most $\ell(k') \cdot d(k')$, it can be implemented by a circuit of size at most $2^{\ell(k') \cdot d(k')} \leq S(k)$.

Using D we break the semantic security of the FHE scheme. Specifically, we construct an adversary $\mathsf{Adv}^{\mathsf{FHE}}$ of size $\mathrm{poly}(S(k))$ that takes as input a set of public keys $\{\mathsf{pk}_i\}_{i \in [\ell'] \setminus S}$ and a set of ciphertexts $\{\mathsf{c}_i\}_{i \in [\ell'] \setminus S}$, and distinguishes between the case that each c_i was sampled from $\mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathbf{q}_i)$ and the case that each c_i was sampled by $\mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathbf{q}'_i)$ with probability at least $\delta(k')$. Following a standard hybrid argument, $\mathsf{Adv}^{\mathsf{FHE}}$ can be used to construct an adversary of size $\mathrm{poly}(S(k))$ that breaks the semantic security of the FHE scheme with probability at least $\frac{\delta(k')}{\ell(k')} \geq \epsilon(k)$.

The adversary $\mathsf{Adv}^{\mathsf{FHE}}$, given $\{(\mathsf{pk}_i, \mathsf{c}_i)\}_{i \in [\ell'] \setminus S}$, does as follows:

1. For every $i \in S$, let
$$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{FHE.Gen}(1^k) \ ,$$
$$\mathsf{c}_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathsf{q}'_i) \ .$$

[7] Note that $\mathsf{Adv}_1^{\mathsf{NS}}$ does not need to compute $k'(k)$. Instead, for every k, the value of k' as above is hard-coded in the (non-uniform) description of $\mathsf{Adv}_1^{\mathsf{NS}}$.

2. Emulate Adv_2 and obtain

$$(\hat{\mathsf{a}}_1, \ldots, \hat{\mathsf{a}}_{\ell'}) \leftarrow \mathsf{Adv}_2(1^k, \mathsf{pp}, ((\mathsf{pk}_1, \mathsf{c}_1), \ldots, (\mathsf{pk}_{\ell'}, \mathsf{c}_{\ell'}))) \ .$$

3. For every $i \in S$, let

$$\mathbf{a}_i = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\hat{\mathsf{a}}_i) \ .$$

4. Output $D(\mathbf{a}_S)$.

By (6) the adversary $\mathsf{Adv}^{\mathsf{FHE}}$ distinguishes between these two cases with probability at least $\delta(k')$ as required. □

References

[BCC+14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *IACR Cryptology ePrint Archive*, 2014:580, 2014.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BEG+91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 90–99, 1991.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448, 2015.

[CCC+15] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Computation-trace indistinguishability obfuscation and its applications. *IACR Cryptology ePrint Archive*, 2015:406, 2015.

[CH15] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. *IACR Cryptology ePrint Archive*, 2015:388, 2015.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 429–437, 2015.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 151–168, 2011.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 54–74, 2012.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GGR15] Oded Goldreich, Tom Gur, and Ron Rothblum. Proofs of proximity for context-free languages and read-once branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:24, 2015.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 404–413, 2014.

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 133–142, 2015.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 469–477, 2015.

[IKO05] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 445–456, 2005.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732, 1992.

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 422–442, 2015.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 169–189, 2012.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 369–378, 1987.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 436–453, 1994.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PR14] Omer Paneth and Guy N. Rothblum. Publicly verifiable non-interactive arguments for delegating computation. Cryptology ePrint Archive, Report 2014/981, 2014. `http://eprint.iacr.org/`.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802, 2013.