Secure Computation on Untrusted Platforms

by

Justin Lee Holmgren

S.B., Massachusetts Institute of Technology (2013)
M.Eng., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . Signature redacted .....
Department of Electrical Engineering and Computer Science
May 23, 2018

Certified by . . . Signature redacted May 23, 2018
Shafi Goldwasser
RSA Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . Signature redacted .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Secure Computation on Untrusted Platforms

by

Justin Lee Holmgren

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

In this thesis, we present two lines of research developing tools that, in addition to being of independent theoretical interest, yield improved protocols for secure outsourcing of computation:

Succinct Garbling Schemes.
A garbling scheme is a way to encode a program P and input x as $\tilde{P}$ and $\tilde{x}$ such that $\tilde{P}$ can be evaluated on $\tilde{x}$ to obtain $P(x)$, but $(\tilde{P}, \tilde{x})$ reveals nothing more than $P(x)$.

We devise an efficient garbling scheme, based on the recent notion of indistinguishability obfuscation, in which the RAM running time and space usage of P on x are each the same as for $\tilde{P}$ on $\tilde{x}$.

No-Signaling Multi-Prover Interactive Proofs.
A multi-prover interactive proof (MIP) is a protocol by a which a "verifier" can ascertain the truth of a mathematical statement by interacting with two or more "provers" that cannot communicate with each other.

We devise an MIP that achieves better efficiency and stronger soundness guarantees than previous constructions. In terms of efficiency, our MIP allows proving many statements with roughly the same (small) communication complexity as is required to prove a single statement. The soundness guarantee is that the verifier cannot be fooled even by malicious provers that can, in a very limited sense, collude in their messages to the verifier.

The latter guarantee crucially enables an application to *delegation of computation.* Specifically, we obtain a protocol by which a weak device can outsource expensive computations to a powerful but untrusted server, while being assured that the computation is performed correctly.

Thesis Supervisor: Shafi Goldwasser
Title: RSA Professor of Electrical Engineering and Computer Science

Acknowledgments

I am fortunate to have been surrounded by such intellectually and interpersonally fantastic people during my graduate studies. As a newcomer both to cryptography and to the broader theory community, I could not have hoped for friendlier or more welcoming colleagues.

I particularly thank Shafi Goldwasser, Ran Canetti, and Vinod Vaikuntanathan.

Shafi: There's so much that I often take for granted about your advising, and now is the time to thank you for all of it. No matter how busy your schedule and no matter which country you're in, you've always found time to listen to my problems and more often than not make these problems disappear. When our research interests diverged, you gave me the freedom – and funding – to work on the problems that interested me most. You make it all look easy, but I would be proud to be an advisor half as effective as you are.

Ran: Collaborating with you is something special. Beyond our clearly productive professional relationship, you taught me that colleagues are humans, and can be friends. When I was visiting Israel, you and Ronitt made me feel welcome as if I were part of your own family, for which I am extremely grateful. I'm remembering especially when you invited me to join your trip to Eilat – it was a lot of fun. Thank you also for your fine taste in food and the many delicious meals to which you've treated me. It's difficult for me to imagine what grad school would have been without you.

Vinod: It's remarkable how consistently you have a fascinating new perspective to offer on whatever topic I pose to you. It makes me regret that I haven't spent more time working with you. I want to thank you now because every time I *have* had a question for you (often with no more advance notice than a knock on your door), you've always taken the time and energy to give me an insightful and well thought out answer. Beyond technical matters, I'm especially grateful for the help and advice you've given me recently regarding academic job applications.

This thesis is based on joint works with Zvika Brakerski, Ran Canetti, Yilei Chen, Yael Kalai, and Mariana Raykova, and I thank them along with my other coauthors Aloni Cohen, Alex Lombardi, Ryo Nishimaki, Silas Richelson, Vinod Vaikuntanathan, Daniel Wichs, and Lisa Yang. I specifically want to thank several coauthors with whom I've had the opportunity to collaborate particularly closely: Aloni Cohen, Yael Kalai, Alex Lombardi, Ron Rothblum, and Lisa Yang. Your drive, curiosity, and creativity are an inspiration.

Finally, thanks to my family, to whom I ultimately owe everything.

Contents

Chapter 1

Introduction

An old adage states, "If you need something done right, do it yourself." In other words, there is a choice to be made. One can either live an easy life, delegating tasks to others, or one can have confidence that the tasks are done correctly. One surprising lesson of modern cryptography is that it is often possible to achieve the best of both worlds.

This is exemplified in delegation of computation, where the goal is to solve the following problem. One party (a "client") wishes to perform a computation (evaluating a function f on an input x) but is highly resource constrained. Imagine that this computation is expensive and is critically important to get right and/or involves private data. Specific examples of such computations might include "*Does radar data indicate an incoming attack?*" or "*Does a patient's genetic data indicate increased risk of disease?*".

There are two trivial yet unsatisfactory approaches by which the client may obtain $f(x)$.

- (Direct Computation) The first approach asks the client to compute $f(x)$ on its own. Depending on the client's resource constraints, this may be prohibitively expensive or impossible.

- (Outsourcing) In the second approach, the client sends f and x to a server and asks for the result $f(x)$. This approach corresponds to using a cloud "computation as a service" offering, or even simply to using less trusted (e.g., foreign manufactured) computing chips. The downside of this approach is that a bad server may return some value not equal to $f(x)$, for example out of laziness, malice, or incompetence. Just as dangerously, the server might learn private information about the provided computation.

In this thesis, we construct improved protocols that achieve the efficiency benefits of outsourcing, while simultaneously achieving the integrity and privacy guarantees of direct computation. To do this, we first construct two main technical tools: succinct garbling schemes and no-signaling multi-prover interactive proofs (MIPs).

1.1 Succinct Garbling

A garbling scheme $\mathcal{G}$ converts programs and input values into "opaque" constructs that reveal nothing but the corresponding output values. That is, $\mathcal{G}$ turns a program P into a garbled program $\widetilde{\mathbf{P}}$ and, separately, turns a value x into a garbled input $\tilde{\mathbf{x}}$, with the guarantees that $\widetilde{\mathbf{P}}(\tilde{\mathbf{x}}) = P(x)$ but the pair $(\widetilde{\mathbf{P}}, \tilde{\mathbf{x}})$ reveals *only* $P(x)$. Originally conceived by Yao [Yao86], garbling schemes are a pillar of cryptographic protocol design, with numerous applications such as secure two-party and multiparty computation protocols, verifiable delegation schemes, randomized encoding schemes, one time programs, and functional encryption.

A drawback of Yao's construction is that the size and runtime of $\widetilde{\mathbf{P}}$ are proportional to the size of P *when represented as a circuit.* This drawback is especially pronounced in situations where the input x is much larger than the program's size or RAM runtime — as in, say, keyword search in a large-but-sorted database — or when the runtime of the plaintext program varies from input to input. In these situations, the program P being garbled has a succinct representation, e.g. as a Turing or RAM machine, and converting P into a circuit incurs an exponential increase in its size.

Because of this drawback, there has been a sequence of work seeking to directly garble succinctly represented programs with better efficiency. In particular, two goals are the following:

- (Succinctness) A garbling scheme is succinct if whenever P has description length ℓ, then $\widetilde{\mathbf{P}}$ has description length $\text{poly}(\ell)$, and moreover is computable from P in time $\text{poly}(\ell)$.

- (Per-Instance Complexity Preservation) A garbling scheme preserves per-instance complexity if whenever P runs in time T (resp., space S) on an input x, then $\widetilde{\mathbf{P}}$ runs in time $\tilde{O}(T)$ (resp., space $\tilde{O}(S)$) on input $\tilde{\mathbf{x}}$

Goldwasser et al. [GKP+13] construct a succinct garbling scheme for *Turing machines,* namely a scheme where the size, runtime and space requirements of the garbled program are proportional to those of the Turing machine representation of the plaintext program. The security of their scheme relies on strong *extractability* assumptions that, loosely speaking, postulate the existence of an efficient algorithm for extracting secrets from a certain class of adversaries. Such assumptions are often viewed as problematic – technically speaking they are not complexity assumptions, as formalized by Goldwasser and Kalai [GK16], nor are they even "falsifiable" assumptions, as defined by Naor [Nao03] and Gentry and Wichs [GW11].

Another line of work initiated by Lu and Ostrovsky [LO13, GHL+14, GLOS15] constructs, from standard cryptographic assumptions (by now one-way functions), garbling schemes for *RAM programs* that preserve per-instance complexity. On the one hand, this is a great improvement, as in many cases RAM machines run exponentially faster than any equivalent Turing machine. In particular, RAM machines can for some tasks run in sublinear time. On the other hand, these schemes all fail to achieve succinctness. In particular, the size of a garbled RAM program is as large as its worst-case running time.

Garbling schemes for Turing machines that are both succinct and complexity-preserving were more recently developed based on the assumptions of indistinguishability obfuscation (IO) for circuits and injective one-way functions [BGL+15, CHJV15, KLW15]. The following questions remained open:

> *Does there exist a fully succinct garbling scheme for RAM programs? If so, under what assumptions?*

Any advancement on this question directly applies to the many applications of succinct garbling.

In this thesis, we obtain the following affirmative answer, based on the ITCS 2016 paper "Fully Succinct Garbled RAM" by Canetti and Holmgren.

Informal Theorem 1.1 ([CH16]). *If injective one-way functions and indistinguishability obfuscation for circuits exist, then there is a fully succinct garbling scheme for RAM programs.*

Applications of Succinct Garbling. Succinct garbling schemes have many usages in secure computation. The basic paradigm is to reduce the amount of computation that must be performed in a trusted manner. That is, if one needs to securely compute $P(x)$, one can instead securely compute $(\tilde{\mathbf{P}}, \tilde{\mathbf{x}})$ and from there evaluate $\tilde{\mathbf{P}}(\tilde{\mathbf{x}})$ in an untrusted environment. This paradigm is applicable in a variety of contexts: multi-party computation [IK00], delegation of computation [AIK10], obfuscation [App14, BGL+15, CHJV15], and more.

1.1.1 Memory Delegation

One extension of delegation involves computations that depend on a large amount of auxiliary information. Consider, for instance, the following scenario.

Alice is a researcher who is embarking on a groundbreaking experiment that involves collecting huge amounts of data over several months and then querying and running analytics on the data in ways to be determined as the data accumulates. Alas she does not have sufficient storage and processing power. Eve, who runs a large competing lab, offers servers for rent, but charges proportionally to storage and computing time. Can Alice make use of Eve's offer while being guaranteed that Eve does not learn or modify Alice's data and algorithms? Can she do it at a cost that's reasonably proportional to the size of the actual data and resource requirements? Repeatedly applying a protocol designed for one-off computations (e.g., [GKR08a, KRR13, KRR14, RRR16]) would result in excessive work for Alice. Specifically, for each computation she outsources she must run in time (and communication) proportional to her database size.

These considerations motivate memory delegation, in which after an initial setup phase each outsourced computation requires a nearly constant amount of work on Alice's part.[1] Fortunately, most garbling schemes, including ours, can be extended to support memory delegation.

[1] More precisely, the amount of work is a polynomial in the program size and a security parameter.

Adaptive Security

The increased functionality of memory delegation brings with it the possibility of a new, subtle vulnerability. Most garbling schemes are shown to have "static" security: for any *fixed* input x and program P, the distribution of $(\tilde{\mathbf{P}}, \tilde{\mathbf{x}})$ reveals only $P(x)$. However, in the scenario described above, P may be chosen much later than x, based on information that has been published in the interim by external parties. In particular, this admits the possibility that an adversary seeking to learn x may influence Alice into choosing P in a way that depends on $\tilde{\mathbf{x}}$, thus bypassing static security guarantees. Although this may seem far-fetched, it is analogous to fruitful efforts by Allied code-breakers during World War II to manipulate Germans into encrypting specific messages (e.g., the locations of planted British naval mines).

Adaptive security is considered in [GKR08b, BHR12, HJO+16] in the context of (non-succinct) garbled circuits. An adaptive garbling scheme for Turing machines is constructed in [AS16]. Still, adaptive security had not been achieved in the pertinent setting of succinct and persistent RAM garbling.

We would like a scheme to allow a user to garble an initial memory, and then garble RAM programs that arrive one by one in sequence. The machines should be able to read from and write to the memory, and also produce output. It should be guaranteed that:

1. **(Correctness)** Running the garbled programs one after another in sequence on the garbled memory results in the same sequence of outputs as running the plaintext machines one by one in sequence on the plaintext memory.

2. **(Security)** The view of any adversary that generates a database and programs and obtains their garbled versions is simulatable by a machine that sees only the initial database size and sequence of outputs of the plaintext programs when run in sequence on the plaintext database. This should hold even when the adversary chooses new plaintext programs adaptively, based on the garbled memory and garbled programs seen so far.

3. **(Efficiency)** The costs of all operations are comparable to what would anyways be required in an insecure solution. Specifically, up to polynomial factors in a security parameter:

    - Memory garbling takes linear time.
    - Program garbling takes polynomial time (in the *size* of the program, not its running time).
    - A garbled program's running time and space usage are the same as those of the plaintext program.

In this thesis, we show (based on the TCC 2016-B paper "Adaptive Succinct Garbled RAM or: How To Delegate Your Database" by Canetti, Chen, Holmgren, and Raykova) that one can construct an *adaptively secure* fully succinct garbling scheme for RAM programs, based on nearly the same computational assumptions as in the non-adaptive case.

Informal Theorem 1.2 ([CCHR16]). *If indistinguishability obfuscation for circuits exists, and if either factoring or the discrete log problem is hard, then there is an adaptively secure memory delegation scheme.*

1.2 No-Signaling PCPs and MIPs

A different approach to delegation relies on the powerful machinery of probabilistically checkable proofs (PCPs), or closely related multi-prover interactive proofs (MIPs). These protocols primarily achieve verifiability: on input a function f and input x, the verifier's output is either $f(x)$ or $\perp$. It is possible using fully homomorphic encryption to generically add secrecy to a delegation scheme, although this may incur a serious loss in prover efficiency. The main benefit of these schemes is that they can be based on milder assumptions and have the potential for greater generality and concrete efficiency.

Probabilistically checkable proofs (PCPs) [BFLS91, ALM+98] allow verification with very little communication. Specifically, they enable a prover to write a proof string π, but the verifier only needs to read very few locations in that proof (chosen according to a specific distribution) in order to be convinced of the correctness of the statement. Kilian [Kil92] translated this into an argument system using *collision resistant hash families* (CRHFs). CRHFs allow a prover to produce a short commitment to a long string (the PCP proof string π), and subsequently to credibly reveal specific locations of the committed string with a short proof. Kilian's protocol involves the exchange of 4 messages between the prover and the verifier.

A central question is to what degree a verifiable computation protocol can be made *non-interactive*, and under what assumptions. Micali [Mic94] observed that, following ideas of Fiat and Shamir [FS86], Kilian's protocol can be made entirely non-interactive in the random oracle model. However, random oracles do not exist in the real world. Indeed it was shown by Canetti, Goldreich, and Halevi that there are protocols that are secure in the random oracle model, but for which replacing the random oracle by any concrete hash family results in an insecure protocol [CGH98]. Moreover, even the specific round reduction heuristic of [FS86] is highly suspect. In particular, Goldwasser and Kalai demonstrated a protocol for which the heuristic fails for every hash family [GK03].

Biehl, Meyer, and Wetzel [BMW98] proposed an alternative approach for reducing the message complexity to 2 using any (computational) private information retrieval scheme. For simplicity, in what follows, we explain their idea in terms of a fully homomorphic encryption scheme (FHE). This is an encryption scheme that allows publicly computing $\mathsf{Enc}(f(x))$ from $\mathsf{Enc}(x)$ for any function f, in time comparable to that required just to compute f.

In their proposed delegation scheme, the verifier samples PCP queries $q_1, \ldots, q_k$ and sends $\mathsf{Enc}(q_1), \ldots, \mathsf{Enc}(q_k)$ to the prover, with each query encrypted under a different key. The prover generates the PCP string π, and (using the FHE functionality) sends back $\mathsf{Enc}(\pi_{q_1}), \ldots, \mathsf{Enc}(\pi_{q_k})$. The verifier decrypts each answer, and checks that the returned answers are consistent with a valid PCP string. The intuition behind

this construction is that because each query is encrypted under an independently generated key, a cheating prover should not be able to correlate the answers and will therefore respond as if according to a predetermined string π. However, this intuition is not correct, as was shown by Dwork et al. [DLN+01].

The crux of the problem is that the standard notion of "semantic security" for an encryption scheme does not actually force an adversary to answer each query locally; rather, it guarantees that the *distribution* of answers returned under a subset of keys does not depend on the queries encrypted under other keys. The latter property of an answer-generating process is called "no-signaling", and it is not the same as locality (e.g., [CHSH69]). It was later shown by Dodis et al. [DHRW16] that there are pathological encryption schemes for which this difference actually can be exploited, leading to an attack on the scheme of [BMW98]. The approach of [BMW98] *is* sound, however, if the underlying PCP is sound against *no-signaling* provers.

No-Signaling Sound PCPs Kalai, Raz, and Rothblum [KRR14] constructed a PCP for any polynomial-time computable language $\mathcal{L}$ with soundness against no-signaling strategies and the following efficiency:

- The verifier runs in nearly linear time, and makes a polylogarithmic number of queries to the proof string.
- The (honest) proof is a string of polynomial length over an alphabet of polylogarithmic size, and is computable in polynomial time.

Combining this with the transformation of [BMW98] yields a delegation protocol for P.

One extension of delegation concerns computations in which a prover has auxiliary information, also called a witness, with which the truth of its claim is more easily verified (say in polynomial time). Protocols in this setting are called NP delegation protocols.

A trivial solution to NP delegation is for the prover to first send its witness to the verifier. The problem is then reduced to P delegation, with the following caveats:

- The communication from the prover to verifier is at least the length of the witness, which may be very large.
- The reduction adds an additional round of interaction to the protocol.

In this thesis, we describe (based on the STOC 2017 paper "Non-Interactive RAM and Batch NP Delegation from any PIR" by Brakerski, Holmgren, and Kalai) a no-signaling PCP for "batch NP".

Informal Theorem 1.3 ([BHK17]). *For any* NP *language L with witnesses of length m, and for any k, there is a* PCP *for L^k with soundness against no-signaling strategies, in which the number of verifier queries is $m \cdot \mathrm{polylog}(k)$.*

This in particular yields a 2-message protocol for delegation in which, if a prover wants to prove many statements, the total communication is roughly the length of a *single* witness rather than the *sum* of the witness lengths. This protocol also has additional desirable properties, such as RAM efficiency for the prover and adaptive soundness in the case of P delegation.

Chapter 2

Preliminaries

We begin with some basic notations.

- We write $\mathbb{N}$ to denote the set of positive integers. For any non-negative integer n, we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$.
- When A and B are sets, we write $A \sqcup B$ to denote the disjoint union of A and B.
- For a set X and a set Y, $\mathrm{Func}(X, Y)$ or X^Y denotes the set of all functions from X to Y.
- For $n \in \mathbb{N}$, X^n denotes the set of n-tuples of elements of X.
- X^* denotes $\cup_{i \in \mathbb{N}} X^i$.
- For a set $S \subseteq [n]$, $S = \{i_1, \ldots, i_\ell\}$ with $i_1 < \cdots < i_\ell$, and a sequence $\vec{a} = (a_1, \ldots, a_n) \in X^n$, we write $\vec{a}_S$ to denote the tuple $(a_{i_1}, \ldots, a_{i_\ell})$. We use analogous notation for subsequences of infinite sequences ($X^{\mathbb{N}}$). More generally, if f is a function from X to Y, and if S is a subset of X, we write $f(S)$ to denote $\{f(x) : x \in S\}$. If S is an ordered set, $f(S)$ inherits the same ordering.
- For a finite set S, we write ℓ_S to denote the worst-case length of binary strings encoding elements of S (typically this will be $\lceil \log(|S|) \rceil$). We identify S with a subset of $\{0,1\}^{\ell_S}$.
- For a randomized algorithm $\mathcal{A}$, we write $\mathcal{A}(x; r)$ to denote the result of running $\mathcal{A}$ on input x with randomness r.

Definition 2.1 (Negligible Functions). *A function $\nu : \mathbb{N} \to [0,1]$ is* negligible *if $\nu(n)$ is $O(n^{-c})$ for every $c > 0$.*

2.1 Ensembles of Circuits and Distributions

Definition 2.2 (Circuit Ensembles). *A circuit ensemble is a collection of boolean circuits $\{C_\lambda\}_{\lambda \in I}$ indexed by some "index set" I.*

Most commonly I is the set of natural numbers $\mathbb{N}$, and we sometimes just write $\{C_\lambda\}$ or $\{C_\lambda\}_\lambda$ (possibly with a different variable instead of λ) when this is the case. We will frequently be concerned with the size of C_λ as a function of λ.

Definition 2.3. *For any function $s : \mathbb{N} \to \mathbb{N}$, we say that $\{C_\lambda\}$ is a* size-s circuit ensemble *if $|C_\lambda| = O(s(\lambda))$. The ensemble is said to be* polynomial size *if it is size-λ^c for some c.*

Definition 2.4 (Concrete Computational Indistinguishability). *For any $s \in \mathbb{N}$ and any $\delta \in (0,1)$, a random variable X is said to be computationally (s,δ)-*indistinguishable *from a random variable Y if for every size-s circuit C,*

$$\left|\Pr[C(X) = 1] - \Pr[C(Y) = 1]\right| \leq \delta.$$

For the following definitions, let $s : \mathbb{N} \to \mathbb{N}$ and $\delta : \mathbb{N} \to (0,1)$ be arbitary functions.

Definition 2.5 (Asymptotic Computational Indistinguishability). *Two ensembles of random variables $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are* computationally δ-indistinguishable *(denoted $X \approx_\delta Y$) if for every polynomial size circuit ensemble $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, it holds that*

$$\left|\Pr\left[C_\lambda(X_\lambda) = 1\right] - \Pr\left[C_\lambda(Y_\lambda) = 1\right]\right| \leq O(\delta(\lambda)).$$

We say simply that X is computationally indistinguishable *from Y (denoted $X \approx Y$) if X is λ^{-c}-computationally indistinguishable from Y for every $c > 0$.*

Definition 2.6 (Statistical Distance). *The* statistical distance *between two random variables X and Y, each with countable support, is*

$$\sum_{z\in \mathrm{Supp}(X)\cup\mathrm{Supp}(Y)} \frac{1}{2} \cdot \left|\Pr[X = z] - \Pr[Y = z]\right|.$$

2.2 Basic Cryptographic Primitives

Definition 2.7 (Oblivious RAM [GO96]). *An oblivious RAM (ORAM) is a pair of algorithms* (OMem, OProg)[1] *such that*

- OMem *is a probabilistic algorithm that takes as input a security parameter 1^λ and an input $x \in \{0,1\}^n$, and outputs a memory configuration $\mathbf{x}'$.*
- OProg *is a polynomial time algorithm that takes as input a security parameter 1^λ and a RAM machine M, and outputs a new* probabilistic *RAM machine M'.*

[1]ORAM is sometimes viewed as a transformation *only* on a RAM machine. When implemented in this way, a transformed machine first runs for an $\Omega(n)$-time setup phase, and therefore cannot support sublinear time computations.

An ORAM is required to satisfy the following correctness, efficiency, and security properties.

Let M be a polynomial-time RAM machine and let x be an input. Consider the probability space defined by sampling $\mathbf{x}' \leftarrow \mathsf{OMem}(1^\lambda, x)$, and define $M' := \mathsf{OProg}(1^\lambda, M)$.

Correctness and Efficiency. *If M executed on x yields output y within time T, then with all but negligible (in λ) probability, M' executed on $\mathbf{x}'$ must yield y within time $\tilde{O}(T)$ steps.*

Security. *Let addr' denote the sequence of memory addresses accessed by M' when executed on $\mathbf{x}'$. The distribution of addr' must depend (up to negligible statistical distance) only on the space usage of M, and the running time of M on x,*

Definition 2.8 (One-Way Function). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a δ-secure* one-way function *if it is computable in polynomial time, and for every polynomial size circuit ensemble $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, it holds that*

$$\Pr[f(x') = f(x)] \leq O(\delta(\lambda))$$

in the probability space defined by sampling $x \leftarrow \{0,1\}^\lambda$ and defining $x' = \mathcal{A}_\lambda(f(x))$.

Definition 2.9 (Puncturable PRFs [BW13, BGI14, KPTZ13]). *A δ-secure puncturable pseudo-random function (PPRF) family is an ensemble $\mathcal{F} = \{\mathcal{F}_\lambda\}$ of function families*

$$\mathcal{F}_\lambda = \left\{F_k : \{0,1\}^* \to \{0,1\}^{\mathbb{N}}\right\}_{k \in \mathcal{K}_\lambda},$$

each indexed by a key space $\mathcal{K}_\lambda$ that without loss of generality is $\{0,1\}^{\ell(\lambda)}$ for some polynomial $\ell(\cdot)$. The PPRF family is required to have associated polynomial time algorithms $(\mathcal{F}.\mathsf{Eval}, \mathcal{F}.\mathsf{Puncture}, \mathcal{F}.\mathsf{PuncEval})$ that satisfy the following properties.

Correctness. *For all $x \in \{0,1\}^*$, all $\lambda \in \mathbb{N}$, all $k \in \mathcal{K}_\lambda$, and all $i \in \mathbb{N}$, it holds that $\mathcal{F}.\mathsf{Eval}(k, x, i) = F_k(x)_i$.*

Punctured Correctness. *For all $S \subseteq \{0,1\}^*$, all $x \in \{0,1\}^* \setminus S$, all $\lambda \in \mathbb{N}$, all $k \in \mathcal{K}_\lambda$, and all $i \in \mathbb{N}$, it holds that*

$$\mathcal{F}.\mathsf{PuncEval}\Big(\mathcal{F}.\mathsf{Puncture}(k, S), x, i\Big) = F_k(x)_i. \tag{2.1}$$

For ease of notation, we write $F_k(x)_i$ and $\mathcal{F}.\mathsf{Eval}(k, x, i)$ interchangeably, and we write $k\{S\}$ to denote $\mathcal{F}.\mathsf{Puncture}(k, S)$. We write $F_{k\{S\}}(x)_i$ as short-hand for the left-hand side of Eq. (2.1).

Pseudorandomness. *For every polynomial-sized ensemble $\{\mathcal{A}_\lambda\}$ of (inputless) oracle circuits, it holds that*

$$\left| \Pr_{k \leftarrow \mathcal{K}_\lambda}[\mathcal{A}_\lambda^{F_k}() = 1] - \Pr_{U \leftarrow \mathrm{Func}(\{0,1\}^*, \{0,1\}^*)}[\mathcal{A}_\lambda^{U}() = 1] \right| \leq O(\delta(\lambda)).$$

Pseudorandomness at Non-Adaptively Punctured Points. *For all ensembles $\{S_\lambda\}$ of polynomial-length[2] subsets of $\{0,1\}^*$, and all polynomial-size ensembles of oracle circuits $\{\mathcal{A}_\lambda\}$, it holds that*

$$\left| \Pr_{k \leftarrow \mathcal{K}_\lambda}[\mathcal{A}_\lambda^{F_k|_{S_\lambda}}(k\{S_\lambda\}) = 1] - \Pr_{\substack{k \leftarrow \mathcal{K}_\lambda \\ U \leftarrow \mathrm{Func}(S_\lambda, \{0,1\}^{\mathbb{N}})}}[\mathcal{A}_\lambda^{U}(k\{S_\lambda\}) = 1] \right| \leq O(\delta),$$

where $F_k|_{S_\lambda}$ denotes the restriction of the function F_k to S_λ.

Remark 2.10. Our definition of a puncturable PRF family differs from the usual definition in that, for a given key k, we do not have any bound on the input or output length of F_k. Our definition implies the usual one, and is achieved by the construction of [GGM86] when combined with a prefix-free code in the natural way.

Definition 2.11 (Collision Resistant Hashing). *A length-halving* collision resistant hash family *is a distribution ensemble $\{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$, where $\mathcal{H}_\lambda$ is a distribution on circuits with 2λ input bits and λ output bits satisfying:*

Efficient Sampleability. *There is a p.p.t. algorithm that on input 1^λ outputs a sample from $\mathcal{H}_\lambda$.*

Collision Resistance. *For every polynomial size circuit ensemble $\{\mathcal{A}_\lambda\}$:*

$$\Pr\left[(x \neq x') \wedge (h(x) = h(x'))\right] \leq \mathrm{negl}(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned} h &\leftarrow \mathcal{H}_\lambda \\ (x, x') &\leftarrow \mathcal{A}_\lambda(h) \end{aligned}$$

Below we define the notion of a computational PIR scheme with polylogarithmic succinctness, and then proceed to derive a variant that will be useful for our construction.

Definition 2.12. *A 2-message private information retrieval (PIR) scheme over alphabet $\Gamma = \Gamma_\lambda$, with $\log|\Gamma| \leq \mathrm{poly}(\lambda)$, is a tuple of p.p.t. algorithms* (PIR.Send, PIR.Respond, PIR.Decode)*, where:*

- $(q, s) \leftarrow \mathsf{PIR.Send}(1^\lambda, i, N)$*: given 1^λ and some $i, N \in \mathbb{N}$ such that $i \leq N$, outputs a query string q and a secret state s.*

[2]Here by the "length" of a subset $S \subseteq \{0,1\}^*$, we mean $\sum_{x \in S} |x|$.

- $a \leftarrow \mathsf{PIR.Respond}(q, D)$: *given a query string q and a database $D \in \Gamma^N$, outputs a response string a.*

- $x \leftarrow \mathsf{PIR.Decode}(s, a)$: *given an answer a and state s, outputs an element $x \in \Gamma$.*

We say that the scheme is a polylogarithmic PIR if $|a| = \mathrm{poly}(\lambda, \log(N))$. We say that it is (perfectly) correct if for all $i \leq N \leq 2^\lambda$ and $D \in \Gamma^N$ it holds that when setting $(q, s) \leftarrow \mathsf{PIR.Send}(1^\lambda, i, N)$, $a \leftarrow \mathsf{PIR.Respond}(1^\lambda, q, D)$, and $x \leftarrow \mathsf{PIR.Decode}(1^\lambda, s, a)$, then $x = D[i]$ with probability 1.

We say that the scheme is secure if for any sequence of $N_\lambda = \mathrm{poly}(\lambda)$, $i_\lambda, i'_\lambda \leq N_\lambda$ it holds that $q \approx q'$, where: $(q, s) \leftarrow \mathsf{PIR.Send}(1^\lambda, i, N)$, $(q', s') \leftarrow \mathsf{PIR.Send}(1^\lambda, i', N)$.

It had been shown in [IKO05] that a succinct 2-message PIR implies the existence of a family of collision resistant hash functions. Succinct PIR can be constructed based on the ϕ-hiding assumption [CMS99] or based on the existence of leveled fully homomorphic encryption scheme [Gen09], which in turn can be based on the (polynomial) hardness of approximating short-vector lattice problems (SIVP and GapSVP) to within a polynomial factor in *worst case* lattices [BV14].

We define a variant that we call "succinct" PIR, where only a bound on the size of the database N needs to be known. This notion will be useful for our construction and can be derived from the standard notion of PIR as we explain below.

Definition 2.13. *A* succinct *2-message private information retrieval (PIR) scheme over alphabet $\Gamma = \Gamma_\lambda$, with $\log|\Gamma| \leq \mathrm{poly}(\lambda)$, is a tuple of p.p.t. algorithms* ($\mathsf{ScPIR.Send}$, $\mathsf{ScPIR.Respond}$, $\mathsf{ScPIR.Decode}$)*, where:*

- $(q, s) \leftarrow \mathsf{ScPIR.Send}(1^\lambda, i)$: *given 1^λ and some $i \leq 2^\lambda$, outputs a query string q and a secret state s.*

- $a \leftarrow \mathsf{ScPIR.Respond}(q, D)$: *given a query string q and a database $D \in \Gamma^{\leq 2^\lambda}$, outputs a response string a.*

- $x \leftarrow \mathsf{ScPIR.Decode}(s, a)$: *given an answer a and state s, outputs an element $x \in \Gamma$.*

We say that the scheme is a succinct PIR if $|a| = \mathrm{poly}(\lambda)$. We say that it is (perfectly) correct if for all $i \leq 2^\lambda$ and $D \in \Gamma^{\leq 2^\lambda}$ with $|D| \geq i$ it holds that when setting $(q, s) \leftarrow \mathsf{ScPIR.Send}(1^\lambda, i)$, $a \leftarrow \mathsf{ScPIR.Respond}(1^\lambda, q, D)$, and $x \leftarrow \mathsf{ScPIR.Decode}(1^\lambda, s, a)$, then $x = D[i]$ with probability 1.

We say that the scheme is secure if for any sequences i_λ, i'_λ it holds that $q \approx q'$, where: $(q, s) \leftarrow \mathsf{ScPIR.Send}(1^\lambda, i)$, $(q', s') \leftarrow \mathsf{ScPIR.Send}(1^\lambda, i')$.

Whereas the definition here is syntactically stronger (since N does not need to be known), succinct PIR can be constructed from any polylogarithmic PIR scheme as in Definition 2.12. This can be done by considering all databases of size 2^t, for $t = 1, \ldots, \lambda$, and running the query algorithm on all of them in parallel. This will incur an overhead of λ which does not change the polynomial dependence on λ. Furthermore, specific constructions of PIR, e.g. from leveled homomorphic encryption, can be adapted to the new definition without overhead at all.

Definition 2.14 (Indistinguishability Obfuscation for Circuits [BGI+01]). *A δ-secure indistinguishability obfuscator for circuits is a p.p.t. algorithm $i\mathcal{O}$ that takes as input a security parameter 1^λ and a circuit C, and outputs a circuit $\tilde{\mathbf{C}}$ such that:*

Correctness. *With probability 1, C and $\tilde{\mathbf{C}}$ compute the same function – that is, for every x, $C(x) = \tilde{\mathbf{C}}(x)$.*

Security. *Let $\left\{C_\lambda^{(0)}\right\}_\lambda$ and $\left\{C_\lambda^{(1)}\right\}_\lambda$ be any two polynomial-sized ensembles of circuits. If for every λ, the circuits $C_\lambda^{(0)}$ and $C_\lambda^{(1)}$ have the same size and compute the same function (we succinctly denote these conditions by writing $C_\lambda^{(0)} \equiv C_\lambda^{(1)}$), then the distribution ensembles $\left\{i\mathcal{O}(1^\lambda, C_\lambda^{(0)})\right\}_\lambda$ and $\left\{i\mathcal{O}(1^\lambda, C_\lambda^{(1)})\right\}_\lambda$ are computationally δ-indistinguishable.*

The security definition of indistinguishability obfuscation given above actually implies the following useful (and ostensibly stronger) security property.

Claim 2.14.1 (Distributional IO Security). *Suppose that $i\mathcal{O}$ is a δ-secure indistinguishability obfuscator for circuits, and let $\left\{\mathcal{C}_\lambda^{(0)}\right\}$ and $\left\{\mathcal{C}_\lambda^{(1)}\right\}$ be two ensembles of circuit distributions. Let $s : \mathbb{N} \to \mathbb{N}$ be a polynomially-bounded function and $\epsilon : \mathbb{N} \to \mathbb{R}$ be any function such that for all λ:*

- *Both $\mathcal{C}_\lambda^{(0)}$ and $\mathcal{C}_\lambda^{(1)}$ are distributions on size $s(\lambda)$ circuits.*
- *When sampling $\mathbf{C}^{(0)} \leftarrow \mathcal{C}_\lambda^{(0)}$ and $\mathbf{C}^{(1)} \leftarrow \mathcal{C}_\lambda^{(1)}$, the* distributions *of the functionalities[3] of $\mathbf{C}^{(0)}$ and $\mathbf{C}^{(1)}$ are $\epsilon(\lambda)$-close in total variational distance.*

Then the distributions of $\tilde{\mathbf{C}}^{(0)}$ and $\tilde{\mathbf{C}}^{(1)}$ are computationally $(\epsilon + \delta)$-indistinguishable in the probability space defined by sampling

$$\begin{aligned} \mathbf{C}^{(0)} &\leftarrow \mathcal{C}_\lambda^{(0)} \\ \mathbf{C}^{(1)} &\leftarrow \mathcal{C}_\lambda^{(1)} \\ \tilde{\mathbf{C}}^{(0)} &\leftarrow i\mathcal{O}(1^\lambda, \mathbf{C}^{(0)}) \\ \tilde{\mathbf{C}}^{(1)} &\leftarrow i\mathcal{O}(1^\lambda, \mathbf{C}^{(1)}). \end{aligned} \tag{2.2}$$

Proof. Suppose for contradiction that for some polynomial-sized circuit ensemble $\mathcal{A} = \{\mathcal{A}_\lambda\}$, it holds for infinitely many λ that

$$\left|\Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(0)}) = 1] - \Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(1)}) = 1]\right| > \omega\Big(\epsilon(\lambda) + \delta(\lambda)\Big) \tag{2.3}$$

in the probability space (2.2). Let Λ denote the set of λ for which Eq. (2.3) holds.

For $\lambda \in \Lambda$, Eq. (2.3) also holds in a probability space where:

[3]Note that we are not requiring that $\mathcal{C}_\lambda^{(0)}$ and $\mathcal{C}_\lambda^{(1)}$ themselves be statistically close – indeed, they may even be disjoint.

1. $\mathbf{C}^{(0)}$ and $\mathbf{C}^{(1)}$ are each individually distributed as in (2.2), but are jointly distributed so that
$$\Pr[\mathbf{C}^{(0)} \equiv \mathbf{C}^{(1)}] \geq 1 - \epsilon(\lambda).$$
The existence of such a joint distribution is one of many equivalent characterizations of the total variational distance.

2. Conditioned on the values of $\mathbf{C}^{(0)}$ and $\mathbf{C}^{(1)}$, the obfuscations $\tilde{\mathbf{C}}^{(0)}$ and $\tilde{\mathbf{C}}^{(1)}$ are distributed as in (2.2).

In this probability space, let EQV denote the event that $\mathbf{C}^{(0)} \equiv \mathbf{C}^{(1)}$. It then holds that

$$\begin{aligned}\omega\big(\epsilon(\lambda) + \delta(\lambda)\big) &< \Big|\Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(0)}) = 1] - \Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(1)}) = 1]\Big| \\ &\leq \Pr[\neg \text{EQV}] + \Big|\Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(0)}) = 1 \wedge \text{EQV}] - \Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(1)}) = 1 \wedge \text{EQV}]\Big|\end{aligned}$$

By averaging, there must exist a pair of *functionally equivalent* size-$s(\lambda)$ circuits $C_\lambda^{(0)}$ and $C_\lambda^{(1)}$ for every $\lambda \in \Lambda$, such that

$$\Big|\Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(0)}) = 1 | \mathbf{C}^{(0)} = C^{(0)}] - \Pr[\mathcal{A}_\lambda(\tilde{\mathbf{C}}^{(1)}) = 1 | \mathbf{C}^{(1)} = C^{(1)}]\Big| > \omega(\delta(\lambda)).$$

This contradicts the assumption that $\mathsf{i}\mathcal{O}$ is δ-secure. □

One stronger and easier to apply notion of security for a circuit obfuscator is that of differing input obfuscation. Roughly speaking, this notion requires that if it is difficult to find an input x for which $C_0(x) \neq C_1(x)$ for some distribution over circuit pairs (C_0, C_1), then an obfuscation of C_0 is computationally indistinguishable from an obfuscation of C_1.

Definition 2.15 (Differing-Inputs Sampler [ABG+13, BCP14]). *A p.p.t. algorithm* Sam *is a* differing-inputs circuit sampler *if for all polynomial size circuit ensembles* $\{\mathcal{A}_\lambda\}$*, there exists a negligible function* ν *such that:*

$$\Pr[C_0(x) \neq C_1(x)] \leq \nu(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned}&(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sam}(1^\lambda) \\ &x \leftarrow \mathcal{A}_\lambda(C_0, C_1, \mathsf{aux}).\end{aligned}$$

Definition 2.16 (Differing-Inputs Obfuscation [ABG+13, BCP14]). *$\mathcal{O}$ is a differing-inputs obfuscator for a differing-inputs circuit sampler* Sam *if* $(\tilde{C}_0, \mathsf{aux})$ *and* $(\tilde{C}_1, \mathsf{aux})$ *are computationally indistinguishable in the probability space defined by sampling*

$$\begin{aligned}&(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sam}(1^\lambda) \\ &\tilde{C}_0 \leftarrow \mathcal{O}(1^\lambda, C_0) \\ &\tilde{C}_1 \leftarrow \mathcal{O}(1^\lambda, C_1).\end{aligned}$$

Boyle, Chung and Pass show that an indistinguishability obfuscator is also a differing-input obfuscator for functions with only polynomially many differing-inputs [BCP14].

Theorem 2.17 ([BCP14]). *Suppose that for a differing-inputs circuit sampler* Sam*, it holds with probability* 1 *over sampling* $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sam}(1^\lambda)$ *that*

$$|\{x \ : \ C_0(x) \neq C_1(x)\}| \leq p(\lambda)$$

for some polynomial p*.*

Then any indistinguishability obfuscator is also a differing-input obfuscator for Sam*.*

Chapter 3

Garbling Schemes and Randomized Encodings

This chapter is based on the ITCS 2016 paper "Fully Succinct Garbled RAM" by Canetti and Holmgren.

The construction is in several stages, where each successive stage achieves a stronger notion of security. The starting point is the scheme of Koppula, Lewko, and Waters [KLW15], which can be viewed as a very weak garbling scheme for RAM machines. This scheme guarantees indistinguishability that a garbling of (M_0, x) is indistinguishable from a garbling of (M_1, x) iff the computations have identical transcripts (sequences of internal control states and memory operations). We refer to this security property as *same transcript indistinguishability.*

We next generically upgrade any garbling scheme satisfying same transcript indistinguishability into a scheme that satisfies full security. This proceeds in two steps. First, we construct a garbler that satisfies "same address indistinguishability", meaning that as long as M_0 accesses the same sequences of *addresses* on x as M_1 does, then the garblings of M_0 and M_1 will be indistinguishable.

Finally, we combine any garbling scheme satisfying same address indistinguishability with an *oblivious RAM* compiler, a notion originally due to Goldreich and Ostrovsky [GO96]. Oblivious RAMs were originally developed in the context of hiding the address access patterns of CPUs that act as a "black box". Although neither indistinguishability obfuscation nor our "same address indistinguishability" garblers truly realize this idealization, we are able to show that a specific ORAM, due to Chung and Pass [CP13], suffices in our setting.

3.1 Technical Overview

The bulk of our technical contribution lies in our construction of a fully secure garbler from one that satisfies only same address indistinguishability. We use the following formalization of an ORAM. An ORAM consists of a probabilistic algorithm OMem, which transforms an input x into an initial memory configuration $\mathbf{x}'$, and a deterministic algorithm OProg, which transforms a RAM machine M into a probabilistic

RAM machine M'.

The main difficulty for us, and the essential difference between RAM machines and Turing machines, is that in general, oblivious RAMs (ORAM) rely on randomness, which must be kept secret. In our case, we must reconcile this with the fact that we can directly garble only *deterministic* programs. Because we aim to construct a *succinct* garbler, we cannot afford to hard-wire a huge amount of randomness.

Our construction instead garbles a RAM machine M'' that is equipped with a PRF F (described by a small seed), and emulates M', at the i^{th} step using $F(i)$ as the contents of its random tape. To prove security, we will require that F is actually a *puncturable* PRF (see Definition 2.9), and we will require that the ORAM satisfies a certain "puncture-friendly" property that in fact implies the usual notions of ORAM security.

For any fixed M and x, view the sequence $\vec{a}$ of addresses accessed by M' on $\mathbf{x}'$ as a (deterministic) function of the randomness used by OMem in generating $\mathbf{x}'$ and the randomness used in the execution of M'. Think of these two sources of randomness as comprising a single random tape. We require that this function is "blockwise decomposable", in the following sense. The sequence $\vec{a}$ must be decomposable into a sequence of (short, equally-sized) blocks of i.i.d. addresses $\vec{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_T)$, such that each $\mathbf{a}_i$ is a function of a small portion of the random tape, and moreover no two blocks depend on overlapping portions. Suppose that each block consists of η sequentially accessed addresses.

To prove this construction achieves full security, we consider for any RAM machine M and input x, the distribution of $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by garbling M and x. We present a sequence of changes to the RAM program M'' which change the distribution of $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ only in a computationally indistinguishable way. Through these changes, we will obtain a distribution that depends only on $M(x)$.

To describe the changing functionality of M'', it will be convenient to describe its behavior in terms of two types of steps.

- In a *real* execution step, M'' runs M' for η steps, using the PPRF F to provide randomness as described above.
- In a *dummy* execution step, M'' samples η addresses from an appropriate "simulated" distribution, using a different PPRF G for randomness, and accesses these addresses.[1]

We progress through a sequence of distributions in which for some i, $\widetilde{\mathbf{M}}$ is the garbling of a RAM machine M'' that performs $T - i$ real execution steps, followed by i dummy steps. Our challenge is to show that incrementing i does not noticeably affect the distribution of $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$. We take a punctured programming approach [SW] to demonstrating this.

1. Because the $(T-i)^{th}$ block of accessed addresses depends on an otherwise unused part of F, we puncture F at this part, and use independent randomness in its place. This change is indistinguishable by the punctured PRF security of F.

[1]The exact type of access is irrelevant; it could be for example be a read, or a write of an arbitrary value.

2. We unpuncture F, and instead simply hard-code the addresses $\vec{a}_{T-i}$ to access in the $(T-i)^{th}$ step. We also puncture G at the places where its evaluations would have been used in the $(T-i)^{th}$ step if a dummy step were to be performed. This change is indistinguishable by the same address indistinguishability of the base garbling scheme.

3. We change the $(T-i)^{th}$ step to actually be a dummy step. This change is indistinguishable by the punctured PRF security of G.

3.1.1 RAM Machines and Computations

In this chapter, we will always use the "CPU circuit" representation of (bounded) RAMs, as described in Appendix A.2. Recall that this representation includes a syntactic bound on the worst-case space usage of the RAM machine.

Definition 3.1. *A* RAM computation *is a pair (M, x) consisting of a (bounded) RAM machine M and an input x.*

Classes of RAM Computation Ensembles In all of our constructions, we will study the asymptotic behavior (efficiency and security) on *ensembles* of RAM computations – a collection of computations $\{(M_\lambda, x_\lambda)\}_{\lambda\in\mathbb{N}}$. For the sake of cleanly stating general results, we several important *classes* of ensembles.

Definition 3.2. *An ensemble $\{(M_\lambda, x_\lambda)\}$ is* polynomial size *if for some polynomial* poly*, the sizes of both M_λ and x_λ are bounded by* $\mathrm{poly}(\lambda)$.

Definition 3.3. *For any function $T : \mathbb{N} \to \mathbb{N}$, an ensemble $\{(M_\lambda, x_\lambda)\}$ is in* $\mathsf{TIME}(T)$ *if the running time of M_λ on x_λ is $O(T(\lambda))$.*

Definition 3.4. *An ensemble $\{(M_\lambda, x_\lambda)\}$ is* polynomial time *if it is in* $\mathsf{TIME}(\lambda^c)$ *for some $c > 0$.*

3.1.2 Garbling

Definition 3.5. *A δ-secure garbling scheme for a class $\mathfrak{M}$ of RAM computations is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ *satisfying the following properties for every ensemble $\{(M_\lambda, x_\lambda)\} \in \mathfrak{M}$ of RAM computations.*

Consider the ensemble of probability spaces defined by sampling

$$\begin{aligned} \mathbf{K}_\lambda &\leftarrow \mathsf{KeyGen}(1^\lambda) \\ \widetilde{\mathbf{M}}_\lambda &\leftarrow \mathsf{GbPrg}(\mathbf{K}_\lambda, M_\lambda) \\ \tilde{\mathbf{x}}_\lambda &\leftarrow \mathsf{GbMem}(\mathbf{K}_\lambda, x_\lambda). \end{aligned} \tag{3.1}$$

Let T_λ denote the running time of M_λ on x_λ, and let S_λ denote the (syntactic) bound on the space usage of M_λ. In what follows we omit the λ subscript and instead work with expressions that are implicitly functions of λ.

Correctness. *The scheme is* correct *if with all but negligible (in λ) probability, it holds that $\widetilde{\mathbf{M}}(\tilde{\mathbf{x}}) = M(x)$.*

Efficiency. *The scheme is* efficient *if* KeyGen, GbPrg, *and* GbMem *are all probabilistic polynomial time algorithms. This property is sometimes also called* succinctness, *in contrast to schemes (e.g., [LO13]) in which the running time and output length of* GbPrg *are both proportional to the (worst-case) running time of M.*

Efficiency Preservation. *The scheme is* efficiency preserving *if*

- *The running time of $\widetilde{\mathbf{M}}$ on $\tilde{\mathbf{x}}$ is $T \cdot \mathrm{poly}(\lambda, \log T, \log S)$.*
- *$\widetilde{\mathbf{M}}$'s space usage is (syntactically) bounded by $S \cdot \mathrm{poly}(\lambda)$.*

Security. *The scheme is δ-*secure *if there is a p.p.t. algorithm* Sim *such that the distribution of $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ in (3.1) is computationally δ-indistinguishable from*

$$\mathsf{Sim}(1^\lambda, M(x), |M|, |x|, T, S).$$

Theorem 3.6. *If δ-secure one-way functions, δ-secure injective pseudo-random generators, and δ-secure indistinguishability obfuscation for circuits exist, then there is a δ-secure garbling scheme for the class of polynomial size, polynomial time RAM computations.*

Sections 3.2 to 3.5 are devoted to proving Theorem 3.6.

3.2 Same Transcript Indistinguishability

We first construct a garbling scheme satisfying a very weak form of security. Recall that a RAM machine's *computation transcript* on an input s is a list of each of the successive memory configurations and internal control states reached in an execution on s.

Definition 3.7. *A garbling scheme $\mathcal{G} = (\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ for $\mathfrak{M}$ satisfies* same-transcript δ-indistinguishability *if for all RAM computation ensembles $\{(M_\lambda^{(0)}, x_\lambda)\}$ and $\{(M_\lambda^{(1)}, x_\lambda)\}$ in $\mathfrak{M}$, the following holds.*

If for all λ, the executions of $M_\lambda^{(0)}$ on x_λ and of $M_\lambda^{(1)}$ on x_λ produce identical transcripts, then the garblings $(\widetilde{\mathbf{M}}_0, \tilde{\mathbf{x}})$ and $(\widetilde{\mathbf{M}}_1, \tilde{\mathbf{x}})$ are computationally δ-indistinguishable in the probability space defined by sampling

$$\begin{aligned}
&\mathbf{K} \leftarrow \mathsf{KeyGen}(1^\lambda)\\
&\widetilde{\mathbf{M}}_0 \leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(0)})\\
&\widetilde{\mathbf{M}}_1 \leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(1)})\\
&\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}(\mathbf{K}, x_\lambda).
\end{aligned}$$

The non-triviality of Definition 3.7 stems from the possibility that two RAM machines M_0 and M_1 may produce the same transcripts only when executed on a subset of their possible inputs. Moreover, M_0 and M_1 may be distinguishable by a malicious evaluator that tampers with their memories and internal control states.

Theorem 3.8. *If δ-secure one-way functions, δ-secure injective pseudo-random generators, and δ-secure indistinguishability obfuscation for circuits exist, then there is a garbling scheme that satisfies* same-transcript $(T \cdot \delta)$-indistinguishability *for the class of polynomial size, time-T RAM computations.*

Theorem 3.8 is implicit in [KLW15]. A stronger version of this theorem (with marginally stronger assumptions) is proved in Chapter 5.

3.3 Same Memory Indistinguishability

Again assuming that one-way functions exist, we upgrade any garbling scheme that satisfies same-transcript indistinguishability into one that satisfies a slightly stronger notion of security, which we call same-memory indistinguishability. A garbling scheme is said to satisfy same-memory indistinguishability if the garblings of two different machine-input pairs are indistinguishable as long as they induce the same memory accesses. Notably, it is possible for the two machines to have differing sequences of internal control states.

Definition 3.9. *A garbling scheme* $(\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ *for $\mathfrak{M}$ satisfies* same-memory δ-indistinguishability *if for all equally-sized ensembles of RAM computations $\{(M_\lambda^{(0)}, x_\lambda)\}$ and $\{(M_\lambda^{(1)}, x_\lambda)\}$ in $\mathfrak{M}$, the following holds.*

If for each λ, it holds that $M_\lambda^{(0)}(x_\lambda) = M_\lambda^{(1)}(x_\lambda)$ and additionally the two corresponding transcripts differ only on their sequences of internal control states, then the garblings $(\widetilde{\mathbf{M}}_0, \tilde{\mathbf{x}})$ and $(\widetilde{\mathbf{M}}_1, \tilde{\mathbf{x}})$ are computationally δ-indistinguishable in the probability space defined by sampling

$$\begin{aligned}
&\mathbf{K} \leftarrow \mathsf{KeyGen}(1^\lambda) \\
&\widetilde{\mathbf{M}}_0 \leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(0)}) \\
&\widetilde{\mathbf{M}}_1 \leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(1)}) \\
&\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}(\mathbf{K}, x_\lambda).
\end{aligned}$$

3.3.1 Construction

Given a garbling scheme $\mathcal{G}' = (\mathsf{KeyGen}', \mathsf{GbPrg}', \mathsf{GbMem}')$ satisfying same-transcript indistinguishability and a puncturable PRF family $\mathcal{F} = \{\mathcal{F}_\lambda\}$, we construct a garbling scheme $\mathcal{G} = (\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ satisfying same-memory indistinguishability.

Construction 3.10. *We define $\mathcal{G}$ as follows:*

- $\mathsf{KeyGen}(1^\lambda)$ *samples* $K' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ *and samples* $F \leftarrow \mathcal{F}_\lambda$, *and outputs* $K = (1^\lambda, K', F)$.

- $\mathsf{GbPrg}(K, M)$ *outputs* $\mathsf{GbPrg}'(K', M')$*, where* M' *is a RAM machine defined as follows.*

  If M *is given by a CPU circuit*

  $$C : Q \times W \to Q \times \mathsf{Ops}_{A,W}$$

  with initial, accepting, and rejecting states q_0, q_{acc}, *and* q_{rej} *in* Q*, then* M' *is given by the CPU circuit*

  $$C'[\lambda, C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F] : Q' \times W \to Q' \times \mathsf{Ops}_{A,W},$$

  defined in Algorithm 1 with $Q' \stackrel{\mathsf{def}}{=} \{0,1\}^{\ell'}$ *for* ℓ' *large enough that*

  $$Q' \supseteq (\{0,1\}^{\lambda} \times Q) \sqcup \{q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}}\}$$

  for some arbitrary fixed values $q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}} \notin \{0,1\}^{\lambda} \times Q$*. The initial, accepting, and rejecting states for* $C'[\lambda, C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F]$ *are respectively defined to be* q'_0*,* q'_{acc}*, and* q'_{rej}*.*

- $\mathsf{GbMem}(K, x)$ *outputs* $\mathsf{GbMem}'(K', x)$*.*

Input: state $q'_{\mathsf{in}} \in Q'$, read memory word $w_{\mathsf{read}} \in W$
if $q'_{\mathsf{in}} = q'_0$ **then** $t := 0$; $q_{\mathsf{in}} := q_0$;
else
  Parse q'_{in} as (t, c);
  $q_{\mathsf{in}} := F(t) \oplus c$;
$(q_{\mathsf{out}}, \mathsf{op}) := C(q_{\mathsf{in}}, w_{\mathsf{read}})$;
if $q_{\mathsf{out}} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op})$;
if $q_{\mathsf{out}} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op})$;
$q'_{\mathsf{out}} := (t+1, F(t+1) \oplus q_{\mathsf{out}})$;
return $(q'_{\mathsf{out}}, \mathsf{op})$;

Algorithm 1: CPU circuit $C'[\lambda, C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F]$. Emulates C, but encrypts the RAM's internal control state, using the PPRF F to provide one-time pads.

3.3.2 Proof of Security

Theorem 3.11. *If* $\mathcal{G}'$ *satisfies same transcript* δ*-indistinguishability for polynomial size, time-*T *computations, and if* $\mathcal{F}$ *is a* δ*-secure puncturable PRF family, then* $\mathcal{G}$ *as in Construction 3.10 satisfies same memory* $(T \cdot \delta)$*-indistinguishability for polynomial size, time-*T *computations.*

Proof. Let $\{(M_\lambda^{(0)}, x_\lambda)\}$ and $\{(M_\lambda^{(1)}, x_\lambda)\}$ be two polynomial size, time-$T(\cdot)$ ensembles of RAM computations that satisfy the preconditions of Definition 3.9. Namely, for

all λ, $M_\lambda^{(0)}(x_\lambda) = M_\lambda^{(1)}(x_\lambda)$, and the two corresponding computation transcripts differ only in their sequences of internal control states.

To prove Theorem 3.11, we need to show that two distribution ensembles, which we denote by $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{RW}_\lambda^{(1)}\}$, are computationally $(T \cdot \delta)$-indistinguishable. Specifically, for $b \in \{0,1\}$, define $\mathcal{RW}_\lambda^{(b)}$ to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling

$$\begin{aligned}\mathbf{K} &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ \widetilde{\mathbf{M}} &\leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(b)})\\ \tilde{\mathbf{x}} &\leftarrow \mathsf{GbMem}(\mathbf{K}, x_\lambda).\end{aligned}$$

We prove the indistinguishability of $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{RW}_\lambda^{(1)}\}$ via a hybrid argument. Let $T_\lambda \leq T(\lambda)$ denote the running time of M_λ on x_λ. For each λ, we define T_λ hybrid distributions $\mathcal{H}_\lambda^{(0)}, \ldots, \mathcal{H}_\lambda^{(T_\lambda - 1)}$.

Hybrid $\mathcal{H}_\lambda^{(i)}$: For $0 \leq i \leq T_\lambda - 1$, hybrid $\mathcal{H}_\lambda^{(i)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and $F \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a RAM machine given by CPU circuit

   $$C_1'[\lambda, C_0, C_1, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, T_\lambda - i - 1, \mathsf{out}^*, F] : Q' \times W \to Q' \times \mathsf{Ops}_{A,W}$$

   defined in Algorithm 1 with the following parameters:

   - C_0 and C_1 are the CPU circuits that define the RAM machines $M_\lambda^{(0)}$ and $M_\lambda^{(1)}$.
   - q_0 is the initial state of $M_\lambda^{(0)}$, while q_{acc} and q_{rej} are the accepting (resp., rejecting) states of $M_\lambda^{(1)}$.
   - If the computation transcript of $M_\lambda^{(1)}$ on x_λ is

     $$(q_0^{(1)}, \mathsf{op}_0^{(1)}), \ldots, (q_{T_\lambda}^{(1)}, \mathsf{op}_{T_\lambda}^{(1)}),$$

     then $\mathsf{out}^* \stackrel{\mathsf{def}}{=} (q', \mathsf{op}_{T_\lambda - i}^{(1)})$, where

     $$q' = \begin{cases} q'_{\mathsf{acc}} & \text{if } q = q_{\mathsf{acc}} \\ q'_{\mathsf{rej}} & \text{if } q = q_{\mathsf{rej}} \\ (T_\lambda - i, F(T_\lambda - i) \oplus q_{T_\lambda - i}^{(1)}) & \text{if } q \notin \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\} \end{cases}$$

     The initial, accepting, and rejecting states for $C_1'[\lambda, C_0, C_1, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, T_\lambda - i - 1, \mathsf{out}^*, F]$ are respectively defined to be q_0', q'_{acc}, and q'_{rej}.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', x_\lambda)$.

Theorem 3.11 directly follows from Claims 3.11.1 and 3.11.2 below.

Input: state $q'_{\mathsf{in}} \in Q'$, read memory word w_{read}

if $q'_{\mathsf{in}} = q'_0$ **then** $t := 0$; $q_{\mathsf{in}} := q_0$;
else
  Parse q'_{in} as (t, c);
  $q_{\mathsf{in}} := F(t) \oplus c$;
if $t = t^*$ **then return** out^* ;
if $t < t^*$ **then** $(q_{\mathsf{out}}, \mathsf{op}) := C_0(q_{\mathsf{in}}, w_{\mathsf{read}})$;
if $t > t^*$ **then** $(q_{\mathsf{out}}, \mathsf{op}) := C_1(q_{\mathsf{in}}, w_{\mathsf{read}})$;
if $q_{\mathsf{out}} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op})$;
if $q_{\mathsf{out}} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op})$;
$q'_{\mathsf{out}} := (t+1, F(t+1) \oplus q_{\mathsf{out}})$;
return $(q'_{\mathsf{out}}, \mathsf{op})$;

Algorithm 2: CPU circuit $C'_1[\lambda, C_0, C_1, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, t^*, \mathsf{out}^*, F]$

Claim 3.11.1. *The distribution ensembles* $\{\mathcal{H}_\lambda^{(0)}\}$ *and* $\{\mathcal{RW}_\lambda^{(0)}\}$ *are computationally* δ*-indistinguishable, and likewise* $\{\mathcal{H}_\lambda^{(T_\lambda - 1)}\}$ *and* $\{\mathcal{RW}_\lambda^{(1)}\}$ *are computationally* δ*-indistinguishable.*

Proof. This directly follows from the assumption that $\mathcal{G}'$ satisfies same transcript δ-indistinguishability. □

Claim 3.11.2. *For all* $\{i_\lambda\}$ *such that* $0 \leq i_\lambda < T_\lambda - 1$ *for every* λ*, the distribution ensembles* $\{\mathcal{H}_\lambda^{(i_\lambda)}\}$ *and* $\{\mathcal{H}_\lambda^{(i_\lambda+1)}\}$ *are computationally* δ*-indistinguishable.*

Proof. We define hybrid distributions $\mathcal{H}_\lambda^{(i,1)}$ and $\mathcal{H}_\lambda^{(i,2)}$ and show that

$$\{\mathcal{H}_\lambda^{(i_\lambda)}\} \approx_\delta \{\mathcal{H}_\lambda^{(i_\lambda,1)}\} \approx_\delta \{\mathcal{H}_\lambda^{(i_\lambda,2)}\} \approx_\delta \{\mathcal{H}^{(i_\lambda+1)}\}.$$

Hybrid $\mathcal{H}_\lambda^{(i,1)}$: Hybrid $\mathcal{H}_\lambda^{(i,1)}$ is defined as the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $K' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and $F \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(K', M')$, where the RAM machine M' is given by the CPU circuit $C'_2[\lambda, C_0, C_1, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, T_\lambda - i - 2, \mathsf{out}^*, \mathsf{out}^{**}, F]$ defined in Algorithm 3 with the following parameters:
   - C_0 and C_1 are the CPU circuits that define the RAM machines $M_\lambda^{(0)}$ and $M_\lambda^{(1)}$.
   - q_0 is the initial state of $M_\lambda^{(0)}$, while q_{acc} and q_{rej} are the accepting (resp., rejecting) states of $M_\lambda^{(1)}$.
   - If the computation transcript of $M_\lambda^{(0)}$ on x_λ is

$$(q_0^{(0)}, \mathsf{op}_0^{(0)}), \ldots, (q_{T_\lambda}^{(0)}, \mathsf{op}_{T_\lambda}^{(0)}),$$

then $\mathsf{out}^* \stackrel{\mathsf{def}}{=} (q^*, \mathsf{op}^{(0)}_{T-i-1})$, where $q^* \stackrel{\mathsf{def}}{=} (T-i-1, F(T-i-1) \oplus q^{(0)}_{T-i-1})$.

- If the computation transcript of $M^{(1)}_\lambda$ on x_λ is

$$(q^{(1)}_0, \mathsf{op}^{(1)}_0), \ldots, (q^{(1)}_{T_\lambda}, \mathsf{op}^{(1)}_{T_\lambda}),$$

then $\mathsf{out}^{**} \stackrel{\mathsf{def}}{=} (q^{**}, \mathsf{op}^{(1)}_{T_\lambda - i})$, where $q^{**} \stackrel{\mathsf{def}}{=} (T-i, F(T-i) \oplus q^1_{T-i})$.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(K', x_\lambda)$.

Input: state $q'_{\mathsf{in}} \in Q'$, read word w_{read}
1 **if** $q'_{\mathsf{in}} = q'_0$ **then** $t := 0$; $q_{\mathsf{in}} := q_0$;
2 **else**
3 Parse q'_{in} as (t, c);
4 **if** $t = t^*$ **then return** out^* ;
5 **if** $t = t^* + 1$ **then return** out^{**} ;
6 $q_{\mathsf{in}} := F(t) \oplus c$;
7 **if** $t < t^*$ **then** $(q_{\mathsf{out}}, \mathsf{op}) := C_0(q_{\mathsf{in}}, w_{\mathsf{read}})$;
8 **if** $t > t^* + 1$ **then** $(q_{\mathsf{out}}, \mathsf{op}) := C_1(q_{\mathsf{in}}, w_{\mathsf{read}})$;
9 **if** $q_{\mathsf{out}} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op})$;
10 **if** $q_{\mathsf{out}} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op})$;
11 $q'_{\mathsf{out}} := (t+1, F(t+1) \oplus q_{\mathsf{out}})$;
12 **return** $(q'_{\mathsf{out}}, \mathsf{op})$;

Algorithm 3: CPU circuit $C'_2[\lambda, C_0, C_1, t^*, \mathsf{out}^*, \mathsf{out}^{**}, F]$

Hybrid $\mathcal{H}^{(i,2)}$: Hybrid $\mathcal{H}^{(i,2)}$ is defined as the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $K' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and $F \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(K', M')$, where the RAM machine M' is given by the CPU circuit $C'_2[\lambda, C_0, C_1, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, T_\lambda - i - 1, \mathsf{out}^*, \mathsf{out}^{**}, F]$ described in Algorithm 3, with the following parameters.
   - C_0 and C_1 are the CPU circuits that define the RAM machines $M^{(0)}_\lambda$ and $M^{(1)}_\lambda$.
   - q_0 is the initial state of $M^{(0)}_\lambda$, while q_{acc} and q_{rej} are the accepting (resp., rejecting) states of $M^{(1)}_\lambda$.
   - If the computation transcript of $M^{(1)}_\lambda$ on x_λ is

$$(q^{(1)}_0, \mathsf{op}^{(1)}_0), \ldots, (q^{(1)}_{T_\lambda}, \mathsf{op}^{(1)}_{T_\lambda}),$$

then $\mathsf{out}^* \stackrel{\mathsf{def}}{=} (q^*, \mathsf{op}^{(1)}_{T-i-1})$, where $q^* \stackrel{\mathsf{def}}{=} (T-i-1, F(T-i-1) \oplus q^{(1)}_{T-i-1})$. and $\mathsf{out}^{**} \stackrel{\mathsf{def}}{=} (q^{**}, \mathsf{op}^{(1)}_{T_\lambda - i})$, where $q^{**} \stackrel{\mathsf{def}}{=} (T-i, F(T-i) \oplus q^{(1)}_{T-i})$.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(K', x_\lambda)$.

We need to establish that

$$\{\mathcal{H}_\lambda^{(i_\lambda)}\} \approx \{\mathcal{H}_\lambda^{(i_\lambda,1)}\} \approx \{\mathcal{H}_\lambda^{(i_\lambda,2)}\} \approx \{\mathcal{H}^{(i_\lambda+1)}\}.$$

The first and third indistinguishabilities follow by reduction to the fixed transcript security of $\mathcal{G}'$. The second indistinguishability follows by reduction to the pseudorandomness of F at the selectively punctured point $T_\lambda - i_\lambda$. □

This concludes the proof of Theorem 3.11. □

3.4 Same Address Indistinguishability

We now to construct a slightly stronger notion of garbling, which we call same-address indistinguishability. Roughly, this security property requires that the *data* in memory is hidden, but does not require that the sequence of accessed addresses is hidden. As discussed in the introduction, in applications where the memory access pattern is known not to leak sensitive information, this notion of garbling has the potential to be significantly more efficient. In particular, the garbling scheme that we now construct preserves the efficacy of memory caches, for which real-world RAM programs are extensively optimized.

Definition 3.12. *A garbling scheme* $(\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ *for* $\mathfrak{M}$ *satisfies* same address δ-indistinguishability *if for all ensembles of equally-sized RAM computations* $\{(M_\lambda^{(0)}, x_\lambda^{(0)})\}$ *and* $\{(M_\lambda^{(1)}, x_\lambda^{(1)})\}$ *in* $\mathfrak{M}$*, the following holds.*

If for each λ, $M_\lambda^{(0)}(x_\lambda^{(0)}) = M_\lambda^{(1)}(x_\lambda^{(1)})$ *and additionally the two corresponding computation transcripts differ only in the addresses accessed, then the garblings* $(\widetilde{\mathbf{M}}_0, \tilde{\mathbf{x}}_0)$ *and* $(\widetilde{\mathbf{M}}_1, \tilde{\mathbf{x}}_1)$ *are computationally* δ*-indistinguishable in the probability space defined by sampling*

$$\begin{aligned}
\mathbf{K} &\leftarrow \mathsf{KeyGen}(1^\lambda) \\
\widetilde{\mathbf{M}}_0 &\leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(0)}) \\
\widetilde{\mathbf{M}}_1 &\leftarrow \mathsf{GbPrg}(\mathbf{K}, M_\lambda^{(1)}) \\
\tilde{\mathbf{x}}_0 &\leftarrow \mathsf{GbMem}(\mathbf{K}, x_\lambda^{(0)}) \\
\tilde{\mathbf{x}}_1 &\leftarrow \mathsf{GbMem}(\mathbf{K}, x_\lambda^{(1)})
\end{aligned}$$

3.4.1 Construction

Given a garbling scheme $\mathcal{G}' = (\mathsf{KeyGen}', \mathsf{GbPrg}', \mathsf{GbMem}')$ satisfying same memory indistinguishability and a puncturable PRF family $\{\mathcal{F}_\lambda\}$, we upgrade $\mathcal{G}'$ into a garbling scheme $\mathcal{G} = (\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ satisfying same address indistinguishability.

Overview. Our construction of $\mathcal{G}$ essentially applies $\mathcal{G}'$ to a transformed version of the machine M and a correspondingly transformed version of the input x. The transformed machine, which we will denote by M', differs from M in three ways:

- M' executes two copies of M in parallel (thereby using twice as much memory). We think of these two copies as an 'A' execution and a 'B' execution. We think of the external storage of M' as correspondingly consisting of an 'A' track and a 'B' track. We implement the 'A' and 'B' tracks by modifying the memory alphabet Σ to hold two symbols.

- M' writes metadata alongside each value to indicate the time and address at which it is written.

- M' masks each value it writes: instead of writing (t, a, v, v) to an address a, it writes $\big(t, a, F_A(t, a) \oplus v, F_B(t, a) \oplus v\big)$, where F_A and F_B are puncturable pseudorandom functions.

Construction 3.13. *We define* $\mathcal{G} = (\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ *as follows:*

- $\mathsf{KeyGen}(1^\lambda)$ *samples* $K' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ *as well as puncturable PRFs* F_A, F_B *from* $\mathcal{F}_\lambda$*, and outputs* $K \stackrel{\mathsf{def}}{=} (1^\lambda, K', F_A, F_B)$.

- $\mathsf{GbPrg}(K, M)$ *outputs* $\mathsf{GbPrg}'(K', M')$*, where* M' *is a RAM machine defined as follows.*

  If the CPU circuit representation of M *is* $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ *with*

  $$C : Q \times W \to Q \times \mathsf{Ops}_{A,W},$$

  then the CPU circuit representation of M' *is*

  $$(C'[\lambda, C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F_A, F_B], q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}}),$$

  where:

  - *The CPU circuit*

    $$C'[\lambda, C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F_A, F_B] : Q' \times W' \to Q' \times \mathsf{Ops}_{A,W'}$$

    is defined in Algorithm 4 with

    $$Q' \stackrel{\mathsf{def}}{=} \{0,1\}^\lambda \times Q \times Q$$

    and

    $$W' \stackrel{\mathsf{def}}{=} \{0,1\}^\lambda \times A \times W \times W.$$

  - $q'_0 \stackrel{\mathsf{def}}{=} (0, q_0, q_0)$, $q'_{\mathsf{acc}} \stackrel{\mathsf{def}}{=} (0, q_{\mathsf{acc}}, q_{\mathsf{acc}})$, *and* $q'_{\mathsf{rej}} \stackrel{\mathsf{def}}{=} (0, q_{\mathsf{rej}}, q_{\mathsf{rej}})$.

- $\mathsf{GbMem}(K, x)$ *outputs* $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(K', x')$*, where* x' *is defined so that its* a^{th} *word is*

  $$x'_a \stackrel{\mathsf{def}}{=} (0, a, F_A(0, a) \oplus x_a, F_B(0, a) \oplus x_a).$$

Input: State $(t, q_{\mathsf{in}}^{(A)}, q_{\mathsf{in}}^{(B)})$, read memory word $(t_{\mathsf{read}}, a_{\mathsf{read}}, w_{\mathsf{read}}^{(A)}, w_{\mathsf{read}}^{(B)})$

$(q_{\mathsf{out}}, \mathsf{op}) := C(q_{\mathsf{in}}^{(A)}, F_A(t_{\mathsf{read}}, a_{\mathsf{read}}) \oplus w_{\mathsf{read}}^{(A)})$;

if $q_{\mathsf{out}} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op})$;

if $q_{\mathsf{out}} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op})$;

Parse op as $(a_{\mathsf{write}}, w_{\mathsf{write}})$;

$\mathsf{op}' := (a_{\mathsf{write}}, (t, a_{\mathsf{write}}, F_A(t, a_{\mathsf{write}}) \oplus w_{\mathsf{write}}, F_B(t, a_{\mathsf{write}}) \oplus w_{\mathsf{write}}))$;

$q'_{\mathsf{out}} := (t+1, q_{\mathsf{out}}, q_{\mathsf{out}})$;

return $(q'_{\mathsf{out}}, \mathsf{op}')$;

Algorithm 4: CPU circuit $C'[\lambda, C, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F_A, F_B]$

3.4.2 Proof of Security

Theorem 3.14. *Suppose that $\mathcal{G}'$ satisfies same memory δ-indistinguishability for polynomial size, time-T computations, and that $\{\mathcal{F}_\lambda\}$ is a δ-secure family of puncturable PRFs. Then $\mathcal{G}$ as in Construction 3.13 is a garbling scheme that satisfies same address $(T \cdot \delta)$-indistinguishability for polynomial size, time-T computations.*

Proof. Let $\{(M_\lambda^{(0)}, x_\lambda^{(0)})\}$ and $\{(M_\lambda^{(1)}, x_\lambda^{(1)})\}$ be two polynomial size, time-$T(\cdot)$ ensembles of RAM computations that satisfy the preconditions of Definition 3.12. Namely, for all λ, the executions of $M_\lambda^{(0)}$ on $x_\lambda^{(0)}$ and of $M_\lambda^{(1)}$ on $x_\lambda^{(1)}$ produce computation transcripts that differ only in the sequence of accessed memory addresses. Let $T_\lambda = T(\lambda)$ denote the common running time of $M_\lambda^{(0)}$ on $x_\lambda^{(0)}$ and of $M_\lambda^{(1)}$ on $x_\lambda^{(1)}$. Let $C_\lambda^{(0)}$ and $C_\lambda^{(1)}$ respectively denote the CPU circuits of $M_\lambda^{(0)}$ and $M_\lambda^{(1)}$.

We want to show that two "real world" distribution ensembles, which we denote by $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{RW}_\lambda^{(1)}\}$, are indistinguishable.

Real world b: For $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, the distribution $\mathcal{RW}_\lambda^{(b)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling

$$\begin{aligned}\mathbf{K} &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ \widetilde{\mathbf{M}} &\leftarrow \mathsf{GbPrg}(M)\\ \tilde{\mathbf{x}} &\leftarrow \mathsf{GbInp}(x).\end{aligned}$$

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and puncturable PRFs $F_A, F_B \leftarrow \mathcal{F}_\lambda$.

2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows. Suppose that the CPU circuit representation of $M_\lambda^{(b)}$ is $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$. Then the CPU circuit representation of M' is

$$(C'[\lambda, C, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F_A, F_B], q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}})$$

where the CPU circuit template C' is defined in Algorithm 4, and:

- $q'_0 \stackrel{\mathsf{def}}{=} (0, q_0, q_0)$;

- $q'_{\mathsf{acc}} \stackrel{\mathsf{def}}{=} (0, q_{\mathsf{acc}}, q_{\mathsf{acc}})$; and
- $q'_{\mathsf{rej}} \stackrel{\mathsf{def}}{=} (0, q_{\mathsf{rej}}, q_{\mathsf{rej}})$.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', x')$, where the a^{th} word of x' is

$$x'_a \stackrel{\mathsf{def}}{=} (0, a, F_A(0,a) \oplus (x_\lambda^{(b)})_a, F_B(0,a) \oplus (x_\lambda^{(b)})_a).$$

We show that $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{RW}_\lambda^{(1)}\}$ are computationally $(T \cdot \delta)$-indistinguishable via a hybrid argument. That is, we define several hybrid distribution ensembles $\{\mathcal{H}_\lambda^{(0)}\}$, $\ldots$, $\{\mathcal{H}_\lambda^{(6)}\}$ such that

$$\{\mathcal{RW}_\lambda^{(0)}\} \approx \{\mathcal{H}_\lambda^{(0)}\} \approx \cdots \approx \{\mathcal{H}_\lambda^{(6)}\} \approx \{\mathcal{RW}_\lambda^{(1)}\}.$$

Each hybrid is an instantiation of a distribution template $\mathcal{H}_\lambda[\gamma, x_A, x_B, C_A, C_B, i]$ that we define below. The parameters used for each hybrid are listed in Table 3.1.

For $\gamma \in \{A, B\}$, inputs x_A and x_B, CPU circuits C_A and C_B, and $i \in \{0, \ldots, T_\lambda - 1\}$, the hybrid $\mathcal{H}_\lambda[\gamma, x_A, x_B, C_A, C_B, i]$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ and $F_A, F_B \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows. Suppose that the initial, accepting, and rejecting states of C_γ are q_0, q_{acc}, and q_{rej}. Then M' is given by the CPU circuit $C'_1[\lambda, \gamma, i, C_A, C_B, q_{\mathsf{acc}}, q_{\mathsf{rej}}, F_A, F_B]$ as defined in Algorithm 4.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', x')$, where the a^{th} word of x' is

$$x'_a \stackrel{\mathsf{def}}{=} (0, a, F_A(0,a) \oplus (x_A)_a, F_B(0,a) \oplus (x_B)_a).$$

Hybrid:	$\mathcal{H}_\lambda^{(0)}$	$\mathcal{H}_\lambda^{(1)}$	$\mathcal{H}_\lambda^{(2)}$	$\mathcal{H}_\lambda^{(3)}$	$\mathcal{H}_\lambda^{(4)}$	$\mathcal{H}_\lambda^{(5)}$	$\mathcal{H}_\lambda^{(6)}$
γ:	A			B			A
x_A:	$x_\lambda^{(0)}$					$x_\lambda^{(1)}$	
x_B:	$x_\lambda^{(0)}$	$x_\lambda^{(1)}$					
C_A:	$C_\lambda^{(0)}$					$C_\lambda^{(1)}$	
C_B:	$C_\lambda^{(0)}$	$C_\lambda^{(1)}$					
i:	0		T_λ		0		

Table 3.1: Top-level hybrid experiments in the proof of Theorem 3.14.

We first dispense with the easier claims.

Claim 3.14.1. *The distribution ensembles $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{H}_\lambda^{(0)}\}$ are computationally δ-indistinguishable.*

Input: State $(t, q_{\mathsf{in}}^{(A)}, q_{\mathsf{in}}^{(B)})$, read memory word $(t_{\mathsf{read}}, a_{\mathsf{read}}, w_{\mathsf{read}}^{(A)}, w_{\mathsf{read}}^{(B)})$

$\left(q_{\mathsf{out}}^{(\gamma)}, (a_{\mathsf{write}}^{(\gamma)}, w_{\mathsf{write}}^{(\gamma)})\right) := C_\gamma(q_{\mathsf{in}}^{(\gamma)}, F_\gamma(t_{\mathsf{read}}, a_{\mathsf{read}}) \oplus w_{\mathsf{read}}^{(\gamma)})$;

if $t \leq i$ **then**

  `// Also execute track` $\bar{\gamma}$

  $\left(q_{\mathsf{out}}^{(\bar{\gamma})}, (a_{\mathsf{write}}^{(\bar{\gamma})}, w_{\mathsf{write}}^{(\bar{\gamma})})\right) := C_{\bar{\gamma}}(q_{\mathsf{in}}^{(\bar{\gamma})}, F_{\bar{\gamma}}(t_{\mathsf{read}}, a_{\mathsf{read}}) \oplus w_{\mathsf{read}}^{(\bar{\gamma})})$;

  $\mathsf{op}' := (a_{\mathsf{write}}^{(\gamma)}, (t+1, a_{\mathsf{write}}^{(\gamma)}, F_A(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(A)}, F_B(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(B)}))$;

else

  `// Duplicate write to track` γ `on track` $\bar{\gamma}$

  $\mathsf{op}' := (a_{\mathsf{write}}^{(\gamma)}, (t+1, a_{\mathsf{write}}^{(\gamma)}, F_A(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(\gamma)}, F_B(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(\gamma)}))$;

if $q_{\mathsf{out}}^{(\gamma)} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op}')$;

if $q_{\mathsf{out}}^{(\gamma)} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op}')$;

$q'_{\mathsf{out}} := (t+1, q_{\mathsf{out}}^{(\gamma)}, q_{\mathsf{out}}^{(\gamma)})$;

return $(q'_{\mathsf{out}}, \mathsf{op}')$;

Algorithm 5: CPU circuit $C'_1[\lambda, \gamma, i, C_A, C_B, F_A, F_B]$. q_{acc} and q_{rej} denote the accepting and rejecting states, respectively, of C_γ. $\bar{\gamma}$ denotes B if γ is A, and vice versa.

Proof. For each λ, the distributions $\mathcal{RW}_\lambda^{(0)}$ and $\mathcal{H}_\lambda^{(0)}$ are each defined via an experiment that samples

$$\mathbf{K} \leftarrow \mathsf{KeyGen}'(1^\lambda)$$
$$\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}, M')$$
$$\tilde{\mathbf{x}} \leftarrow \mathsf{GbInp}'(\mathbf{K}, x')$$

(for a different choice of M' in the two experiments). In both cases though, the memory accesses made by M' on x' are identical. Thus, the indistinguishability of $\mathcal{RW}_\lambda^{(0)}$ and $\mathcal{H}_\lambda^{(0)}$ follows directly from the same memory indistinguishability of the garbling scheme $\mathcal{G}'$. □

Claim 3.14.2. *The distribution ensembles $\{\mathcal{H}_\lambda^{(5)}\}$ and $\{\mathcal{RW}_\lambda^{(1)}\}$ are computationally δ-indistinguishable.*

Proof. Completely analogous to Claim 3.14.1. □

Claim 3.14.3. *The distribution ensembles $\{\mathcal{H}_\lambda^{(0)}\}$ and $\{\mathcal{H}_\lambda^{(1)}\}$ are computationally $(n \cdot \delta)$-indistinguishable, where $n(\lambda)$ is the (common) length of $x_\lambda^{(0)}$ and $x_\lambda^{(1)}$.*

Proof. For each λ, we define hybrid distributions $\mathcal{H}_\lambda^{(0,0)}$ and $\mathcal{H}_\lambda^{(0,1)}$ such that (ignoring parameters)

$$\{\mathcal{H}_\lambda^{(0)}\} \approx \{\mathcal{H}_\lambda^{(0,0)}\} \approx \{\mathcal{H}_\lambda^{(0,1)}\} \approx \{\mathcal{H}_\lambda^{(1)}\}.$$

$\mathcal{H}_\lambda^{(0,0)}$ is defined nearly identically to $\mathcal{H}_\lambda^{(0)}$, differing only in the CPU circuit used to define M'. The CPU circuit is still Algorithm 5, but with a parameter choice that differs as follows:

- $\mathcal{H}_\lambda^{(0,0)}$ uses a key for F_B that is punctured at the set of points $\{(0,a) : a \in \{0,1\}^\lambda\}$.[2]

- $\mathcal{H}_\lambda^{(0,0)}$ sets $C_B = C_\lambda^{(1)}$ rather than $C_B = C_\lambda^{(0)}$.

These changes to M' do not affect its execution on x' ($i = 0$, so Line 3 – the only place where $F_B(0,\cdot)$ or C_B could potentially be used – is never executed). Thus, it indeed holds that $\{\mathcal{H}_\lambda^{(0,0)}\}$ and $\{\mathcal{H}_\lambda^{(0)}\}$ are δ-indistinguishable by the same access δ-indistinguishability of $\mathcal{G}'$.

$\mathcal{H}_\lambda^{(0,1)}$ is similarly defined nearly identically to $\mathcal{H}_\lambda^{(1)}$, differing only in that F_B is punctured in the same way as above. By identical reasoning, $\{\mathcal{H}_\lambda^{(0,2)}\}$ and $\{\mathcal{H}_\lambda^{(1)}\}$ are δ-indistinguishable.

$\mathcal{H}_\lambda^{(0,0)}$ and $\mathcal{H}_\lambda^{(0,1)}$ differ only in their definition of x'. Specifically, they use different values for x_B, masked by an appropriate one-time pad generated from $F_B(0,\cdot)$. But both in $\mathcal{H}^{(0,0)}$ and in $\mathcal{H}_\lambda^{(0,1)}$, F_B is punctured on $\{(0,a) : a \in \{0,1\}^\lambda\}$. Thus, by reduction to the punctured PRF security of F_B (with an adversary that makes n PRF queries), the distribution ensembles $\{\mathcal{H}_\lambda^{(0,0)}\}$ and $\{\mathcal{H}_\lambda^{(0,1)}\}$ are $(n \cdot \delta)$-indistinguishable. □

Claim 3.14.4. *The distribution ensembles $\{\mathcal{H}_\lambda^{(4)}\}$ and $\{\mathcal{H}_\lambda^{(5)}\}$ are computationally $(n \cdot \delta)$-indistinguishable, where $n(\lambda)$ is the (common) length of $x_\lambda^{(0)}$ and $x_\lambda^{(1)}$.*

Proof. Completely analogous to Claim 3.14.3. □

Claim 3.14.5. *The distribution ensembles $\{\mathcal{H}_\lambda^{(2)}\}$ and $\{\mathcal{H}_\lambda^{(3)}\}$ are computationally δ-indistinguishable.*

Proof. This follows directly from the security of $\mathcal{G}'$ – namely, the fact that it satisfies same memory δ-indistinguishability. □

It remains to show the following two claims.

Claim 3.14.6. *The distribution ensembles $\{\mathcal{H}_\lambda^{(1)}\}$ and $\{\mathcal{H}_\lambda^{(2)}\}$ are computationally $(T \cdot \delta)$-indistinguishable.*

Claim 3.14.7. *The distribution ensembles $\{\mathcal{H}_\lambda^{(3)}\}$ and $\{\mathcal{H}_\lambda^{(4)}\}$ are computationally $(T \cdot \delta)$-indistinguishable.*

We prove Claim 3.14.6, and the proof of Claim 3.14.7 is analogous.

Proof of Claim 3.14.6. For each λ, we show a sequence of $T_\lambda + 1$ indistinguishable hybrid distributions $\mathcal{H}_\lambda^{(1,0)}, \ldots, \mathcal{H}_\lambda^{(1,T_\lambda)}$ such that:

- $\mathcal{H}_\lambda^{(1,0)} = \mathcal{H}_\lambda^{(1)}$.

[2]While not guaranteed by Definition 2.9, the puncturable PRF of [GGM86] is *succinctly* puncturable on such "interval" sets. That is, the size of the punctured key is poly(λ) and not 2^λ. It is also possible to prove Claim 3.14.3 with a standard puncturable PRF (puncturable only at small sets of points) with a slightly more complicated argument.

- $\mathcal{H}_\lambda^{(1,T_\lambda)} = \mathcal{H}_\lambda^{(2)}$.
- For each $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, it holds that $\{\mathcal{H}_\lambda^{(1,i_\lambda)}\}$ and $\{\mathcal{H}_\lambda^{(1,i_\lambda+1)}\}$ are computationally δ-indistinguishable.

Hybrid $\mathcal{H}_\lambda^{(1,i)}$: For $0 \leq i < T_\lambda$, hybrid $\mathcal{H}_\lambda^{(1,i)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ and $F_A, F_B \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows.

   M' is given by the CPU circuit $C_2'[\lambda, A, i, C_0, C_1, F_A, F_B]$ defined in Algorithm 5, where C_0 and C_1 are the CPU circuits defining $M_\lambda^{(0)}$ and $M_\lambda^{(1)}$.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}(\mathbf{K}', x')$, where x' is defined so that the a^{th} word of x' is

$$x'_a \stackrel{\mathsf{def}}{=} \Big(0, a, F_A(0,a) \oplus (x_\lambda^{(0)})_a, F_B(0,a) \oplus (x_\lambda^{(1)})_a\Big).$$

Claim 3.14.8. *For every ensemble $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(1,i_\lambda)}\}$ and $\{\mathcal{H}_\lambda^{(1,i_\lambda+1)}\}$ are computationally δ-indistinguishable.*

Proof. For each λ, we define two other hybrid distributions $\mathcal{H}_\lambda^{(1,i_\lambda,0)}$ and $\mathcal{H}_\lambda^{(1,i_\lambda,1)}$.

For each $b \in \{0,1\}$, let the computation transcript of $M_\lambda^{(b)}$ on $x_\lambda^{(b)}$ be denoted by

$$(q_0^{(b)}, (a_0^{(b)}, w_0^{(b)})), \ldots, (q_{T_\lambda}^{(b)}, (a_{T_\lambda}^{(b)}, w_{T_\lambda}^{(b)})).$$

Recall that by assumption, $a_t^{(0)} = a_t^{(1)}$ for every t. We will denote this value by a_t.

Hybrid $\mathcal{H}_\lambda^{(1,i_\lambda,0)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ and $F_A, F_B \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows.

   M' is given by the CPU circuit $C_3'[\lambda, A, i_\lambda, c, C_0, C_1, F_A, F_B\{(i_\lambda, a_{i_\lambda})\}]$ defined in Algorithm 6, where C_0 and C_1 are the CPU circuits defining $M_\lambda^{(0)}$ and $M_\lambda^{(1)}$, and $c \stackrel{\mathsf{def}}{=} F_B(i_\lambda, a_{i_\lambda}) \oplus w_{i_\lambda}^{(0)}$.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}(\mathbf{K}', x')$, where x' is defined so that the a^{th} word of x' is

$$x'_a \stackrel{\mathsf{def}}{=} \Big(0, a, F_A(0,a) \oplus (x_\lambda^{(0)})_a, F_B(0,a) \oplus (x_\lambda^{(1)})_a\Big).$$

By the fact that $\mathcal{G}'$ satisfies same address δ-indistinguishability, the distribution ensembles $\{\mathcal{H}_\lambda^{(1,i_\lambda)}\}$ and $\{\mathcal{H}_\lambda^{(1,i_\lambda,0)}\}$ are computationally δ-indistinguishable.

Hybrid $\mathcal{H}_\lambda^{(1,i_\lambda,1)}$ is defined nearly identically to $\mathcal{H}_\lambda^{(1,i_\lambda,0)}$, the only difference being that the parameter c is set to $(i_\lambda, F_B^{(B)}(i_\lambda, a_{i_\lambda}) \oplus w_{i_\lambda}^{(1)})$.

Input: State $(t, q_{\mathsf{in}}^{(A)}, q_{\mathsf{in}}^{(B)})$, read memory word $(t_{\mathsf{read}}, a_{\mathsf{read}}, w_{\mathsf{read}}^{(A)}, w_{\mathsf{read}}^{(B)})$

$\left(q_{\mathsf{out}}^{(\gamma)}, (a_{\mathsf{write}}^{(\gamma)}, w_{\mathsf{write}}^{(\gamma)})\right) := C_\gamma(q_{\mathsf{in}}^{(\gamma)}, F_\gamma(t_{\mathsf{read}}, a_{\mathsf{read}}) \oplus w_{\mathsf{read}}^{(\gamma)})$;

if $t < i$ **then**

  // Also execute track $\bar{\gamma}$

  $\left(q_{\mathsf{out}}^{(\bar{\gamma})}, (a_{\mathsf{write}}^{(\bar{\gamma})}, w_{\mathsf{write}}^{(\bar{\gamma})})\right) := C_{\bar{\gamma}}(q_{\mathsf{in}}^{(\bar{\gamma})}, F_{\bar{\gamma}}(t_{\mathsf{read}}, a_{\mathsf{read}}) \oplus w_{\mathsf{read}}^{(\bar{\gamma})})$;

  $\mathsf{op}' := (a_{\mathsf{write}}^{(\gamma)}, (t+1, a_{\mathsf{write}}^{(\gamma)}, F_A(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(A)}, F_B(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(B)}))$;

else if $t = i$ **then**

  // Write c to track $\bar{\gamma}$.

  $c_\gamma := F_\gamma(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(\gamma)}$, $c_{\bar{\gamma}} := c$;

  $\mathsf{op}' := (a_{\mathsf{write}}^{(\gamma)}, (t+1, a_{\mathsf{write}}^{(\gamma)}, c_A, c_B))$;

else

  // Duplicate write to track γ to track $\bar{\gamma}$

  $\mathsf{op}' := (a_{\mathsf{write}}^{(\gamma)}, (t+1, a_{\mathsf{write}}^{(\gamma)}, F_A(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(\gamma)}, F_B(t+1, a_{\mathsf{write}}^{(\gamma)}) \oplus w_{\mathsf{write}}^{(\gamma)}))$;

if $q_{\mathsf{out}}^{(\gamma)} = q_{\mathsf{acc}}$ **then return** $(q'_{\mathsf{acc}}, \mathsf{op}')$;

if $q_{\mathsf{out}}^{(\gamma)} = q_{\mathsf{rej}}$ **then return** $(q'_{\mathsf{rej}}, \mathsf{op}')$;

$q'_{\mathsf{out}} := (t+1, q_{\mathsf{out}}^{(\gamma)}, q_{\mathsf{out}}^{(\gamma)})$;

return $(q'_{\mathsf{out}}, \mathsf{op}')$;

Algorithm 6: CPU circuit $C'_1[\lambda, \gamma, i, c, C_A, C_B, F_A, F_B]$. q_{acc} and q_{rej} denote the accepting and rejecting states, respectively, of C_γ. $\bar{\gamma}$ denotes B if γ is A, and vice versa.

By the pseudorandomness of $F^{(B)}$ at the point $(i_\lambda, a_{i_\lambda})$, the distribution ensembles $\{\mathcal{H}_\lambda^{(1,i_\lambda,0)}\}$ and $\{\mathcal{H}^{(1,i_\lambda,1)}\}$ are computationally δ-indistinguishable.

Finally, $\{\mathcal{H}_\lambda^{(1,i_\lambda,1)}\}$ and $\{\mathcal{H}_\lambda^{(1,i_\lambda+1)}\}$ are computationally δ-indistinguishable by the same address δ-indistinguishability of $\mathcal{G}'$. □

Claim 3.14.6 follows directly from Claim 3.14.8. □

Theorem 3.14 follows directly from Claims 3.14.1 to 3.14.7. □

3.5 Full Security

This section constructs a garbling scheme for RAM machines with full security, as defined in Section 3.1.2, from any garbling scheme that satisfies same address indistinguishability. As sketched and motivated in Section 3.1, this is done by combining the fixed address garbling scheme with an oblivious RAM (ORAM) scheme. While this composition may generically fail to achieve full security, we show that it works if the ORAM satisfies a stronger than usual security requirement, and if the randomness is generated via a puncturable PRF. We also show that existing ORAMs (unconditionally) satisfy the stronger security requirement.

Localized Randomness

Our new notion of localized randomness constrains how addr' depends *as a deterministic function* on the randomness used in generating $\mathbf{x}'$ and in executing M' on $\mathbf{x}'$. Let S denote the space usage of the RAM machine M, and let T denote the running time of M on x.

Formally, we view addr' as a deterministic function of M, x, S, λ, and $\vec{r}$, where $\vec{r}$ is a concatenation of the randomness used in generating $\mathbf{x}' \leftarrow \mathsf{OMem}(1^\lambda, x)$ and the randomness used in executing $M' = \mathsf{OProg}(1^\lambda, M)$ on $\mathbf{x}'$. Localized randomness requires that for all values of S and λ, there is some $\eta > 0$ and some deterministic algorithm $\mathsf{Sim}_{S,\lambda}$ such that for every space-S RAM machine M and every input x, there exist pairwise disjoint sets $R_1, \ldots, R_T \subseteq \mathbb{N}$ (where $T = \mathsf{Time}(M, x)$) such that for each $i \in [T]$:

- R_i has size at most $\mathrm{poly}(\log S, \lambda)$.
- It holds with all but negligible (in λ) probability over a uniformly random choice of $\vec{r}$ that the tuple consisting of the $\left(\eta \cdot (i-1)\right)^{th}$ through $(\eta \cdot i - 1)^{th}$ addresses in addr' is equal to $\mathsf{Sim}_{S,\lambda}(\vec{r}_{R_i})$.

We now show that the Chung-Pass ORAM [CP13] satisfies localized randomness.

3.5.1 Locally Random ORAM Construction

Memory Layout For a RAM machine M and input $x \in \{0,1\}^n$, the Chung-Pass ORAM begins by partitioning x into blocks of size α for some constant $\alpha > 1$. These blocks are labeled with their addresses, randomly shuffled, and a position map Pos_1 is generated, with $\mathsf{Pos}_1(i)$ storing the shuffled position of block i. For $i = 1, \ldots, \log(n)$, Pos_i is also randomly shuffled in blocks of size α, with Pos_{i+1} storing this shuffling.

For each i, there is a balanced binary tree $\mathcal{T}_i$ of buckets in memory, with each bucket sized to hold $\mathrm{polylog}(\lambda)$ blocks. When $\mathsf{Pos}_i(\beta) = \gamma$, it means that the block labeled β is present in this tree in one of the buckets from the root to leaf γ.

Memory Accesses To obliviously access an address a in this data structure, one first looks up $\mathsf{Pos}_{\log n}(a)$ from private registers or a brute-force ORAM. Having computed $\mathsf{Pos}_i(a)$, one now looks up $\mathsf{Pos}_{i-1}(a)$ by reading each bucket on the path to the $\mathsf{Pos}_i(a)^{th}$ leaf in $\mathcal{T}_i$, searching for the block which is labeled a (or more precisely $\lfloor a/\alpha^i \rfloor$) and retrieving its contents. This block is not written back to the bucket, but is instead assigned a new position pos^*. Labeled as such, it is inserted into the root bucket in $\mathcal{T}_i$. Next, and this step is crucial to prevent buckets from overflowing, the path in $\mathcal{T}_i$ from root to a random leaf is traversed, with each block moved to the leafmost admissible bucket along this path.

Localized Randomness

In order to establish that the localized randomness property is satisfied, we must define an algorithm Sim, and given a RAM machine M and input x with $\mathsf{Time}(M, x) =$

T, we must define subsets $R_1, \ldots, R_T$ of the ORAM randomness such that the addresses accessed on the i^{th} access when using randomness $\vec{r}$ are given by $\mathsf{Sim}(\vec{r}_{R_i})$.

Sim takes a random string r and interprets it as labeling two leaf nodes in each tree $\mathcal{T}_{\log n}, \ldots, \mathcal{T}_1$. For each such leaf node ℓ, Sim outputs the addresses of each bucket on the path from the root to ℓ.

Suppose that M on x accesses addresses $a_1, \ldots, a_T$. The set R_i is defined by considering, for each $j \in \{1, \ldots, \log n\}$, $k_{i,j} = \max\{k : k < i \wedge a_k/\alpha^j = a_i/\alpha^j\}$. Then R_i is defined as the concatenation of, for each j, the randomness used in choosing the new positions pos^* in $\mathcal{T}_j$ on the $(k_{i,j})^{th}$ underlying access, as well as the randomness in choosing the random path along which blocks should be flushed towards the root.

3.5.2 Garbling Scheme Construction

Our garbling scheme is very simple; essentially, we just compose the fixed address garbler on top of an ORAM scheme with localized randomness. That is, to garble a machine M, we first transform it via the ORAM, and then apply the fixed address garbler to that transformed machine.

Construction 3.15. *Let* $(\mathsf{KeyGen}', \mathsf{GbPrg}', \mathsf{GbMem}')$ *be a garbling scheme that satisfies same address indistinguishability, and let* $(\mathsf{OMem}, \mathsf{OProg})$ *be an ORAM scheme with localized randomness. We define a garbling scheme* $\mathcal{G} = (\mathsf{KeyGen}, \mathsf{GbPrg}, \mathsf{GbMem})$ *as follows:*

- $\mathsf{KeyGen}(1^\lambda)$ *samples*

$$\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$$
$$F \leftarrow \mathcal{F}_\lambda$$

  and outputs $\mathbf{K} \stackrel{\mathsf{def}}{=} (1^\lambda, \mathbf{K}', F)$.

- $\mathsf{GbPrg}(\mathbf{K}, M)$ *outputs* $\mathsf{GbPrg}'(\mathbf{K}', M')$, *where* $M' \stackrel{\mathsf{def}}{=} \mathsf{OProg}(1^\lambda, M)^F$.

- $\mathsf{GbMem}(\mathbf{K}, x)$ *samples* $\mathbf{x}' \leftarrow \mathsf{OMem}(1^\lambda, x)$ *and outputs* $\mathsf{GbMem}'(\mathbf{K}', \mathbf{x}')$.

3.5.3 Security Proof

Theorem 3.16. *Suppose that* $\mathcal{G}' = (\mathsf{KeyGen}', \mathsf{GbPrg}', \mathsf{GbMem}')$ *satisfies same memory* δ*-indistinguishability for polynomial size, time-*T *RAM computations, and that* $\{\mathcal{F}_\lambda\}$ *is a* δ*-secure family of puncturable PRFs. Then* $\mathcal{G}$ *as in Construction 3.15 is a* $(T \cdot \delta)$*-secure garbling scheme for polynomial size, time-*T *RAM computations.*

Proof Overview We proceed through a sequence of hybrid distributions, where each hybrid distribution uses a different definition of M'. Specifically, in the i^{th} hybrid, M' is defined to:

1. Act as $\mathsf{OProg}(1^\lambda, M)^F$ for $T_\lambda - i$ steps,

2. Perform i simulated "dummy" accesses,

3. Output $y = M(x)$ as a hard-coded constant.

To prove that these hybrids are indistinguishable, we make crucial use of the ORAM's localized randomness. Indeed, locality allows us to isolate the randomness that determines the particular addresses we are trying to change. In conjunction with our fixed address garbler, which lets us ignore low-level RAM machine details, we then use the punctured programming method to change the addresses accessed.

Proof. Let $\{(M_\lambda, x_\lambda)\}$ be a polynomial size, time-$T(\cdot)$ ensemble of RAM computations.

The Real World Distribution $\{\mathcal{RW}_\lambda\}$: We want to show the simulability of the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ which is obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and $F \leftarrow \mathcal{F}_\lambda$.

2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', \mathsf{OProg}(1^\lambda, M_\lambda))$

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x_\lambda))$.

Let $T = T_\lambda$ denote the running time of M_λ on x_λ, let $S = S_\lambda$ denote the space usage of M_λ, and let $n = n_\lambda$ denote the length of x_λ.

The Simulating Distribution $\{\mathcal{S}_\lambda\}$: $\mathcal{S}_\lambda$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and a puncturable PRF $G \leftarrow \mathcal{F}_\lambda$.

2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine with CPU circuit representation
$$(C_2'[T, G, M_\lambda(x_\lambda)], (0,0), (T,1), (T,0))$$
where C_2' is described in Algorithm 7.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, 0^n))$.

Input: State (t, q_{in}), symbol σ
Let (i', j') be integers such that $t = \eta i' + j'$ for $0 \le j' < \eta$;
if $i' < T$ **then**
  $\mathsf{op} := (\mathsf{Sim}(G(i'))_{j'}, 0)$;
  return $((t+1, q_{in}), \mathsf{op})$
else return y;

Algorithm 7: Simulating CPU circuit $C_2'[T, G, y]$.

One can easily check that $\mathcal{S}_\lambda$ can be sampled given only $M(x)$, T, S, and n.

Hybrid $\mathcal{H}_\lambda^{(i)}$: For $0 \leq i \leq T_\lambda$, Hybrid $\mathcal{H}_\lambda^{(i)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and puncturable PRFs $F, G \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows.

   If the CPU circuit representation of M_λ is $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, then the CPU circuit representation of M' is

   $$(C_1'[\lambda, i, T_\lambda, C, F, G, y], q_0', q_{\mathsf{acc}}', q_{\mathsf{rej}}'),$$

   where C_1' is defined in Algorithm 8 and

   - $q_0' \stackrel{\mathsf{def}}{=} (0, q_{\mathsf{acc}})$
   - $q_{\mathsf{acc}}' \stackrel{\mathsf{def}}{=} (T_\lambda, q_{\mathsf{acc}})$
   - $q_{\mathsf{rej}}' \stackrel{\mathsf{def}}{=} (T_\lambda, q_{\mathsf{rej}})$
   - y is q_{acc}' if $M_\lambda(x_\lambda) = 1$, and q_{rej}' otherwise.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x))$.

Input: State (t, q_{in}), read memory word w_{read}
Let (i', j') be integers such that $t = \eta i' + j'$ for $0 \leq j' < \eta$;
if $i' < T_\lambda - i$ **then**
  $(q_{\mathsf{out}}, \mathsf{op}) := C^F(q_{in}, w_{\mathsf{read}})$;
  return $((t+1, q_{\mathsf{out}}), \mathsf{op})$;
else if $T - i \leq i' < T$ **then**
  $\mathsf{op} := (\mathsf{Sim}(G(i'))_{j'}, 0)$;
  return $((t+1, q_{in}), \mathsf{op})$
else return y;

Algorithm 8: CPU circuit $C_1'[\lambda, i, T, C, F, G, y]$.

It remains to prove the following three claims.

Claim 3.16.1. *The distribution ensembles $\{\mathcal{RW}_\lambda^{(0)}\}$ and $\{\mathcal{H}_\lambda^{(0)}\}$ are δ-indistinguishable.*

Claim 3.16.2. *For all $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(i_\lambda)}\}$ and $\{\mathcal{H}_\lambda^{(i+1)}\}$ are δ-indistinguishable.*

Claim 3.16.3. $\{\mathcal{H}_\lambda^{(T_\lambda)}\} \approx \{\mathcal{S}_\lambda\}$

Claims 3.16.1 and 3.16.3 follow immediately from the same address indistinguishability of $\mathcal{G}'$.

Proof. (of Claim 3.16.2)
We give a sequence of computationally δ-indistinguishable distribution ensembles

$$\{\mathcal{H}_\lambda^{(i)}\} \approx \{\mathcal{H}_\lambda^{(i,1)}\} \approx \cdots \approx \{\mathcal{H}_\lambda^{(i,4)}\} \approx \{\mathcal{H}_\lambda^{(i+1)}\}.$$

Hybrid $\mathcal{H}_\lambda^{(i,1)}$: Hybrid $\mathcal{H}_\lambda^{(i,1)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and puncturable PRFs $F, G \leftarrow \mathcal{F}_\lambda$.

2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine defined as follows.

   The CPU circuit of M' is given in Algorithm 9, with the following parameters: hard-coded values p, $\iota_1, \ldots, \iota_p$ and $b_1, \ldots, b_p$ defined so that $S_i = \{\iota_1, \ldots, \iota_p\}$, and $b_j = F(\iota_j)$. The hard-coded values $\vec{a} = (a_0, \ldots, a_{\eta-1})$ are defined as $\vec{a} = \mathsf{Sim}(b_1, \ldots, b_p)$.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x_\lambda))$.

Data: Punctured PRF $F' = F\{S_i\}$, puncturable PRF G, underlying transition function C', $y = M(s)$, the set $S_i = \{\iota_1, \ldots, \iota_p\}$, bits $b_1, \ldots, b_p$, addresses $a_0, \ldots, a_{\eta-1}$, the running time $t^* = \mathsf{Time}(M, s)$
Input: State (t, q_{in}), symbol σ

Let (i', j') be integers such that $t = \eta i' + j'$ for $0 \leq j' < \eta$;
Define $\bar{F}'$ such that $\bar{F}'(x) = \begin{cases} b_j & \text{if } x = \iota_j \\ F'(x) & \text{otherwise} \end{cases}$;
if $i' < i$ **then**
  $(q_{\mathsf{out}}, \mathsf{op}) := C'^{\bar{F}'}(q_{in}, \sigma)$;
  return $((t+1, q_{\mathsf{out}}), \mathsf{op})$;
else if $i' = i$ **then return** $((t+1, q_{in}), (a_j, \perp))$;
else if $i < i' < t^*$ **then return** $((t+1, q_{in}), (\mathsf{Sim}(G(i'))_{j'}, \perp))$;
else return y;

Algorithm 9: Hybrid transition function $C_{i,1}$.

Hybrid $\mathcal{H}_\lambda^{(i,2)}$: Hybrid $\mathcal{H}_\lambda^{(i,2)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and puncturable PRFs $F, G \leftarrow \mathcal{F}_\lambda$.

2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M_{i,2})$, where the RAM machine $M_{i,2}$'s transition function is given in Algorithm 9, with the hard-coded values p, $\iota_1, \ldots, \iota_p$ and $b_1, \ldots, b_p$ defined so that $S_i = \{\iota_1, \ldots, \iota_p\}$, and each b_j is drawn from $\{0,1\}$ independently and uniformly at random. The hard-coded values $\vec{a} = (a_0, \ldots, a_{\eta-1})$ are defined as $\vec{a} = \mathsf{Sim}(b_1, \ldots, b_p)$.

3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x_\lambda))$.

Hybrid $\mathcal{H}_\lambda^{(i,3)}$: Hybrid $\mathcal{H}_\lambda^{(i,3)}$ is defined as the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$ and puncturable PRFs $F, G \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M_{i,3})$, where the RAM machine $M_{i,3}$'s transition function is given in Algorithm 10, with the hard-coded values $\vec{a} = (a_0, \ldots, a_{\eta-1})$ sampled as $\vec{a} = \mathsf{Sim}(b_1, \ldots, b_p)$ for uniformly random $(b_1, \ldots, b_p) \in \{0,1\}^p$.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x_\lambda))$.

Data: Puncturable PRF F, punctured PRF $G' = G\{i\}$, underlying transition function C', $y = M(s)$, addresses $a_0, \ldots, a_{\eta-1}$, the running time $t^* = \mathsf{Time}(M, s)$
Input: State (t, q_{in}), symbol σ

Let (i', j') be integers such that $t = \eta i' + j'$ for $0 \leq j' < \eta$;
if $i' < i$ **then**
  $(q_{\mathsf{out}}, \mathsf{op}) := C'^F(q_{in}, \sigma)$;
  return $((t+1, q_{\mathsf{out}}), \mathsf{op})$;
else if $i' = i$ **then return** $((t+1, q_{in}), (a_j, \perp))$;
else if $i < i' < t^*$ **then return** $((t+1, q_{in}), (\mathsf{Sim}(G(i'))_{j'}, \perp))$;
else return y;

Algorithm 10: Hybrid transition function $C_{i,3}$.

Hybrid $\mathcal{H}_\lambda^{(i,4)}$: Hybrid $\mathcal{H}_\lambda^{(i,4)}$ is defined to be the distribution on $(\widetilde{\mathbf{M}}, \tilde{\mathbf{x}})$ obtained by sampling:

1. $\mathbf{K}' \leftarrow \mathsf{KeyGen}'(1^\lambda)$, and puncturable PRFs $F, G \leftarrow \mathcal{F}_\lambda$.
2. $\widetilde{\mathbf{M}} \leftarrow \mathsf{GbPrg}'(\mathbf{K}', M')$, where M' is a new RAM machine whose CPU circuit is given in Algorithm 10, with the hard-coded values $\vec{a} = (a_0, \ldots, a_{\eta-1})$ sampled as $\vec{a} = \mathsf{Sim}(G(i))$.
3. $\tilde{\mathbf{x}} \leftarrow \mathsf{GbMem}'(\mathbf{K}', \mathsf{OMem}(1^\lambda, x_\lambda))$.

Claim 3.16.4. *For any $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(i_\lambda)}\}$ and $\{\mathcal{H}_\lambda^{(i_\lambda,1)}\}$ are computationally δ-indistinguishable.*

Proof. This follows from the same address indistinguishability of $\mathcal{G}'$. □

Claim 3.16.5. *For all $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(i_\lambda,1)}\}$ and $\{\mathcal{H}_\lambda^{(i_\lambda,2)}\}$ are computationally δ-indistinguishable.*

Proof. This follows from the pseudorandomness of F at the (selectively) punctured points $\{\iota_1, \ldots, \iota_p\}$ in a straight-forward way. □

Claim 3.16.6. *For all $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(i_\lambda,2)}\}$ and $\{\mathcal{H}_\lambda^{(i_\lambda,3)}\}$ are computationally δ-indistinguishable.*

Proof. This follows from the same address indistinguishability of $\mathcal{G}'$. Indeed, $\iota_1, \ldots, \iota_p$ are defined so that changing $F(\iota_j)$ can only possibly change the addresses accessed at time $T_\lambda - i_\lambda$. But at time $T_\lambda - i_\lambda$, the accessed addresses $a_0, \ldots, a_{\eta-1}$ are hard-coded in M', both in $\mathcal{H}_\lambda^{(i_\lambda,2)}$ and in $\mathcal{H}_\lambda^{(i_\lambda,3)}$. □

Claim 3.16.7. *For all $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}_\lambda^{(i_\lambda,3)}\}$ and $\{\mathcal{H}_\lambda^{(i_\lambda,4)}\}$ are computationally δ-indistinguishable.*

Proof. This follows from the pseudorandomness of G at the selectively punctured point $T_\lambda - i$ in a straight-forward way. □

Claim 3.16.8. *For all $\{i_\lambda\}$ with $0 \leq i_\lambda < T_\lambda$, the distribution ensembles $\{\mathcal{H}^{(i_\lambda,4)}\}$ and $\{\mathcal{H}_\lambda^{(i_\lambda+1)}\}$ are computationally δ-indistinguishable.*

Proof. This follows from the same address indistinguishability of $\mathcal{G}'$. □

This concludes the proof of Claim 3.16.2. □

This concludes the proof of Theorem 3.16. □

3.6 Randomized Encodings

Randomized encodings were first introduced by Ishai and Kushilevitz [IK00] for the purpose of improving MPC efficiency, and they have since grown to play an important role in efficient cryptography.

Definition 3.17. *A* computationally δ-private randomized encoding scheme *for a function $f : \{0,1\}^* \to \{0,1\}^*$ is a pair of p.p.t. algorithms $\mathcal{RE} = (\mathsf{Enc}, \mathsf{Dec})$ satisfying the following properties.*

Correctness. *For all $x \in \{0,1\}^*$ and all λ, when sampling $\tilde{x} \leftarrow \mathsf{Enc}(1^\lambda, x)$, it holds with probability 1 that $\mathsf{Dec}(\tilde{x}) = f(x)$.*

Security. *There is a p.p.t. algorithm* Sim *such that for all polynomial-sized ensembles of strings $\{x^{(\lambda)} \in \{0,1\}^*\}_\lambda$, the distribution ensembles $\left\{\mathsf{Enc}(1^\lambda, x^{(\lambda)})\right\}_\lambda$ and $\left\{\mathsf{Sim}(1^\lambda, f(x^{(\lambda)}))\right\}_\lambda$ are computationally δ-indistinguishable.*

Encoder Efficiency. *The algorithm* $\mathsf{Enc}(1^\lambda, x)$ *runs in time* $\tilde{O}(n) \cdot \mathrm{poly}(\lambda)$ *where* n *denotes the length of* x*. See additional discussion below.*

The requirement on encoder efficiency rules out the following trivial perfectly secure randomized encoding scheme for arbitrary (polynomial-time computable) functions. For a given function f, Enc is defined so that $\mathsf{Enc}(1^\lambda, x)$ is $f(x)$, and Dec is defined to be the identity function. We therefore say that a scheme is *non-trivial* if the complexity of Enc is significantly less than what is required to *compute* f. In the above definition, we chose to instead require nearly linear time complexity for Enc, for two reasons. First, our schemes achieve such efficiency. Second, we find it aesthetically preferable to quantify over *all* polynomial-time computable functions in our theorem statements, and leave it implicit that a construction is trivial for linear-time computable functions.

We also could have specified a weaker measure of complexity relative to which Enc is required to be more efficient than f. For example, Yao garbled circuits [Yao86] yield, for any polynomial-time computable function f, a randomized encoding scheme in which the encoder's *parallel* running time is small, but the encoder's total work is as large as the time comlexity of f.

Theorem 3.18. *If injective one-way functions and indistinguishability obfuscation for circuits exist, then for every polynomial-time computable function* f*, there is a randomized encoding scheme for* f*.*

Proof. Follows directly from Theorem 3.6. □

3.7 Verifiable Computation from Succinct Randomized Encodings

Suppose that for every polynomial-time computable function f, there is a randomized encoding scheme for f. For example, Theorem 3.18 shows that this is the case if injective one-way functions and indistinguishability obfuscation for circuits exist. Then there is a verifiable computation protocol due to [AIK04] for any language $\mathcal{L} \in \mathsf{P}$ wherein:

1. The verifier's input is a string $x \in \{0,1\}^*$, while the prover has no input.

2. There is a single round (two messages) of interaction: the verifier sends a first "challenge" message to the prover, and the prover responds with a "proof" message.

We state the construction for completeness, but omit the proof.

Construction 3.19. *Given a language* $\mathcal{L} \in \mathsf{P}$*, define a two-party protocol between a prover* P *and verifier* V *as follows.*

Let $(\mathsf{Enc}, \mathsf{Dec})$ *be any randomized encoding scheme for* $f_{\mathcal{L}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$*, defined as*

$$f_{\mathcal{L}}(x, r) \stackrel{\mathsf{def}}{=} \begin{cases} r & \text{if } x \in \mathcal{L} \\ 0 & \text{otherwise.} \end{cases}$$

On input $(1^\lambda, x)$ *with* $x \in \{0,1\}^n$*,* V *samples* $r \leftarrow \{0,1\}^\lambda$ *and* $\tilde{x} \leftarrow \mathsf{Enc}(1^\lambda, (x, r))$*, and sends* $\tilde{x}$ *to* P*. Upon receiving* $\tilde{x}$*,* P *responds with* $\tilde{r} = \mathsf{Dec}(\tilde{x})$*.* V *then accepts iff* $\tilde{r} = r$*.*

Remark 3.20 (Efficiency). In Construction 3.19, the verifier's complexity is equal to that of Enc. With the randomized encoding scheme of Theorem 3.18, this is $n \cdot \mathrm{poly}(\lambda)$ time and $O(\lambda)$ space.

The prover's complexity is equal to that of Dec. With the randomized encoding scheme of Theorem 3.18, this is $T(n) \cdot \mathrm{poly}(\lambda)$ time and $S(n) \cdot \mathrm{poly}(\lambda)$ space.

Chapter 4

Verifiable Computation

This chapter is based on the STOC 2017 paper "Non-Interactive RAM and Batch NP Delegation from any PIR" by Brakerski, Holmgren, and Kalai [BHK17].

Efficient verification of computation is one of the most fundamental and studied tasks in computer science, and in particular it lies at the heart of the P vs. NP question. In the most basic setting, a prover P proves to a verifier V that a certain instance x is in some language L. In a *delegation protocol* the resources for verification should be much lower than those required to compute whether x is indeed in L. There are several variants of this question, each of which has been the subject of intense research.

One common framing of this question, as proposed e.g. in [Kil92, Mic94, GKR08a, GGP10], asks whether it is possible to "succinctly" verify $\mathsf{TIME}(T)$ computations, i.e. with verification communication and computation $\mathrm{polylog}(T)$.[1]

Assuming $\mathsf{EXP} \not\subseteq \mathsf{PSPACE}$ (which is widely believed), this is generally unattainable with soundness against computationally unbounded provers. Thus, similar to most previous work in the literature, we focus on achieving soundness only against computationally bounded provers (such proof systems are also known in the literature as *arguments*). Honest provers are also required to be efficient, in keeping with the philosophy that security should hold against adversaries who are at least as powerful as honest parties.

An additional variant is delegation of *non-deterministic* languages, say in $\mathsf{NTIME}(T)$. Here an even stronger implausibility of unconditional succinct verification applies, but succinct four-message *arguments* were constructed by Kilian, using PCPs and collision-resistant hash functions [Kil92]. If the protocol is additionally required to be non-interactive, then Gentry and Wichs showed that (under plausible complexity assumptions) one cannot even prove *computational* soundness with black-box reductions to falsifiable assumptions [GW11]. In the face of these negative results, Reingold, Rothblum and Rothblum [RRR16] recently asked whether one can obtain non-trivial verifier efficiency by amortizing the verification of many NP instances at reduced per-instance cost, and give such an interactive proof for the special class of unambiguous interactive proofs.

[1] Some subtleties arise when formalizing this problem, e.g. the time it takes to read the input x. These will be discussed below.

Aside from its foundational value, the question of efficient delegation carries significance to applications as well. In many cases, computation today is becoming asymmetric, with smart devices able only to perform lightweight local computation, and outsourcing large computational tasks to be performed off-site (e.g. by a cloud server). The ability to verify that the computation is carried out correctly without investing significant computational resources is obviously useful in this situation. This serves as additional motivation to explore which classes of computation admit succinct and efficient delegation schemes, and what amount of communication and interaction are required in order to accomplish this task.

4.1 Our Results

We construct the first one-message delegation scheme (in a public-key model), under standard *generic* hardness assumptions (the existence of succinct computational private information retrieval) for polynomial-time computation. In addition, our scheme obtains a stronger notion of soundness that allows any party to adaptively choose a computation as a function of the verifier's public key, and convince the verifier of that computation's result. Prior to this work, the state of the art (barring random oracles or non-standard cryptographic assumptions) was a two-message protocol relying on super-polynomial hardness [KRR14, KP16]. A recent concurrent work by Dwork, Naor and Rothblum [DNR16] achieves two-message protocols under polynomial assumptions, but only for log-space uniform NC circuits.

We also construct the first two-message batch argument system for NP based on the same assumptions, and achieve a communication complexity proportional to the length of a single witness. We show that in this setting non-adaptivity is inherent, and that our attained communication complexity is optimal for schemes whose soundness has a black-box reduction to a falsifiable computational assumption. We are not aware of any prior work achieving such a two-message system without random oracles or non-standard assumptions.

As a main building block in our constructions, we define and construct a "computationally no-signaling" (CNS) probabilistically checkable proof (PCP), which strengthens the definition of a statistically no-signaling PCP [KRR14]. We believe that this tool may be of independent interest.

4.1.1 Non-interactive Delegation

Let A be a party with a public/private key pair, generated with respect to some security parameter λ. Let M denote a Turing machine, let x denote an input, and let T denote the running time of M on x. Assume that λ is chosen so that $|x| \leq \text{poly}(\lambda)$ and assume M is a poly-time machine, so also $T \leq \text{poly}(\lambda)$. At its simplest, our scheme is the first, assuming only standard cryptographic assumptions, to enable the following state of affairs.

- One can efficiently (in time $\text{poly}(T, \lambda)$) compute a proof string to convince A of any true statement "$M(x) = y$".

- No polynomially bounded adversary can, given A's public key, come up with a false statement (that $M'(x') = y'$) and a corresponding "proof" which A will accept.

- A can verify the proof in time $\text{poly}(\lambda, |x|, |y|)$ (independent of the runtime of M).

Our scheme has several other desirable properties, which we now outline.

1. **RAM Efficiency and Persistent Data.** Our adaptive delegation scheme, following [KP16], works in the following model: Given the public key of the verifier, the prover can prove the correctness of the computation of any *RAM* machine M on any database D. Both the prover and verifier can preprocess D independently of M in an offline phase. When they are later given M with running time T in an online phase, the prover[2] runs in time $\text{poly}(T)$ and the verifier runs in time $\text{poly}(\lambda)$ for some fixed polynomial poly. In fact, the verifier needs to store very little state from its preprocessing of D - it just needs to store a λ-bit digest, which we denote by d.[3]

   We note that a RAM computation M^D may alter the database D, resulting in a new database D_{new}. In many cases it is desirable to delegate computations on D_{new} without repeating the pre-processing. Our scheme supports this, and in fact allows one to continue such incremental computation indefinitely.

2. **Soundness With Dishonest Digests.** Our notion of soundness is an adaptive version of the definition of [KP16]. This definition goes beyond just requiring that an efficient adversary cannot prove a false statement on any actual computation, i.e. on any machine M and database D, but additionally requires that the prover cannot prove contradicting statements *even on digests that are generated maliciously.* Namely, a prover cannot find a digest d, machine M, and two accepting proofs for two different outputs of M^D. This means that even if the verifier has been cheated and given a malicious digest, there is still a well defined "functionality" associated with this digest for any machine M. This makes the separation between the offline and online stages even more substantial. This form of adaptive soundness, while it may seem artificial, has been used in [BBK+16] to construct three-message zero-knowledge arguments with soundness against provers with bounded non-uniformity.

3. **Weak, Generic Computational Assumptions.** The soundness of our scheme relies on the existence of any succinct 2-message (computational) private information retrieval (PIR) scheme. Such schemes are known to follow from standard concrete hardness assumptions such as ϕ-hiding or learning with errors.

[2]The prover is modeled as a RAM machine, rather than a Turing machine, in order to match the model of the underlying machine M.

[3]Our digest, as in [KP16], is simply a tree commitment of the database D, and the prover needs to store the whole tree.

Remaining Open Problems. Similarly to previous works, our protocol is vulnerable to the "verifier rejection problem": If one can observe whether the verifier rejects a large number of maliciously crafted proofs, one can then completely learn the verifier's secret key and violate soundness. This problem appears inherent in schemes which, like ours, are based on the PCP-to-argument transformation of [BMW98] (described in Section 4.1.3). The second is to achieve computation privacy while preserving RAM efficiency, i.e. to hide the delegated data and computation from the prover. Computation privacy can be achieved generically through fully homomorphic encryption (FHE), but this necessitates converting the prover into a circuit, which destroys the sometimes exponential efficiency benefits that we obtain for a prover implemented as a random-access machine (RAM).

It is only known how to address these problems under very strong hardness assumptions such as indistinguishability obfuscation or cryptographic multilinear maps. Under such assumptions, publicly verifiable delegation schemes were introduced for deterministic Turing Machines [PR17], as well as schemes that are additionally computation-private for Turing machine [KLW15], RAM [BGL+15, CHJV15, CH16, CCHR16], and PRAM [CCC+16, ACC+16] computations. It is not clear, however, whether secure indistinguishability obfuscators or multilinear maps actually exist, as current proposals appear to have flaws. Despite their strong assumptions, all of the aforementioned schemes except [PR17] require two messages, and it was not known how to achieve a non-interactive scheme from standard assumptions as we do in this work.

4.1.2 Batch delegation for NP

As mentioned above, we also make progress on the task of delegating NP computations. In particular, we construct a batch delegation scheme for any NP language, a problem that was recently considered by Reingold, Rothblum and Rothblum [RRR16]. More specifically, we construct a 2-message (non-adaptive) batch delegation for NP which allows a prover to prove that n-bit inputs $x_1, \ldots, x_k$ are all in an NP language L, where the communication complexity is $m \cdot \text{poly}(\lambda)$ with m being the length of a single witness. Additionally, the verifier runs in time $(kn + m) \cdot \text{poly}(\lambda)$.

The security of our scheme is based on the same assumption as our RAM delegation scheme, i.e., polynomial hardness of the underlying PIR scheme.

This stands in contrast to the recent result of [RRR16], which on the positive side constructs an interactive proof with soundness against computationally unbounded provers and without assumptions, but which on the negative side requires multiple rounds of communication, applies only to a restricted "unambiguous" subclass of NP (or more generally of IP), and achieves worse communication complexity: $\tilde{O}(k + \text{poly}(m))$ (in particular this grows linearly with k). Our work shows that *arguments* for NP enjoy very strong amortization, under standard cryptographic assumptions.

We show that any *adaptively* sound batch NP delegation scheme with non-trivial amortized efficiency could be used in a black-box way with our RAM delegation scheme to obtain a fully succinct non-interactive argument of knowledge (known in the literature as a SNARK). By the work of [GW11] we know that SNARKs cannot

be achieved with black-box reductions to falsifiable assumptions, and thus we derive the same for adaptively sound batch NP delegation. This contribution is described in Section 4.6.

Adaptive, Computationally No-Signaling PCP

Our main result on PCPs, presented in Section 4.4, is that the PCP of [KRR14] maintains soundness against adaptive computationally no-signaling provers.

Recall that a classical PCP for a language L is an oracle algorithm V which, on input x, makes a few queries to a proof string π, and outputs either "accept" or "reject". If $x \in L$, then there must be a string π which causes V to accept with high probability (this is called completeness), while if $x \notin L$, every string π must cause V to reject with high probability (this is called soundness). No-signaling variants of PCPs strengthen the soundness requirement so that even if π is a more general "no-signaling" prover, the verifier must still reject with high probability.

Generalizing the notion of a string (viewed as a deterministic mapping of an index set Γ to an alphabet Σ), a prover is a probabilistic mapping from an ordered subset of Γ to an equal-sized ordered subset of Σ.[4] A prover is said to be statistically no-signaling (with respect to parameters k_{max} and ϵ) if for any sets of queries Q, Q' of size at most k_{max}, the marginal distribution of prover's answers on $Q \cap Q'$ are ϵ-statistically close whether the prover was asked on Q or on Q'. A computationally no-signaling prover is the same, but now we only require that poly(λ)-time algorithms cannot distinguish the two answer distributions with probability better than $\frac{1}{2} + \epsilon$, where λ is a security parameter. This captures a potentially much larger set of cheating provers.

Furthermore, our PCP is adaptively sound, allowing the prover to first see the PCP queries and then adaptively choose the instance x, although the prover is restricted so that the distribution of x (computationally) does not signal about the queries. Despite the apparent weakness of this form of adaptivity, we show that it can offer a new pathway for a malicious prover to prove false statements. Namely, there exists a PCP which is sound against no-signaling provers, but loses all soundness when a (no-signaling) prover is allowed to adaptively choose x.

While many parts of our proof mirror that of [KRR14], our proof is somewhat shorter and more modular due in part to abstractions introduced by [PR17]. The transition from PCPs to a delegation scheme in Section 4.5 is similar to [KP16], but the proof is quite different. Along the way, we simplify a significant component – specifically, in previous works the verifier must delegate an expensive step of its computation to the prover using the previous protocol of [KR09]. Unfortunately, this needs to be done in parallel to avoid increasing the round complexity of the argument system, so they do not use [KR09] in a black-box way, and hairy composition issues arise. We show instead that the verifier's expensive step can actually be implemented efficiently (without any help from the prover)., thus eliminating the problem. We view this as an additional contribution of our work, deepening our understanding of the

[4]This definition limits the verifier to make non-adaptive queries, but PCPs typically have this property automatically.

interplay of the different components in the proof. See more details in Section 4.1.3 below.

Non-Interactive Delegation and Adaptive Soundness. Several works showed that it is possible to delegate even non-deterministic computations [DFH12, BCC+17], and furthermore, the work of [BCCT13] even constructs a *non-interactive* delegation scheme for non-deterministic computations, assuming only a *common setup*. However, all these works rely on knowledge assumptions. Such assumptions, are not only not falsifiable [Nao03], but are of a very different nature than standard hardness assumptions. Gentry and Wichs [GW11] showed that for such languages it will be hard to achieve a non-interactive protocol under polynomial-time standard assumptions.

One would hope that the techniques of [GKR08a, KRR14, KP16] could be adapted to imply a non-interactive protocol (with setup). Indeed, the first message of these delegation schemes relies only superficially on the input and on the function being computed, essentially only (an upper bound on) the time complexity of the computation needs to be known in order to produce the first message. One would hope that the first message can be generated once and for all as setup, and would allow for verification of arbitrary statements.

However, the proof of security is inherently *non-adaptive*. Namely, even though the input is not required in order to produce the first message, security is not guaranteed when the adversary gets to choose the instance after seeing the first message. Technically, this is because the proof of security relies on extracting an entire computation table from the adversary, and in order to do this, the adversary is rewound and made to respond on different queries with regards to the same input. Obviously this technique is not applicable in an *adaptive* setting.

We note that one could try to get adaptive soundness by a "brute force" approach using *complexity leveraging*: The observation is that we can always guess the instance that the adversary will choose with probability 2^n. Therefore, if we pick the parameters of the scheme such that we are guaranteed soundness even with this slowdown, then adaptive security will follow. However, this method, aside from being exorbitant in terms of communication, computation and underlying hardness assumption, requires that there is a predetermined polynomial upper bound on the length of the input. This method is therefore completely inapplicable if we intend to support arbitrary polynomial time computations on arbitrary polynomial length inputs.

4.1.3 Our Techniques

We make novel technical contributions in two aspects: First, in constructing a PCP scheme that is secure against *adaptive* and *computational* no-signaling cheating provers, as opposed to non-adaptive and statistical no-signaling provers as in previous work. Second, in converting such a PCP into protocols for adaptively secure RAM delegation or batched NP delegation. Previous proof techniques were insufficient for this, even given the enhanced PCPs. We will start by describing the delegation protocols, assuming we have a PCP at hand. Then, once we realize what properties of the PCP are actually required for the construction, we will describe the former contribution.

We will consider PCP systems for 3SAT with a standard completeness property, and will be interested in their performance against computational no-signaling (CNS) adaptive provers. A CNS adaptive prover P is a possibly randomized function, that takes a query set Q as input, and outputs a 3CNF formula φ as well as a set of answers A. We require that for every two query sets Q, Q', and for (φ, A) distributed according to $P(Q)$ and (φ', A') distributed according to $P(Q')$, it holds that $(\varphi, A|_{Q\cap Q'}) \approx (\varphi', A'|_{Q\cap Q'})$, where "$\approx$" denotes computational indistinguishability. Clearly, the honest prover that answers according to a prescribed proof string has this property (where "$\approx$" can be replaced with equality). In the non-adaptive setting, which has been considered in previous works, φ is known ahead of time to both the prover and verifier and is not generated adaptively.

We start with recalling the aforementioned [BMW98] delegation approach: generate a set of PCP queries $\{q_i\}$ according to the PCP verifier distribution. Then encrypt the queries using a homomorphic encryption scheme, each using its own key, and send the encrypted queries $\{\mathsf{Enc}_{\mathsf{pk}_i}(q_i)\}$ as the verifier's message. An honest prover will generate a PCP proof string π proving that φ is satisfiable, and use the homomorphism to compute $\{\mathsf{Enc}_{\mathsf{pk}_i}(\pi|_{q_i})\}$. The verifier can then decrypt and check that the PCP indeed accepts. We note that the generality of homomorphic encryption is not needed here, and all we need is to be able to obliviously select an entry out of a string, which is exactly the functionality provided by private information retrieval.

As for soundness, the important observation of [KRR13] is that any efficient prover, when converted to a PCP prover, must be CNS, since a non CNS prover would necessarily violate the security of the encryption scheme: If the distributions $A_1 = A|_{Q\cap Q'}$ and $A_2 = A'|_{Q\cap Q'}$ are distinguishable, then we can also distinguish between an encryption of $Q_1 = Q \setminus Q'$ and an encryption of $Q_2 = Q' \setminus Q$, which would violate the FHE security. This is done by taking the encryptions of the elements of Q_i (where the secret key is unknown), and appending the encryptions of $Q \cap Q'$ under a known secret key. Then we run the prover on the set of encrypted queries, and examine the prover's response using the distinguisher. If the distinguisher says A_1, this means we probably started with Q_1, and vice versa.

The [KRR14] approach to delegation was to construct a PCP that has soundness against no-signaling provers (statistical, in their case, but the distinction will not concern us at this point). Consider a computation, and consider a 3CNF formula φ that represents the evolution of this computation. Namely, consider the complete computation tableau, and assign a variable to each of the bits in it. A proof for a correct computation thus becomes a matter of proving that φ is satisfiable, when the variables that correspond to the initial state and the output are fixed to the respective input and output of the computation. Note that if φ is satisfiable then there is exactly one satisfying assignment which corresponds to the correct evolution of the computation.

The formula φ described above corresponds to a translation of the computation into a circuit. Thus, the prover complexity suffers accordingly (indeed the protocol of [KRR14] was designed for circuits and not for RAM machines). This was improved by [KP16], who considered RAM computations. Their approach was as follows: Rather than considering the state of the RAM memory database at every point in time,

they only considered a *computationally binding commitment* to the memory contents. By using locally updatable commitments (Merkle trees), it was possible to construct a formula φ that describes the evolution of the computation, and in particular the evolution of the committed memory database, in a compact manner, only requiring a fixed polynomial number of variables per time step, regardless of the length of the database. Since the commitments are only computationally binding, it was no longer true that φ only has one satisfying assignment. The guarantee that they get, however, is that finding two different satisfying assignments will break the underlying cryptographic primitive (a collision resistant hash function), and therefore cannot be done efficiently.

In order to use this guarantee, [KP16], following the footsteps of [KRR14], and using an abstraction due to Paneth and Rothblum [PR17], showed that any successful no-signaling prover against their PCP can be efficiently converted into a *partial assignment generator*. A partial assignment generator is an algorithm that takes any sufficiently small subset of the variables of φ and outputs an assignment for these variables that does not violate any of the clauses in which they appear. The generator is allowed to give different responses for the same variable, depending on the other elements in the set, but the assignments have to be no-signaling in an analogous sense to above: intersecting sets of variables should induce the same *distribution* on the values (up to negligible statistical or computational distance). Combining this with computational binding, they get a way to extract an entire satisfying assignment for the formula, by starting from the initial state, and filling up the table sequentially and locally, each time asking about the next variable and the previous variables that determine its value. The no-signaling property guarantees that the distribution on each and every variable is independent of the set it belongs to, and computational binding guarantees that this distribution is in fact *constant*, otherwise the binding of the commitment is broken. Thus, assuming that they have a cheating prover that successfully proves two contradicting statements about the final state of M^D, this translates into two formulae φ_1, φ_2 that correspond to computations with the same initial state but different final state. Using the assignment generator they can find the point of divergence in the two computations which will necessarily imply that they can open a commitment in two different ways, thus breaking the underlying commitment.

Adaptivity, At Last. How can we generalize this approach to the adaptive setting? We now consider a PCP whose query distribution is independent of the instance in question (most known PCPs have this property), and we allow a cheating prover to choose the formula φ, for which it wants to provide a proof *adaptively*, based on the query set Q. This means that a prover who wants to make our lives harder will never even output the same formula twice. Indeed, we can generalize the previous technique, and construct a partial assignment generator, but it will be severely crippled: Upon receiving a set of variables as input, the assignment generator will output an assignment for these variables *together with a formula* φ to which the assignment relates. Whereas all output formulae still relate to some RAM computation, we lose

our anchor for constructing the computation tableau, namely the uniqueness of the initial state. The only ray of light in this situation is that the no-signaling property is still preserved for this assignment generator. Namely the marginal distributions on subsets of variables need to be computationally indistinguishable, regardless of the sets in which they are contained, and this indistinguishability holds even in the presence of the respective φ's (in particular, the φ distribution itself should be computationally indistinguishable for different values of query sets Q).

To see how to use these properties, let us recall our notion of security. We consider a prover that is able to adaptively output M, d (where d is an alleged digest, or commitment to the initial state), together with two possible alleged outputs for the computation M^D, and respective computationally no-signaling (CNS) PCP responses. This means that even though the adversary might change the formulae φ_1, φ_2, it is always the case that it outputs two formulae with the same initial state and different final state. Therefore, there has to exist a step in the computation where the two RAM computations diverge. Perhaps surprisingly, we can find this point of divergence even if the adversary tries to shift it around! The idea is to extend the partial assignment generator into one that outputs φ_1, φ_2 at the same time, and allows to make local tests on variables of both formulae at the same time. This means that we can define a predicate of the sort "φ_1 agrees with φ_2 at step i", denote this predicate by θ_i, and test θ_i or even $(\theta_i \wedge \neg\theta_{i+1})$ using our assignment generator, since these predicates only depend on a small number of variables of the original φ_1, φ_2. The critical observation is that CNS should hold even with respect to the θ predicates, since they are efficiently computable from the variables of φ_1, φ_2. We know that θ_0 is true and θ_T is false, therefore from the CNS property, there must exist an i such that $(\theta_i \wedge \neg\theta_{i+1})$. Having found this i, we can look at the specific assignment of the φ_1, φ_2 variables to find the inconsistency and thus break the binding of the commitment scheme. This allows us to achieve an adaptive delegation scheme in spite of an elusive prover.

Batch NP Delegation. We show that similar methods also allow us to achieve 2-message non-adaptive delegation protocols for batch NP verification. Specifically, to prove that all $x_1, \ldots, x_k \in L$, for some $L \in \mathsf{NTIME}(T)$, we only require $m \cdot \mathrm{poly}(\lambda, \log T, \log k)$ bits of communication, where m is the length of a *single* witness for L (note that since we are in the non-adaptive setting, we can assume that m is known ahead of time). Let $\mathcal{R}_L$ denote the algorithm which takes as inputs x and w, and decides whether w is a valid witness that $x \in L$.

To see how this is done, let us first consider the degenerate case where $k = 1$. In this case the solution is clear: The prover sends the witness w, together with a short proof of the NP verification process of (x, w) via the former delegation protocol. We emphasize that we must require adaptive security of the original protocol even to achieve selective security here, since the computation that is actually being verified depends on w which is adversarially chosen. We note that this protocol still has advantages over the trivial verification of (x, w) since its computational complexity is independent of the specific NP relation. Moreover, we note that prior to this work, such an NP delegation protocol, even for $k = 1$, was known only in the random oracle

model or under knowledge assumptions.

Opening up the above protocol, we essentially reduce the question of whether w is a valid witness for x to whether a certain RAM transcript verification circuit $C_{x,w}$ (with a digest of $x\|w$ hard-coded) is $\text{poly}(\lambda)$-partially assignable. The witness w is sent externally to allow the verifier to construct $C_{x,w}$. In general, the communication complexity of proving ℓ-partial assignability is $\ell \cdot \text{poly}(\lambda)$, so this protocol obtains a total communication complexity of $m + \text{poly}(\lambda)$.

Now we consider a slight variation on this protocol, where instead of sending w in the clear, the prover proves partial assignability of an augmented circuit C'_x, which has $O(|w|)$ variables, $|w|$ of which are meant to encode a witness w, and the rest of which are used to compute the digest of w and extend it into a digest of $x\|w$. Now no external witness needs to be sent; to prove that $x \in L$, it suffices to prove knowledge of an $O(|w|) + \text{poly}(\lambda)$-partial assignment generator for C'. This requires communication complexity $|w| \cdot \text{poly}(\lambda)$.

But partial assignment generators amortize extremely well. To prove knowledge of a partial assignment generator for each of the circuits $C'_{x_1}, \ldots, C'_{x_k}$, it suffices to prove knowledge of the single concatenated circuit $C'_{x_1}\| \cdots \|C'_{x_k}$ – given such a "mega"-partial assignment generator, one can always restrict one's queries to focus on an individual sub-circuit. So despite this circuit's larger size, we still only need to prove $O(|w|) + \text{poly}(\lambda)$-partial assignability, which can again be done with communication $|w| \cdot \text{poly}(\lambda)$.

We note that there are several technical hurdles that we need to overcome to make this idea work. We refer the reader to Section 4.6 for details.

Constructing the PCP

Finally, we attend to the construction of the adaptively sound PCP required for our protocols as described above. In fact, our PCP is exactly the same as the one used in [KRR13, KP16]. We prove soundness in the adaptive and computational no-signaling setting. Namely, we show that adaptive assignment generators (of the type we want) can be constructed given any computational no-signaling adaptive prover (as opposed to non-adaptive statistical no-signaling). The overall proof outline remains the same, and the repetitive parts of the proof are deferred to the appendix. Along the way we were able to simplify parts of the proof.

Adaptive Soundness: A Counter-Example. In the [KRR14] PCP for proving that $x \in L$ for some poly-time language L, the verifier's queries are non-adaptive (they do not depend on x), which gives hope that the PCP might already be sound against provers which choose x adaptively. However, this is *not* generally the case. Assuming one-way functions, there exists a PCP for the empty language with non-adaptive verifier queries, which is sound against non-adaptive provers but is unsound against adaptive provers.

We define the PCP verifier (V_0, V_1) as follows. Let $\{f_k : \{0,1\}^n \to \{0,1\}^n\}_{k \in \{0,1\}^\lambda}$ be a pseudo-random permutation family with $n > 2\lambda$ and with perfect correctness (i.e. for *every* k, f_k is a permutation). V_0 makes a single uniformly random query

$q \leftarrow \{0,1\}^n$. Given an answer $a \in \{0,1\}^\lambda$ and a string x, V_1 outputs "$x \in L$" if $x = f_a(q)$, and otherwise outputs "$x \notin L$".

A non-adaptive prover is given an arbitrary x and a uniformly random q – thus by a union bound, the probability that there exists an a such that $f_a(q) = x$ is at most $2^{\lambda - n}$, which is negligible. On the other hand, an adaptive cheating prover can choose a uniformly at random, and then pick $x = f_a(q)$. x is pseudo-random, so this is a computationally no-signaling strategy.

Adaptive Soundness of KRR. Nevertheless, we show that the PCP of [KRR14], which is not so contrived, *does* have meaningful soundness against adaptive provers. To see this, we examine the original KRR proof, and see that most of the analysis provides guarantees which are helpful no matter how x is chosen. Then, we focus on the part of the analysis that does involve x, and show that we can still obtain an adaptive partial assignment generator.

The question of whether $x \in L$ is first reduced to the partial assignability of a 3-CNF φ with clause indicator function ϕ and variables $\{V_z\}_{z \in H^m}$, such that the prover possesses an assignment $A : H^m \to \{0,1\}$ which satisfies φ. The PCP contains a low-degree extension of A, denoted by $\hat{A}$, and a multivariate polynomial $P_0 : \mathbb{F}^{3m+3} \to \mathbb{F}$, defined such that

$$P_0(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, b_1, b_2, b_3) = \hat{\phi}(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, b_1, b_2, b_3) \cdot (\hat{A}(\mathbf{z}_1) - b_1) \cdot (\hat{A}(\mathbf{z}_2) - b_2) \cdot (\hat{A}(\mathbf{z}_3) - b_3). \tag{4.1}$$

Intuitively, this means $P_0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \cdot, \cdot, \cdot)$ is identicaly 0 iff the values $\hat{A}(\mathbf{h}_1)$, $\hat{A}(\mathbf{h}_2)$, $\hat{A}(\mathbf{h}_3)$ satisfy any constraints of φ.

KRR gives a weak variant of a low-degree test for P_0 and $\hat{A}$, as well as a weak variant of a test that P_0 vanishes on H^{3m+3}. If the verifier accepts, with non-negligible probability, then the amplified prover is such that:

- $\hat{A}$ can be queried at a point $\mathbf{z}$ in H^m via "majority line interpolation". That is, one picks λ random lines $L_1, \ldots, L_\lambda$ such that $L_i(0) = \mathbf{z}$, and then query $\hat{A}$ at the set $(\cup_{i=1}^{\lambda} L_i(\mathbb{F} \setminus \{0\}))$. Most of the returned sets of points $\{(t, \hat{A}(L_i(t)))\}_{t \in \mathbb{F} \setminus \{0\}}$ lie on a (unique) low-degree curve γ_i, and there is a value v such that $\gamma_i(0) = v$ for most i.

- The analogous property is shown for P_0, and in addition the value v is guaranteed to be 0.

These checks and their analysis are entirely independent of x, and they therefore hold even when x is chosen adaptively.

In addition to these checks, the verifier additionally checks that for each of λ independent *random* choices of $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, b_1, b_2, b_3)$, Equation (4.1) holds, with respect to the adaptively chosen x (via φ). We must revisit the analysis of what it means for the prover to satisfy this check.

We show, mirroring the non-adaptive analysis of KRR, that this check ensures the existence of an *adaptive* partial assignment generator. Let $\mathcal{C}$ be any clause of the form $V_{z_1} = b_1 \vee V_{z_2} = b_2 \vee V_{z_3} = b_3$. Imagine picking uniformly lines $\{L_{i,j} : \mathbb{F} \to \mathbb{F}^m\}_{i \in [3], j \in [\lambda]}$

and lines $\{L_j : \mathbb{F} \to \mathbb{F}^{3m+3}\}_{j\in[\lambda]}$ such that the first $3m$ coordinates of $L_j(t)$ are $(L_{1,j}(t), L_{2,j}(t), L_{3,j}(t))$, and the last three coordinates of $L_j(0)$ are (b_1, b_2, b_3).

For each $t \in \mathbb{F} \setminus \{0\}$ and any $j \in [\lambda]$, the point $L_j(t)$ is independently uniformly random, so when querying the amplified prover for $\hat{A}$ at $\{L_{i,j}(t)\}_{t\neq 0}$ and P_0 at $\{L_j(t)\}_{t\neq 0}$, we know that for most $j \in [\lambda]$, it holds that

$$P_0(L_j(t)) = \hat{\phi}(L_j(t)) \cdot \prod_{i=1}^{3} \left(\hat{A}(L_{i,j}(t)) - L_j(t)_{3m+i}\right)$$

Thus (by setting $\lambda \gg |\mathbb{F}|$), it holds for most j that *for all* $t \in \mathbb{F} \setminus \{0\}$, the above equation holds. Furthermore L_j, $P_0 \circ L_j$, and $\hat{A} \circ L_{i,j}$ (for each $i \in \{1, 2, 3\}$) are all low-degree, so for these j the above equation also holds for $t = 0$. In other words, reading $\hat{A}$ at z_1, z_2, and z_3 through majority line interpolation yields values which satisfy $\mathcal{C}$ if $\mathcal{C}$ is in φ – exactly what is needed from a partial assignment generator.

The full proof of the PCP soundness is deferred to Appendix B.2.

Negligible Soundness Error via Amplification. The PCP of [KRR14] is in fact a repetition of a more simple PCP. The repetition is required in order to amplify soundness. In this work we prove a more efficient soundness amplification, which is also more modular.

Specifically, we present an amplification lemma that applies in general to no-signaling PCPs, and shows that a polynomial-time prover with noticeable success probability against a repeated PCP can be transformed into a polynomial-time prover with success probability almost 1 against a relaxed PCP verifier that only checks that some of the copies of the repetition verify correctly. This part is completely self contained (see Section 4.4.3), and is used in a black-box way for the remainder of the proof.

We remark that in our application to memory delegation, the efficiency of the soundness amplification is crucial, and indeed retroactively fixes a gap in the soundness proof for the scheme of Kalai and Paneth [KP16]. They state a lemma (attributed as an abstraction of claims in [KRR14]) which implicitly assumes a more efficient soundness amplification than was actually proved in [KRR14]. In [KRR14], the soundness amplification was *inefficient* – it produced a cheating prover whose running time was exponential in the size of the proof, even if starting from an efficient prover that cheats with non-negligible probability. As a result, the constructed partial assignment generator was also inefficient.

This was not a problem when [KRR14] delegated Turing machine computations, because the existence of an unbounded partial assignment generator is sufficient for soundness. When delegating RAM computations as in [KP16], the soundness proof finds a hash collision from a cheating prover by using the constructed partial assignment generator, which should contradict collision-resistance. However, the length of a proof in [KRR14] is at least as large as a CRHF image, so the partial assignment generator's running time (exponential in this length) is large enough to trivially break any CRHF family.

Efficiently Computing Low-Degree Extensions. In the verification process, it is necessary to compute the low-degree extension (LDE) of the 3-CNF φ's "clause indicator function". This is the case in our scheme, as well as in previous schemes [GKR08a, KRR13, KRR14, KP16, PR17]. A low degree extension of a function is a multivariate low degree polynomial over a finite field whose restriction to a specific subset of the domain is exactly the original function. Usually the cardinality of the field is polylogarithmic in the input length. In our setting, we seek an LDE of the indicator function ϕ, which takes labels of 3 literals and checks whether the clause they define appears in φ. Recall that the verifier only has M, d, which implicitly defines φ, and in particular it can compute ϕ at any point, but computing the LDE requires a global view of ϕ that cannot necessarily be computed with the resources of the verifier. Note that the verifier cannot run in time $|\varphi|$ since this is proportional to the runtime of the RAM computation.

In previous works [KRR13, KRR14, KP16] there was an additional step to the delegation process, where the computation of the LDE itself had been delegated to the prover. A fairly complicated proof was required in order to show that the delegation of the LDE can be composed with the delegation of the computation without increasing the round complexity. We get rid of this step completely, and show that the LDE can be computed in time $\mathrm{poly}(|M|, |\mathsf{d}|, \log T)$.

In a related work, Paneth and Rothblum [PR17] show that, if $\phi : \{0,1\}^{3m} \to \{0,1\}$ is a clause indicator function for the computation defined by a Turing machine M and input x, then one can efficiently evaluate a somewhat low (but not minimal) degree extension $\hat{\phi} : R^{3m} \to R$ for any ring R. To show this, they observe that low-depth boolean circuits have low-degree ring-independent arithmetic circuits, and thus they rely heavily on the fact that the domain of ϕ is $\{0,1\}^{3m}$.

However, in this work we cannot use the same setting of parameters. Specifically, to obtain an efficient prover which relies only on PIR, we need $|R|^{3m}$ to be polynomial in T, but our PCP's analysis requires $|R| \geq 3m = 3\log T$, which is a contradiction. Instead, we think of the clause indicator function as a function $\phi : H^{3m} \to \{0,1\}$, where $H \subset \mathbb{F}$ is a set with $|H| = \log T$, and $m = \frac{\log T}{\log\log T}$. We compute the *minimal* degree extension of such a ϕ, using techniques that depend on the structure of our ring (namely, we show that the formula for ϕ factors well enough for Lagrange interpolation to be an efficient technique). See details in Section 4.5.1 (and in particular in Lemma 4.23).

4.2 Preliminaries

4.2.1 Low Degree Extensions

Let $\mathbb{F}$ be a finite field, and let H be a subset of $\mathbb{F}$, and let m be some integer. If f is a function mapping $H^m \to \{0,1\}$, then there exists a unique extension of f into a function $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ (which agrees with f on H^m; i.e., $\hat{f}|_{H^m} \equiv f$), such that $\hat{f}$ is an m-variate polynomial of degree at most $(|H| - 1)$ in each variable. This function $\hat{f}$ is called the *low degree extension* of f.

Low-Degree Extensions of Strings. If x is a binary string of length n, we pick $H \subseteq \mathbb{F}$ such that $|H| = \log n$ and $|\mathbb{F}| = \text{polylog}(n)$, and m such that $m = \log n / \log\log n$. By mapping $[n]$ into H^m (in lexicographical order), we can view x as a function mapping $H^m \to \{0,1\}$, and we define x's low-degree extension accordingly.

Low-Degree Extensions of 3-CNF Formulas. If φ is a 3-CNF over a set V of n variables, then (with some ordering of V) an assignment to these variables is an n-bit string. As above, such an assignment can be represented by a function mapping $H^m \to \{0,1\}$, where each variable of φ is associated with an element of H^m. We similarly write a "clause indicator function" $\phi : H^{3m} \times \{0,1\}^3 \to \{0,1\}$ for φ:

$$\phi(v_1, v_2, v_3, b_1, b_2, b_3) = \begin{cases} 1 & \text{if the clause } v_1 = b_1 \vee v_2 = b_2 \vee v_3 = b_3 \text{ is in } \varphi \\ 0 & \text{otherwise} \end{cases}$$

In fact, we view ϕ as a function from $H^{3m+3} \to \{0,1\}$ by defining the output of ϕ as 0 if b_1, b_2, or b_3 is not in $\{0,1\}$. When we refer to the *low-degree extension of a formula* φ, we mean the low-degree extension of this clause indicator function ϕ.

In the following we assume that all algorithms have access to m, the set H and the field $\mathbb{F}$, and we assume that the elementary field operations over $\mathbb{F}$ have unit cost.

4.2.2 Computing The Low Degree Extensions of Read-Once Branching Programs

In this section we show a general result for computing the low degree extension of read-once branching programs.

4.2.3 Arithmetic Straight Line Program (ASLP)

Loosely speaking, an *arithmetic straight line program (*ASLP*)* is a program, defined over some ring $\mathcal{R}$, that is composed of a sequence of arithmetic instructions which can simply add or multiply two registers together.

Definition 4.1. *An* Arithmetic Straight Line Program *(*ASLP*) for inputs of length n, over a ring $\mathcal{R}$ and with k registers $R[1], \ldots, R[k]$, is a sequence of instructions $A = (\mathsf{ins}_1, \ldots, \mathsf{ins}_T)$, where each instruction ins_t takes one of the following forms. Either:*

1. $R[i] \leftarrow$ *INPUT*$[j]$ *for some* $i \in [k]$ *and* $j \in [n]$; *or*
2. $R[i] := \alpha$, *where* $i \in [k]$ *and* $\alpha \in \mathcal{R}$; *or*
3. $R[i] \leftarrow R[j] + R[\ell]$ *for some* $i, j, \ell \in [k]$; *or*
4. $R[i] \leftarrow R[j] \times R[\ell]$ *for some* $i, j, \ell \in [k]$; *or*
5. $R[i] \leftarrow \alpha \times R[j]$ *for some* $i, j \in [k]$ *and* $\alpha \in \mathcal{R}$; *or*

6. *OUTPUT* $\leftarrow R[i]$ *for some* $i \in [k]$.

We assume that there are m *different* *OUTPUT* *instructions. Every such* ASLP A *defines in a natural way a function* $A : \mathcal{R}^n \to \mathcal{R}^m$*, where the* m *different outputs appear in the order in which they were generated.*

When the ring $\mathcal{R}$ is clear from the context, we omit it from the notation.

Remark 4.2 (Multiplication by a Constant). Clearly the instruction $R[i] \leftarrow \alpha \times R[j]$ can be easily emulated by other instructions. The reason that we add this instruction explicitly though is that in the context of fully homomorphic encryption, multiplication by a known constant is much cheaper than multiplication of two encrypted values. In particular, multiplication by a known constant does not increase the multiplicative depth (to be defined next) of the ASLP.

We next define the multiplicative depth as the maximal number of sequential multiplication operations that was used to compute the value of any register.

Definition 4.3 (Multiplicative Depth of ASLP). *The* multiplicative depth *of an* ASLP $(\mathsf{ins}_1, \ldots, \mathsf{ins}_T)$ *with* k *registers is defined as* $\max_{i\in[k],t\in[T]}\{d_{i,t}\}$*, where the values* $\{d_{i,t}\}$ *are defined inductively in* t *as follows. We define* $d_{i,0} \stackrel{\text{def}}{=} 0$ *for all* i*, and for* $t > 0$ *we define*

$$d_{i,t} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \textit{INPUT}[j] \text{ for some } j \\ 0 & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \alpha \text{ for some } \alpha \\ \max(d_{j,t-1}, d_{\ell,t-1}) & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow R[j] + R[\ell] \\ 1 + \max(d_{j,t-1}, d_{\ell,t-1}) & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow R[j] \times R[\ell] \\ d_{j,t-1} & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \alpha \times R[j] \\ d_{i,t-1} & \text{if } \mathsf{ins}_t \text{ is } \textit{OUTPUT} \leftarrow R[j] \text{ for some } j. \end{cases}$$

Uniformity of ASLPs. We will mainly use ASLPs to describe computations of a function $f_\alpha(x)$, indexed by α, on a value x. First, a bounded time and space RAM machine gets input $\alpha \in \{0,1\}^*$ and outputs an ASLP P_α that computes f_α. We call such a combined program a *uniform* ASLP (see Definition 4.4 for the precise definition).

Definition 4.4. *Let* $\mathcal{C}$ *be a complexity class. A function family* $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ *is* computable by a $\mathcal{C}$-uniform ASLP over $\mathcal{R}$ with k registers and multiplicative depth d *if there exists a standard word RAM machine in* $\mathcal{C}$ *that on input* $\alpha \in \{0,1\}^*$ *outputs an* ASLP P_α*, with* $k(\alpha)$ *registers and multiplicative depth* d*, such that* $P_\alpha(x) = f_\alpha(x)$.

We think of the function f_α as also being parameterized by the description of the ring and so the RAM machine has explicit access to a full description of $\mathcal{R}$.

Uniform ASLPs can be emulated by RAM machines as follows.

Fact 4.5 (Emulating ASLP by RAM). If a function family $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ is computable by a $\mathsf{TISP}(T,S)$-uniform ASLP over the ring $\mathcal{R}$ with k registers, then $\{f_\alpha\}_{\alpha\in\{0,1\}}$ is computable in time upper bounded by $O(T)$ ring operations and in space $S+O(k\cdot S_\mathcal{R})$, where $S_\mathcal{R}$ is the space to store a single ring element.

Remark 4.6. The multiplicative depth of an ASLP is upper bounded by the length of the ASLP. For a TIME(T)-uniform ASLP, the length of the ASLP (and therefore also the multiplicative depth) are upper bounded by T.

Definition 4.7. *A* branching program *P for inputs of length n over an alphabet H and output space Y is an acyclic directed graph (V, E) with a designated "source" vertex s and a positive number of sink vertices $t_1, \ldots, t_\ell$ such that:*

- *Each non-sink vertex v is labeled with an index $\mathsf{Var}(v) \in [n]$, and has $|H|$ outgoing edges $\{v \to \mathsf{Succ}_h(v))\}_{h \in H}$.*
- *Each sink vertex t is labeled with an output value $y_t \in Y$.*

The size *$|P|$ of P is the number of vertices. The branching program is said to be* read-once *if for every path from source to a sink, the vertices on that path are labeled distinctly.*

A read-once branching program is oblivious if it is a layered graph with $n+1$ layers and all the vertices v within layer $i \in [n]$ are labeled with the same index, which we simply denote by $\mathsf{Var}(i)$. All vertices in the $(n+1)^{th}$ layer are sinks. The width of an oblivious read-once branching program is the maximal number of vertices of any layer in the graph.

Evaluating a Branching Program. Given any string $x \in H^n$, we can iterate the following simple rule to map any vertex v to an output in Y, which we will denote by $P_v(x)$.

> *If at a non-sink vertex v, move to $\mathsf{Succ}_{x_{\mathsf{Var}(v)}}(v)$. If at a sink vertex t, output y_t.*

By mapping x to y_t, the branching program P can thus be thought of as a function $P = P_s : H^n \to Y$.

Uniformity of Branching Programs. A function family $\{f_\alpha\}_{\alpha \in \{0,1\}^*}$ is computable by a (T, S)-uniform read-once branching program, if there exists a standard word RAM machine that on input $x \in \{0,1\}^*$ runs in time $T(x)$ and space $S(x)$ and outputs a read-once branching program P such that $P(x) = f_\alpha(x)$.

Low Degree Extension of Read-Once Branching Programs

The following result shows how to efficiently evaluate the low degree extension of functions computable by a read-once branching program.

Theorem 4.8. *Given any read-once branching program $B : H^n \to Y$ and any field $\mathbb{F}$ that contains H and Y, it is possible to evaluate the low-degree extension $\hat{B} : \mathbb{F}^n \to \mathbb{F}$ in time dominated by $O(|B| \cdot |H|^2)$ field operations.*

Moreover, viewing $\hat{B}$ as a function family indexed by B and $\mathbb{F}$, it holds that $\hat{B}$ is computable by TIME($|B| \cdot |H|^2$)*-uniform* ASLPs *of length $O(|B| \cdot |H|)$ and of multiplicative depth $O(n \cdot \log(|H|))$.*

Proof. By Fact 4.5, it suffices to show only the moreover clause. Without loss of generality, we assume that the vertex set V is $[S]$.

1. For each vertex $v \in V$, in reverse topological order (so that we visit sink vertices first and source vertices last):

   (a) If v is a sink vertex, output the instruction $R[v] \leftarrow y_v$.

   (b) Otherwise, output a sequence of $O(|H|)$ instructions that, loosely speaking, computes
   $$R[v] \leftarrow \sum_{h \in H} \hat{\chi}_h(\mathbb{z}_{\mathsf{Var}(v)}) \cdot R[\mathsf{Succ}_h(v)],$$
   where $\hat{\chi}_h : \mathbb{F} \to \mathbb{F}$ is the Lagrange interpolation polynomial for h:
   $$\hat{\chi}_h(\mathbb{z}) = \prod_{h' \in H \setminus \{h\}} \frac{\mathbb{z} - h'}{h - h'}.$$
   Such a sequence of instructions can be constructed in terms of the ASLP of Claim 4.8.3.

2. Output the instruction $\texttt{OUTPUT} \leftarrow R[s]$.

To show correctness, we must show that the algorithm computes $\hat{f}$ correctly on H^n, and that the algorithm computes a polynomial of degree $|H|-1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. Let $d(v)$ denote the "depth" of a vertex v: specifically, the length of the longest directed path from v to a sink vertex.

Claim 4.8.1. *If $\bar{\mathbf{z}} \in H^n$, then the above algorithm on input $\bar{\mathbf{z}}$ outputs $f(\mathbf{z}_1, \ldots, \mathbf{z}_n)$.*

Proof. We show that for every vertex v, the v^{th} iteration of the loop on line 1 assigns the $P_v(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ to $R[v]$. This is by induction on $d(v)$. If $d(v) = 0$, then v is a sink, and $R[v]$ is assigned y_v, which is $P_v(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ by definition.

If $d(v) > 0$, then $R[v]$ is assigned $\sum_{h \in H} \hat{\chi}_h(\mathbf{z}_{\mathsf{Var}(v)}) \cdot R[\mathsf{Succ}_h(v)]$. But $\mathbf{z}_{\mathsf{Var}(v)} \in H$, so this is simply equal to $R[\mathsf{Succ}_{\mathbf{z}_{\mathsf{Var}(v)}}(v)]$, which by the inductive hypothesis is $P_{\mathsf{Succ}_{\mathbf{z}_{\mathsf{Var}(v)}}(v)}(x)$. □

Claim 4.8.2. *The output of the algorithm on input $\mathbb{z}_1, \ldots, \mathbb{z}_n \in \mathbb{F}$ is degree $|H| - 1$ in $\mathbb{z}_1, \ldots, \mathbb{z}_n$.*

Proof. We show that for every vertex v, $A(v)$ has degree $|H|-1$ in each variable. This is by induction on $d(v)$. If $d(v) = 0$, then v is a sink vertex and $A(v)$ is a constant independent of $\mathbb{z}_1, \ldots, \mathbb{z}_n$, i.e. has degree 0.

If $d(v) > 0$, then by definition $A(v) = \sum_{h \in H} \hat{\chi}_h(\mathbb{z}_{\mathsf{Var}(v)}) \cdot A(\mathsf{Succ}_h(v))$. For every h, $d(\mathsf{Succ}_h(v)) < d(v)$, so by the inductive hypothesis $A(\mathsf{Succ}_h(v))$ has degree $|H| - 1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. Furthermore, because the branching program is read-once, $A(\mathsf{Succ}_h(v))$ is independent of $\mathbb{z}_{\mathsf{Var}(v)}$ (i.e. it has degree 0 in $\mathbb{z}_{\mathsf{Var}(v)}$). Since $\hat{\chi}_h$ has degree $|H| - 1$, it holds that $A(v)$ has degree $|H| - 1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. □

Finally, we must show that Line 1b can in fact be implemented with $O(|H| \cdot S)$ instructions. This follows from the following claim.

Claim 4.8.3 (Linear-Size ASLPs for Batch Computing Lagrange Interpolation Polynomials). *There is an algorithm that takes as input a description of a field $\mathbb{F}$, as well as a subset $H = \{h_1, \ldots, h_{|H|}\} \subseteq \mathbb{F}$, and in time $O(|H|^2)$ outputs an* ASLP *of length $O(|H|)$ (and therefore multiplicative depth $O(|H|)$) that on input $\mathbb{z} \in \mathbb{F}$ outputs $\left(\hat{\chi}_{h_1}(\mathbb{z}), \ldots, \hat{\chi}_{h_{|H|}}(\mathbb{z})\right)$, where*

$$\hat{\chi}_{h_i}(\mathbb{z}) = \prod_{j \neq i} \frac{\mathbb{z} - h_j}{h_i - h_j}.$$

Proof. The algorithm does the following:

1. Output $O(|H|)$ instructions that jointly compute all "prefix products"

$$\left\{\prod_{j=1}^{i-1}(\mathbb{z} - h_j)\right\}_{i \in [|H|]},$$

storing the i^{th} prefix product in register $R[i]$. Specifically, one such sequence of instructions is

$$\begin{aligned}
&R[1] \leftarrow 1 \\
&(R[i] \leftarrow h_{i-1})_{i \in \{2,\ldots,|H|\}} \\
&R\left[|H|+1\right] \leftarrow \texttt{INPUT}[1] \\
&\left(R[i] \leftarrow R\left[|H|+1\right] - R[i]\right)_{i \in \{2,\ldots,|H|\}} \\
&\left(R[i] \leftarrow R[i] \times R[i-1]\right)_{i \in \{2,\ldots,|H|\}}
\end{aligned}$$

2. Similarly, output $O(|H|)$ instructions that jointly compute all "suffix products" $\left\{\prod_{j=i+1}^{|H|}(\mathbb{z} - h_j)\right\}_{i \in [|H|]}$, storing the i^{th} suffix product in register $R\left[|H|+i\right]$.

3. Finally, output $O(|H|)$ instructions that jointly compute and store $\hat{\chi}_{h_i}(\mathbb{z})$ in register $R\left[2|H|+i\right]$ for every $i \in \left[|H|\right]$ by computing $R\left[2|H|+i\right] \leftarrow R[i] \cdot R\left[|H|+i\right] \cdot \prod_{j \neq i} \frac{1}{h_i - h_j}$. Specifically, one sequence of instructions that accomplishes this is

$$\begin{aligned}
&\left(R\left[2|H|+i\right] \leftarrow \prod_{j \neq i} \frac{1}{h_i - h_j}\right)_{i \in [|H|]} \\
&\left(R\left[2|H|+i\right] \leftarrow R\left[2|H|+i\right] \times R[i]\right)_{i \in [|H|]} \\
&\left(R\left[2|H|+i\right] \leftarrow R\left[2|H|+i\right] \times R\left[|H|+i\right]\right)_{i \in [|H|]}
\end{aligned} \tag{4.2}$$

□

Remark 4.9. We note that the complexity of *computing* the $O(|H|)$ instructions of Eq. (4.2) is $O(|H|^2)$. For certain choices of H and $\mathbb{F}$ it is possible to compute this sequence of instructions more efficiently, namely in time equivalent to $O(|H|)$ operations over $\mathbb{F}$. In particular, if H is of the form $\{h_i \stackrel{\text{def}}{=} h_0 + i \cdot \delta\}_{i \in \{1,\ldots,|H|\}}$ for some $h_0 \in \mathbb{F}$, $\delta \in \mathbb{F} \setminus \{0\}$, then $\prod_{i=1}^{k}(h_k - h_i) = \delta^k \cdot k!$, which is computable in $O(1)$ operations given $\prod_{i=1}^{k-1}(h_{k-1} - h_i) = \delta^{k-1} \cdot (k-1)!$. For a given field $\mathbb{F}$, such sets H exist of all sizes up to and including the characteristic of $\mathbb{F}$.

This concludes the proof of Theorem 4.8. □

In the case of *oblivious* read-once branching programs, it is possible to achieve much better multiplicative depth.

Theorem 4.10. *Viewing $\hat{B}$ as a function family indexed by B and $\mathbb{F}$, it holds that $\hat{B}$ is computable by a* $\mathsf{TIME}\Big(O\big(n \cdot (w^3 + |H|^2)\big)\Big)$*-uniform* ASLP*s of length* $O\big(n \cdot (w^3 + |H|)\big)$ *and multiplicative depth* $\lceil \log(n) \rceil + |H| - 2$.

Proof. Let B be a width w oblivious read-once branching program. We assume without loss of generality that all the layers of B have width w. Let $s \in [w]$ denote the index of the starting vertex of B within layer 1. Let $T = [w]$ denote the indices of all the sink vertices within layer $n+1$ (recall that we assumed that all the vertices within layer $n+1$ are sinks).

For every $i \in [n]$ let $B_i : H \to \{0,1\}^{w \times w}$ be a function that on input $h \in H$ outputs a matrix $M_{i,h} \in \{0,1\}^{w \times w}$ that is the adjacency matrix corresponding to a transition of the branching program from layer i to layer $i+1$ when the $\mathsf{Var}(i)^{th}$ input symbol is h. More precisely, the $(j,k)^{th}$ entry of $M_{i,h}$ is 1 if there is an h-labeled edge from the j^{th} vertex in layer i to the k^{th} vertex in layer $i+1$; otherwise, the entry is 0.

Consider the function $f : \mathbb{F}^n \to \mathbb{F}$ defined as:

$$f(\mathbb{z}) = \sum_{t \in [w]} \left(\prod_{i \in [n]} \left(\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h) \right) \right)_{s,t} \cdot y_t,$$

where by $A_{(s,t)}$ we refer to the $(s,t)^{th}$ entry of the matrix A.

By inspection, for every $\mathbf{z} \in H^n$, the function f evaluates the branching program on $\mathbf{z}$. On the other hand, f is an individual degree $|H| - 1$ polynomial. Thus, by the uniqueness of the low degree extension, $f \equiv \hat{B}$.

The uniform ASLP for evaluating f proceeds as follows. For every sink $t \in [w]$, the expression $\left(\prod_{i \in [n]} \left(\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h)\right)\right)_{s,t} \cdot y_t$ is computed by a balanced binary multiplication tree of depth $\lceil \log(n) \rceil$. Leaves of this tree correspond to a computation of $\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h)$. This expression can be computed by a $\mathsf{TIME}(O(|H|^2))$-uniform ASLP of length $O(|H|)$ and multiplicative depth $|H| - 2$. Finally, all the results (corresponding to different values of $t \in [w]$ are summed. □

Remark 4.11. The w^3 term in Theorem 4.10 can be improved to w^ω, where ω is the matrix multiplication exponent.

4.3 Our Model: Public-Key Delegation for RAM Computations

In this section we motivate and formally define the model for adaptive non-interactive RAM delegation.

RAM Computation. We consider the standard model of RAM computation where a program M can access an initial database $D \in \{0,1\}^n$. We denote by M^D an execution of the program M with initial database D. For a bit $y \in \{0,1\}$ and for a string $D_{\mathsf{new}} \in \{0,1\}^n$ we also use the notation $y \leftarrow M^{D \to D_{\mathsf{new}}}$ to denote that y is the output of the program M on initial database D, and D_{new} is the final database contents after the execution. For simplicity we think only of RAM programs that output a single bit.[5]

RAM Delegation. Delegation, or verifiable outsourcing, is concerned with a client that wishes to know the result of some computation without performing it himself. Thus the computation is delegated to a prover (or worker), that performs the computation and sends the output back to the client. However, the client wishes to ensure that the prover indeed sent the correct output, and to this end it sends to the prover a challenge message, and the prover responds with the output, accompanied by a proof. One can show that such a proof can only be computationally sound, i.e. the protocol relies on the computational limitations of the prover.

As for efficiency, denoting the complexity of the computation by T, we would like the client's work to only depend (poly)logarithmically on T. This goal might seem beside the point, since the client should at least be able to read the input and have some description of the function being computed. These could already be super-polylogarithmic in the T. Indeed, in almost all previous works on Turing machine delegation, the client's running time was allowed to be linear in the input size n, and in the description of the computation (which was assumed to be uniform and thus asymptotically constant).

This approach can be generalized even further when thinking about RAM computations as in [KP16]. One can think of a uniform RAM machine (w.l.o.g this can even be a universal machine), where the input and code are stored in the RAM dátabase (a.k.a memory or array) itself. In the RAM delegation model, the initial database is preprocessed ahead of time, so as to produce a succinct digest. Given this digest, any computation on the given database can be verified in (fixed) polylog(T) complexity. One can consider, as above, a client that constructs the database himself (and thus runs in time that depends on n and the description of the computation), but this modeling is significantly adventagous when multiple computations are ran on the same database, or in an offline-online setting.

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

A Public-Key Delegation Scheme. In previous works, the soundness of the protocol was only guaranteed if the input (which in the RAM setting contains the machine and the database) was known before the challenge message is sent. If a polynomial upper bound on the length of the input was known, then complexity leveraging could have been used to remove this restriction, at the expense of a significant increase in the communication complexity and significant strengthening of the underlying hardness assumption (namely, sub-exponential hardness would be needed).

In this work, we present an adaptively secure delegation protocol, where nothing about the input (machine and database) needs to be known in order to generate the client's challenge message, *not even the input size.* This amounts to a significant qualitative difference in the applicability of the protocol. In fact, the adaptive protocol can be viewed as *public key scheme* for delegation of computation. In particular, a client can generate at their own leisure a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$. These keys are persistent and can be used throughout the life of the system (assuming that an adversary cannot learn whether maliciously generated proofs are accepted). As usual, the public key is posted for everybody to use, and the secret key is kept private by the client. A server can then prove any statement about arbitrary computations using a client's public key, without the client's involvement. The proof can then be posted for the client to verify whenever it wants. For the verification process, the client needs the description of the computation, which is represented by the RAM machine M and the digest d of the database in the beginning of the computation. It needs to receive from the prover the final state of the computation, represented as an output y, a digest d' of the final state of the database, and the running time T, as well as the proof of correctness pf. As mentioned above, the running time of the verifier can only depend polylogarithmically on the running time of the computation T. One could consider a stronger notion where the verification is done using public information, as opposed to a secret key. This variant does not require any change in the syntax of our scheme, only in the definition of soundness (i.e. allowing our "secret key" to be known to the attacker). Unfortunately, we cannot achieve this stronger notion of security and therefore our definition only considers verification using a secret key.

For correctness, we require that if indeed the initial digest was properly generated, then the prover can produce an accepting proof using only the client's public key (and the description of the computation), and that he can do so in $\mathrm{poly}(T)$ time, i.e. the complexity of proving is comparable to the complexity of computing. The formal syntax of the primitive is given below. Note that we allow a set of public parameters to be generated ahead of time and can be used by all parties, which will allow us to strengthen the notion of soundness we can provide. We remark that in our construction, the public parameters contain only a collision resistant hash function, which can be generated using public randomness from lattice assumptions or discrete log.

The Syntax of a Public-Key Delegation Scheme. A public key delegation scheme consists of a tuple of PPT algorithms

$$(\mathsf{Setup}, \mathsf{ProcessDB}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$$

with the following syntax and efficiency:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: A PPT algorithm that takes as input a security parameter 1^λ, and outputs public parameters pp.

- $\mathsf{ProcessDB}(\mathsf{pp}, D) \to \mathsf{dt}, \mathsf{d}$: A deterministic algorithm running in time $|D| \cdot \mathrm{poly}(\lambda)$ that takes as input public parameters pp and database D, and outputs the processed data dt of size $|D| \cdot \mathrm{poly}(\lambda)$ and a digest d of size $\mathrm{poly}(\lambda)$. We will write $\mathsf{Digest}(\mathsf{pp}, D)$ to denote just d. The processed dt will contain a copy of D, which we will denote $\mathsf{dt}.D$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: A randomized polynomial-time algorithm that takes as input the security parameter 1^λ (in unary representation), and outputs a public key pk and a secret key sk.

- $\mathsf{Prove}^{\mathsf{dt}}(\mathsf{pp}, \mathsf{pk}, M) \to (y, \mathsf{d}_{\mathsf{new}}, T, \mathsf{pf})$: A deterministic algorithm, that takes as input public parameters pp, a public key pk and a RAM machine M, it runs in time $\mathrm{poly}(T, \lambda)$, where $T = \mathsf{Time}(M^{\mathsf{dt}.D})$ is the runtime of the RAM machine M with database $\mathsf{dt}.D$. Prove then outputs the result y of executing $M^{\mathsf{dt}.D}$, a digest $\mathsf{d}_{\mathsf{new}}$ of the resulting database contents, the runtime T of the computation, and a proof pf of size $\mathrm{poly}(\lambda)$.

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}, (M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}, T), \mathsf{pf}) \to b$: A deterministic algorithm running in time $|M| \cdot \mathrm{poly}(\lambda)$ that outputs an acceptance bit b.

Soundness. The basic requirement of soundness is that a prover cannot convince the verifier of a false statement. Our public-key structure allows us to consider *adaptive* security, where an adversary is allowed to choose the computation it wants to forge on *after* seeing the public parameters and the public key. No prior work was able to achieve such a strong notion of security under a standard cryptographic assumption (especially for unbounded input length).

The weakest flavor of adaptive soundness that we can require is one where a cheating prover attempts to generate a database D and a RAM machine M, such that it can convince the verifier of a false output of the computation M^D, providing Verify with an honestly generated digest of D. This corresponds to a setting where the client himself generated the digest ahead of time and kept it for verification time (or alternatively the digest had been generated or certified by a trusted party). However, we can consider (and achieve) an even stronger notion of soundness.

In our final soundness definition, even if the digest d upon which verification is performed is completely spoofed by the cheating prover (i.e. does not necessarily correspond to any database), the prover still cannot prove the correctness of two different outputs with respect to the same RAM machine M and digest d. That is, the cheating prover gets access to the public parameters and the public key of the verifier. It will generate a machine M and a digest d, and two different output values y_1, y_2, each alongside a value for the final digest of the computation $\mathsf{d}'_1, \mathsf{d}'_2$. It will produce proofs $\mathsf{pf}_1, \mathsf{pf}_2$, and will try to convince the verifier that there exists a time

bound T such that applying M to a database whose digest is d results in output y_i and digest d'_i for both $i = 1, 2$.

We also require that the prover also outputs T in *unary* to prevent a degenerate case where the prover claims to have performed a very complex computation that the security reduction itself cannot recreate. This requirement can be removed, and the prover can be allowed to prove arbitrary time $T \leq 2^\lambda$ computations, but only if we assume stronger (super-polynomial) cryptographic hardness. A formal definition follows.

Definition 4.12 (RAM Delegation). *A non-interactive adaptive RAM delegation scheme*

$$(\mathsf{Setup}, \mathsf{ProcessDB}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$$

must satisfy the following properties.

Correctness. *For every RAM machine M and database D, such that $M^{D \to D_{\mathsf{new}}} \to y$, it holds that*

$$\Pr\left[y' = y \wedge \mathsf{d}' = \mathsf{Digest}(\mathsf{pp}, D_{\mathsf{new}})\right] = 1$$

in the probability space defined by sampling

$$\begin{aligned}
&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\
&(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\
&(\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{ProcessDB}(\mathsf{pp}, D) \\
&(y', \mathsf{d}', T, \mathsf{pf}) \leftarrow \mathsf{Prove}^{\mathsf{dt}}(\mathsf{pp}, \mathsf{pk}, M)
\end{aligned}$$

Completeness. *For every RAM machine M and database D such that $\mathsf{Time}(M^D) \leq T \leq 2^\lambda$, it holds that*

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}, (M, \mathsf{d}, y, \mathsf{d}_{\mathsf{new}}, T), \mathsf{pf}) = 1\right] = 1$$

in the probability space defined by sampling

$$\begin{aligned}
&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\
&(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\
&(\mathsf{dt}, \mathsf{d}) \leftarrow \mathsf{ProcessDB}(\mathsf{pp}, D) \\
&(y, \mathsf{d}_{\mathsf{new}}, T, \mathsf{pf}) \leftarrow \mathsf{Prove}^{\mathsf{dt}}(\mathsf{pp}, \mathsf{pk}, M)
\end{aligned}$$

Soundness. *For every polynomial size circuit ensemble $\mathcal{P}^* = \{P^*_\lambda\}_{\lambda \in \mathbb{N}}$,*

$$\Pr\left[\begin{array}{ll}
\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}, (M, \mathsf{d}, y_1, \mathsf{d}'_1, T), \mathsf{pf}'_1) = 1 & \wedge \\
\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}, (M, \mathsf{d}, y_2, \mathsf{d}'_2, T), \mathsf{pf}'_2) = 1 & \wedge \\
(y_1, \mathsf{d}'_1) \neq (y_2, \mathsf{d}'_2) &
\end{array}\right] \leq \mathrm{negl}(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned}&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)\\&(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\\&(M, \mathsf{d}, y_1, \mathsf{d}'_1, \mathsf{pf}'_1, y_2, \mathsf{d}'_2, \mathsf{pf}'_2, 1^T) \leftarrow P^*_\lambda(\mathsf{pp}, \mathsf{pk})\end{aligned}$$

Stronger Notions to Consider for Future Work. As explained above, we cannot achieve (and therefore do not consider) *public verifiability* where verification does not require a secret key. Our notion of security does not even protect against "chosen proof attack" wherein a cheating prover gets to interact with a verifier and observe its outputs on maliciously crafted proofs. Indeed, our construction (as well as any other known construction from standard assumptions) falls short of achieving any type of security in this setting.

With regards to the definition of soundness, a desirable feature would be to forbid the cheating prover from proving inconsistent statements on related computations (as opposed to inconsistencies on the same computation as in our definition above). Namely, the prover should not be able to prove that $f(x) = 0$ and also that $g(x) = 0$ if $g := f + 1$. An even more ambitious goal would be to require that if a cheating prover successfully proved a statement with respect to a digest, then there exists a database that corresponds to this digest. The latter notion would imply an extension of our proof system to non-deterministic computation.

4.4 Adaptively Sound PCPs

In this section, we present a new notion of *adaptively sound PCPs*. These are probabilistically checkable proofs [AS98, ALM+98] where a cheating prover is allowed to first see all of the PCP queries, and only then *adaptively* choose both the (non-accepting) input on which it wants to generate a cheating proof, as well as the proof string itself. Naturally, it is impossible to achieve security against such adaptive adversaries. Therefore, we limit the cheating provers to be *computationally no-signaling* (CNS).

Adaptive CNS provers must satisfy the property that for any two sets of PCP queries $Q' \subseteq Q$ (such that Q is not too large), the distributions of answers on Q' are computationally indistinguishable, even given the respective adaptively selected inputs. Namely, an adaptive CNS cheating prover, must satisfy that for every such Q, Q' it holds that

$$(x, A|_{Q'}) \approx (x', A'),$$

where $(x, A) \leftarrow \mathsf{Prover}^*(Q)$ and $(x', A') \leftarrow \mathsf{Prover}^*(Q')$. We note that this means that unlike the conventional definition of PCP, the PCP queries are chosen independently of the instance. This might seem weird at first glance, but in fact known PCP constructions usually have this property. All that is required is some upper bound on the length of the instance, which we take to be 2^λ, where λ is our security parameter.

For technical reasons, we need to consider cheating provers that are somewhat more involved. As explained in Section 4.3, to prove soundness of our RAM dele-

gation scheme, we need to rule out the adversary's ability to come up with a RAM computation M, a digest d, two outputs y_1, y_2, with corresponding new digests, $\mathsf{d}_1, \mathsf{d}_2$, together with two valid proofs. As we show in Section 4.5, this translates, via our construction, to a PCP prover Prover^* that given a set of queries Q, produces two such instances $(M, \mathsf{d}, y_1, \mathsf{d}_1)$ and $(M, \mathsf{d}, y_2, \mathsf{d}_2)$ (which we will think of as two 3CNFs), together with answers A_1 and A_2, respectively, corresponding to the PCP queries Q.

We thus consider any PCP cheating prover, Prover^*, that for any two sets of PCP queries $Q' \subseteq Q$ (such that Q is not too large), outputs $(\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(Q)$ and $(\varphi_1', \varphi_2', A_1', A_2') \leftarrow \mathsf{Prover}^*(Q')$, and we require that

$$(\varphi_1, \varphi_2, (A_1)|_{Q'}, (A_2)|_{Q'}) \approx (\varphi_1', \varphi_2', A_1', A_2')$$

We note that since the prover is adaptive, working on each of the instances separately is insufficient: the prover might output pairs of inconsistent instances, but if we look at the marginal distribution of each element in the pair, these distributions might even be identical. Therefore, our soundness crucially relies on the cheating prover outputting two instances together with PCP answers for both.

The notion of soundness that we prove is a refinement of notions presented in previous works[6]. We prove that an adaptive CNS adversary that convinces our PCP verifier with noticeable probability implies a *partial assignment generator*. A partial assignment generator is an algorithm that takes as input a set of variables W for a 3SAT formula, and outputs a pair of formulae φ_1, φ_2, together with assignments for the variables W in both φ_1, φ_2 that does not refute the formulae. Furthermore, the distribution of the formulae should be computationally indistinguishable from the one output by the cheating prover, and the partial assignment needs to be (computational) no-signaling (otherwise the task may be trivial).

To understand why this is a meaningful notion of soundness, recall that a cheating prover outputs formulae that correspond to inconsistent statements. A partial assignment generator will allow us to track down the point in the computation where the two statements diverge, and show that being able to prove divergent statements allows to break the underlying hardness assumption. See details in Section 4.5. The crux of our approach is that even though the adversary is allowed to adaptively change the formulae it's outputting, the no-signaling property guarantees that it cannot shift around the point of divergence.

Organization of This Section. We start in Section 4.4.1 with the definition of adaptively sound PCP, and our notion of adaptive cheating provers. Then, in Section 4.4.2 we present Theorem 4.19, which states that there exists a PCP system, that considers adaptive CNS cheating provers, which has (local) soundness guarantees. The PCP construction itself, and parts of the proof, are very similar to [KRR14], and hence are deferred to the appendix. However, the part of the proof, which consists of a soundness amplification lemma, we change completely and simplify. The new proof for this is presented in Section 4.4.3.

[6]Formally, our theorem is incomparable to that of [KRR14], but its use of computational no-signaling seems to make it more suited to cryptographic applications.

4.4.1 Definitions

We start with a variant of the classical definition of a PCP system. The only change is that the PCP verifier is *oblivious* – its queries Q do not depend on the instance x; rather, they depend only on the length of x. As mentioned above, known PCP constructions have this property.

Definition 4.13. *A $k(\cdot)$-query PCP for an* NP *language L is a tuple of* PPT *algorithms (V_0, V_1, P), such that:*

- *(Completeness) For all $\lambda \in \mathbb{N}$ and $x \in L$ (with witness w) such that $|x| \leq 2^\lambda$,*

$$\Pr\left[V_1(\mathsf{st}, x, \pi|_Q) = 1 \;\middle|\; \begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0(1^\lambda) \\ \pi \leftarrow P(1^\lambda, x, w)\end{array}\right] = 1 \ ,$$

  The PCP proof π is a string of symbols over some alphabet Σ. It will be convenient for us to view this string as indexed by a set Γ (w.l.o.g think of $\Gamma = [\ell]$ where ℓ is the length of the string), and $Q \subseteq \Gamma$. Alternatively, π can be thought of as a function from Γ to Σ.

- *(Soundness) For all $\lambda \in \mathbb{N}$, $x \notin \mathcal{L}$, $|x| \leq 2^\lambda$, and all proof strings π,*

$$\Pr\left[V_1(\mathsf{st}, x, \pi|_Q) = 1 \middle| (Q, \mathsf{st}) \leftarrow V_0(1^\lambda)\right] \leq \frac{1}{2}$$

- *(Query Efficiency) If $(Q, \mathsf{st}) \leftarrow V_0(1^\lambda)$, then $|Q| \leq k(\lambda)$ and the combined runtime of V_0 and V_1 is* $\mathrm{poly}(\lambda)$.

- *(Prover Efficiency) The prover P runs in polynomial time, where its input is $(1^\lambda, x, w)$.*

Remark 4.14. A reader may wonder, since V_0 is not given $|x|$, how P can achieve proofs with length polynomial in $|x|$, rather than polynomial in the bound on $|x|$ that V_0 knows (which is 2^λ). In a typical PCP scheme, we assume the verifier and prover are both given a bound N such that $|x| \leq N$. The prover runs in time $\mathrm{poly}(N)$ and the verifier runs in time $\mathrm{poly}(\log N)$. From such a PCP, we can achieve the above "instance-specific efficiency" generically: we just use λ schemes in parallel, where in the i^{th} scheme $N = 2^i$. When given x, a prover generates a proof for the scheme in which $N/2 < x \leq N$.

Adaptive CNS Provers. We now consider cheating PCP provers, who are adaptive and computationally no-signaling (CNS) provers, as described above.

Definition 4.15 (Adaptive Adversarial PCP Prover). *An adaptive adversarial PCP prover for a language L is a polynomial-sized ensemble $\{P^*_\lambda\}$ of probabilistic circuits that take a finite set of queries Q and output strings x_0 and x_1 that it claims are in L, together with assignments A_0 and $A_1 : Q \to \Sigma$.*

Definition 4.16 (Adaptive Computationally No-Signaling Prover). *An adaptive adversarial PCP prover is said to be $k_{max}(\cdot)$-wise computationally no-signaling if for all sets $Q'_\lambda \subseteq Q_\lambda$, with $|Q_\lambda| \leq k_{max}(\lambda)$, it holds that*

$$(x_0, x_1, A_0|_{Q'_\lambda}, A_1|_{Q'_\lambda}) \approx_c (x'_0, x'_1, A'_0, A'_1)$$

in the probability space defined by sampling

$$(x_0, x_1, A_0, A_1) \leftarrow P^*_\lambda(Q_\lambda)$$
$$(x'_0, x'_1, A'_0, A'_1) \leftarrow P^*_\lambda(Q'_\lambda).$$

Adaptive Local Soundness. As explained above, our notion of adaptive local soundness relies on the notion of partial assignment generators. A formal definition follows. From this point and on, we will focus only on proofs for 3SAT (with the natural witness relation).

Definition 4.17 (Adaptive Local Assignment Generator). *An adaptive $(\ell(\cdot), \epsilon(\cdot))$-partial assignment generator* Assign *on variables $\{V_\lambda\}_{\lambda\in\mathbb{N}}$ is an algorithm that takes as input a security parameter 1^λ and a set of at most $\ell(\lambda)$ queries $W \subseteq V_\lambda$, and outputs two 3-CNFs (φ_0, φ_1) (each on variables V_λ) and assignments $A_0 : W \to \{0,1\}$ and $A_1 : W \to \{0,1\}$, such that the following two properties hold.*

- **Everywhere Local Consistency.** *For every λ, every set $W \subseteq V_\lambda$ with $|W| \leq \ell(\lambda)$, with probability at least $1 - \epsilon(\lambda)$ over sampling*

  $$(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Assign}(1^\lambda, W),$$

  the assignments A_0 and A_1 are respectively "locally consistent" with the formulas φ_0 and φ_1. That is, every clause in φ_b whose variables v_1, v_2, v_3 are in W is satisfied by the assignment $A_b(v_1), A_b(v_2), A_b(v_3)$.

- **Computational No-Signaling.** *For every ensemble $\{W'_\lambda, W_\lambda\}_{\lambda\in\mathbb{N}}$ of subsets $W'_\lambda \subset W_\lambda \subset V_\lambda$ with $|W_\lambda| \leq \ell(\lambda)$, we have*

  $$\varphi_0, \varphi_1, A_0|_{W'_\lambda}, A_1|_{W'_\lambda} \approx_c \varphi'_0, \varphi'_1, A'_0, A'_1$$

  in the probability space defined by sampling

  $$(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Assign}(1^\lambda, W_\lambda)$$
  $$(\varphi'_0, \varphi'_1, A'_0, A'_1) \leftarrow \mathsf{Assign}(1^\lambda, W'_\lambda)$$

We say that Assign *is an $\ell(\cdot)$-partial assignment generator if it is an $(\ell(\cdot), \mathrm{negl}(\cdot))$-partial assignment generator for some negligible function* negl.

Threshold Verifiers. For the purpose of our construction and proof, we consider a variant of the common sequential repetition principle, where properties of a PCP are enhanced by running the verifier several times on the same proof string. In our

case, the purpose will not be to enhance soundness but rather to prove no-signaling properties. We would like to consider a case where the verifier accepts even if not all answers are satisfying, but rather only some fraction. We will require the definition of a threshold verifier as follows.

Definition 4.18 (Threshold Verifier). *Given a PCP verifier $V = (V_0, V_1)$, we define the t-of-n threshold verifier $(V_0^{\otimes n}, V_1^{\geq t})$ (where both n and t may be functions of λ).*

$V_0^{\otimes n}$ takes a security parameter 1^λ and does the following:

1. *Compute $(Q_i, \mathsf{st}_i) \leftarrow V_0(1^\lambda)$ for $i = 1, \ldots, n$.*
2. *Output $(\cup_{i=1}^n Q_i, (Q_1, \mathsf{st}_1), \ldots, (Q_n, \mathsf{st}_n))$.*

$V_1^{\geq t}$ takes input $(((Q_1, \mathsf{st}_1), \ldots, (Q_n, \mathsf{st}_n)), x, A)$ and does the following:

1. *Compute $y_i \leftarrow V_1(\mathsf{st}_i, x, A_{Q_i})$ for $i = 1, \ldots, n$.*
2. *Output 1 if at least t of the y_i's are 1; otherwise outputs 0.*

For the special case where $t = n$, we write $V_1^{\otimes n}$ instead of $V_1^{\geq n}$.

4.4.2 An Adaptively Sound PCP with Local Soundness

We show that there exists an adaptively sound PCP with local soundness against CNS cheating provers. Relating our result to previous work, [KRR14] showed a PCP in which: if a non-adaptive statistically no-signaling prover convinces the verifier to accept a 3-CNF φ, then there is a (non-adaptive) partial assignment generator for φ. We show that the same PCP provides stronger soundness guarantees. The cheating prover is now allowed to only be *computationally* no-signaling, and can be adaptive (i.e. output the instances on which they attempt to convince the verifier). Consequently, the partial assignment generator, which results from a cheating prover, is computationally no-signaling and adaptive.

Theorem 4.19. *There is a PCP $(\tilde{V}_0, \tilde{V}_1, \tilde{P})$ for 3-SAT satisfying the following soundness property.*

For every $c > 0$ there is a PPT oracle machine Assign_c and a polynomial ℓ_0 such that for all polynomials ℓ, for all poly-sized adversarial PCP provers $\mathcal{P}^* = \{P_\lambda^*\}_{\lambda \in \mathbb{N}}$ which are $(\ell \cdot \ell_0 + \lambda^2)$-wise CNS, if for all λ in an infinite set Λ,

$$\Pr\left[\begin{array}{l|l} 1 \leftarrow \tilde{V}_1(\mathsf{st}, \varphi_0, A_0) \ \wedge & (Q, \mathsf{st}) \leftarrow \tilde{V}_0(1^\lambda) \\ 1 \leftarrow \tilde{V}_1(\mathsf{st}, \varphi_1, A_1) & (\varphi_0, \varphi_1, A_0, A_1) \leftarrow P_\lambda^*(Q) \end{array}\right] \geq \lambda^{-c},$$

then for all $\lambda \in \Lambda$, $\mathsf{Assign}_c^{P_\lambda^*}(1^\lambda, \cdot)$ is an adaptive ℓ-partial assignment generator such that for any set of variables W, sampling

$$(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Assign}_c^{P_\lambda^*}(1^\lambda, W)$$

produces a distribution on (φ_0, φ_1) that is computationally indistinguishable from that obtained by sampling (φ_0, φ_1) according to the conditional distribution

$$\begin{array}{l|l} (Q, \mathsf{st}) \leftarrow \tilde{V}_0(1^\lambda) & \tilde{V}_1(\mathsf{st}, \varphi_0, A_0) = 1 \ \wedge \\ (\varphi_0, \varphi_1, A_0, A_1) \leftarrow P^*_\lambda(Q) & \tilde{V}_1(\mathsf{st}, \varphi_1, A_1) = 1. \end{array}$$

Moreover, $\tilde{V}_1$ only uses the LDEs of φ_0 and φ_1. Given oracle access to these LDEs, $\tilde{V}_0$ and $\tilde{V}_1$'s total running time is $\mathrm{poly}(\lambda)$ *for some fixed (i.e. independent of $|\varphi_0|$ or $|\varphi_1|$) polynomial* poly.

Proof. As explained above, our PCP is identical to the one from [KRR14]. However, our theorem requires different guarantees than previous works and we do it in a more modular manner and prove a new amplification theorem on CNS provers along the way. We start with a "base PCP" which is formally described in Appendix B. We denote this PCP by (V_0, V_1, P), and we let k (=polylog(λ)) denote the number of queries made by V_0. We define our PCP $(\tilde{V}_0, \tilde{V}_1, \tilde{P})$ to be $(V_0^{\otimes\lambda}, V_1^{\otimes\lambda}, P)$, i.e. the λ-repeated (λ-out-of-λ) version of (V_0, V_1, P), as per Definition 4.18. Completeness holds trivially. We thus focus on proving the adaptive computational no-signaling local soundness property.

To this end, fix any ℓ, and suppose there exists an $(\ell \cdot \ell_0 + \lambda^2)$-wise computationally no-signaling cheating Prover^* (where ℓ_0 is a polynomial that will be determined later), with a constant $c \in \mathbb{N}$ such that for infinitely many λ's,

$$\Pr\left[(V_1^{\otimes\lambda})^{\hat{\phi}_1, \hat{\phi}_2}(\mathsf{st}, A_1, A_2) = 1\right] \geq \lambda^{-c}.$$

in the probability space defined by sampling

$$\begin{array}{l} (Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda); \\ (\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(Q); \end{array}$$

We show that by lowering the threshold for verification to a small yet super-logarithmic value, the success probability of a related cheating prover increases to near perfect. In particular, Lemma 4.20, which is stated and proven in Section 4.4.3 below, implies that there exists a $(\ell \cdot \ell_0 + \lambda^2 - \lambda \cdot k)$-wise computationally no-signaling cheating prover, which will be denoted by Prover^{**}, such that for any $r = \omega(\log \lambda)$ (for concreteness, the reader can think of $r = \sqrt{\lambda}$), there exists a negligible function ϵ such that for infinitely many values of λ,

$$\Pr\left[(V_1^{\geq\lambda - r})^{\hat{\phi}'_1, \hat{\phi}'_2}(\mathsf{st}, A_1, A_2) = 1\right] \geq 1 - \epsilon(\lambda),$$

in the probability space defined by sampling

$$\begin{array}{l} (Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda); \\ (\varphi'_1, \varphi'_2, A_1, A_2) \leftarrow \mathsf{Prover}^{**}(Q); \end{array}$$

Moreover, the distribution of (φ'_1, φ'_2) is indistinguishable from the conditional distri-

bution of (φ_1, φ_2) given that V accepts. Note that since $\lambda \geq k$,

$$\ell \cdot \ell_0 + \lambda^2 - \lambda \cdot k \geq \ell \cdot \ell_0.$$

In what follows we let $k_{max} = \ell \cdot \ell_0$.

The remainder of the proof is showing how to construct an assignment generator out of the amplified cheating prover. This is done using methods that are very similar to those of [KRR14]. In particular, Lemma B.2 in Appendix B.2 implies that there exists a probabilistic polynomial-time oracle machine Assign such that $\mathsf{Assign}^{\mathsf{Prover}^{**}}$ is an adaptive (ℓ, ϵ')-partial assignment generator with $\epsilon' = \epsilon \cdot \mathrm{poly}(\lambda) + \mathrm{negl}(\lambda) = \mathrm{negl}(\lambda)$, which completes the proof of the theorem. □

4.4.3 New Soundness Amplification for No-Signaling PCPs

In the proof of Theorem 4.19, we use a soundness amplification lemma that uses the notion of PCPs with "t-of-n threshold" verifiers.

Roughly speaking, in our soundness amplification lemma below, we show that if an adaptive CNS prover convinces a λ-of-λ threshold verifier with any non-negligible probability, then there is an adaptive CNS prover which convinces a corresponding $(\lambda - \omega(\log \lambda))$-of-$\lambda$ verifier with high probability. Furthermore, the latter prover produces almost the same distribution of (φ_0, φ_1) as the first, conditioned on the first convincing the λ-of-λ verifier.

Lemma 4.20 (Soundness Amplification). *For all $k(\cdot)$-query PCPs (V_0, V_1, P) for 3SAT and all $c > 0$, there is a* PPT *oracle algorithm* $\mathsf{Amplify}_c$ *such that* if *there is an adaptive k_{max}-wise CNS adversarial prover $\{P^*_\lambda\}_{\lambda \in \mathbb{N}}$ satisfying*

$$\Pr\left[V_1^{\otimes\lambda}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\otimes\lambda}(\mathsf{st}, \varphi_1, A_1) = 1\right] \geq \lambda^{-c}$$

for infinitely many λ in the probability space defined by sampling

$$\begin{aligned} &(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ &(\varphi_0, \varphi_1, A_0, A_1) \leftarrow P^*_\lambda(Q), \end{aligned} \tag{4.3}$$

then $\left\{\mathsf{Amplify}_c^{P^*_\lambda}(1^\lambda, \cdot)\right\}_{\lambda \in \mathbb{N}}$ is an adaptive $(k_{max} - \lambda \cdot k)$-wise CNS adversarial prover ensemble and for any $r = \omega(\log \lambda)$, there is a negligible function negl *such that for infinitely many λ,*

$$\Pr\left[V_1^{\geq\lambda - r}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\geq\lambda - r}(\mathsf{st}, \varphi_1, A_1) = 1\right] \geq 1 - \mathrm{negl}(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned} &(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ &(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Amplify}_c^{P^*_\lambda}(1^\lambda, Q). \end{aligned} \tag{4.4}$$

Furthermore, the distribution of (φ_0, φ_1) in (4.4) is computationally indistinguish-

able from the conditional distribution of (φ_0, φ_1) *in* (4.3) *given*

$$V_1^{\otimes\lambda}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\otimes\lambda}(\mathsf{st}, \varphi_1, A_1) = 1.$$

Proof. By assumption, there exists a constant $c > 0$ such that for infinitely many λ, P_λ^* convinces V with probability at least λ^{-c}. Let Λ denote this set of λ.

We describe the algorithm $\mathsf{Amplify}_c$. On an input set of queries $Q \in \Gamma^{k_{max}-\lambda\cdot k}$, $\mathsf{Amplify}_c$ independently samples $(Q_i, \mathsf{st}_i) \leftarrow V_0^{\otimes\lambda}(1^\lambda)$ and $(\varphi_i^0, \varphi_i^1, A_i^0, A_i^1) \leftarrow P_\lambda^*(Q \cup Q_i)$ for $i = 1, \ldots, \lambda^{c+1}$.

$\mathsf{Amplify}_c$ then outputs $(\varphi_{i^*}^0, \varphi_{i^*}^1, (A_{i^*}^0)_Q, (A_{i^*}^1)_Q)$ for the first i^* such that

$$V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \wedge V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1.$$

Call such an i^* "good". If no good i^* exists, then $\mathsf{Amplify}_c$ outputs $\perp$ (in Claim 4.21.1 below, we show that for $\lambda \in \Lambda$, this happens with negligible probability).

Let us first show the "furthermore" of the lemma.

Claim 4.20.1. *The distribution of* (φ_0, φ_1) *in* (4.4) *is computationally indistinguishable from the conditional distribution of* (φ_0, φ_1) *in* (4.3) *given*

$$V_1^{\otimes\lambda}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\otimes\lambda}(\mathsf{st}, \varphi_1, A_1) = 1.$$

Proof. Recall that the definition of computational no-signaling says that for all $Q'_\lambda \subseteq Q_\lambda$ such that $|Q_\lambda| \leq k_{max}$, the distributions

$$(\varphi_0, \varphi_1, (A_0)|_{Q'_\lambda}, (A_1)|_{Q'_\lambda})$$

and

$$(\varphi'_0, \varphi'_1, A'_0, A'_1)$$

are computationally indistinguishable, when sampling

$$\begin{aligned}(\varphi'_0, \varphi'_1, A'_0, A'_1) &\leftarrow P_\lambda^*(Q'_\lambda)\\ (\varphi_0, \varphi_1, A_0, A_1) &\leftarrow P_\lambda^*(Q_\lambda).\end{aligned}$$

This is equivalent to the following, seemingly stronger statement:

Corollary 4.21. *For all poly-sized auxiliary information* aux_λ *and sets* $Q'_\lambda \subseteq Q_\lambda$, *such that* $|Q_\lambda| \leq k_{max}$, *the distributions*

$$(\mathsf{aux}_\lambda, \varphi_0, \varphi_1, (A_0)|_{Q'}, (A_1)|_{Q'_\lambda})$$

and

$$(\mathsf{aux}_\lambda, \varphi'_0, \varphi'_1, A'_0, A'_1)$$

are computationally indistinguishable, when sampling

$$\begin{aligned}(\varphi'_0, \varphi'_1, A'_0, A'_1) &\leftarrow P_\lambda^*(Q'_\lambda)\\ (\varphi_0, \varphi_1, A_0, A_1) &\leftarrow P_\lambda^*(Q_\lambda).\end{aligned}$$

Consider $\mathsf{Amplify}'_c$, a modified version of $\mathsf{Amplify}_c$ which samples each $(\varphi_i^0, \varphi_i^1, A_i^0, A_i^1)$ from $P_\lambda^*(Q_i)$ instead of from $P_\lambda^*(Q \cup Q_i)$. But, (φ_0, φ_1) are distributed identically in the conditional distribution of (4.3) given

$$V_1^{\otimes\lambda}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\otimes\lambda}(\mathsf{st}, \varphi_1, A_1) = 1.$$

as they are in the conditional probability space

$$\left.\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ (\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Amplify}_c'^{P_\lambda^*}(1^\lambda, Q)\end{array}\right| \text{a good } i^* \text{ exists}$$

We later show (as part of Claim 4.21.1) that a good i^* exists with overwhelming probability, this distribution on (φ_0, φ_1) is statistically close to that obtained by sampling

$$\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ (\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Amplify}_c'^{P_\lambda^*}(1^\lambda, Q)\end{array} \tag{4.5}$$

without any condition.

We claim that this distribution on (φ_0, φ_1) is computationally indistinguishable from that obtained by sampling

$$\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ (\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, Q)\end{array} \tag{4.6}$$

It suffices for us to show that the distribution on

$$(\mathsf{st}_i, \varphi_i^0, \varphi_i^1, (A_i^0)|_{Q_i}, (A_i^1)|_{Q_i})$$

obtained when sampling (4.5) is computationally indistinguishable from that obtained when sampling (4.6), because these are all the intermediate values needed to compute the first good i^* and hence the resulting (φ_0, φ_1). But this indistinguishability follows from Corollary 4.21, where st_i is the auxiliary information. □

Claim 4.21.1. *For $\lambda \in \Lambda$,*

$$\Pr\left[\ V_1^{\geq\lambda-r}(\mathsf{st}, \varphi_0, A_0) = 1 \wedge V_1^{\geq\lambda-r}(\mathsf{st}, \varphi_1, A_1) = 1\ \right] \geq 1 - \mathrm{negl}(\lambda).$$

in the probability space defined by sampling

$$\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda) \\ (\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, Q).\end{array}$$

Proof. The probability that the verifier isn't convinced is bounded by the probability that there is no good i^* plus the probability that $V_1^{\geq\lambda-r}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_Q) = 0$ or $V_1^{\geq\lambda-r}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_Q) = 0$ for a good i^*. We show that both of these probabilities are negligible.

First, because P_λ^* convinces $(V_0^{\otimes\lambda}, V_1^{\otimes\lambda})$ with probability λ^{-c} and because $\mathcal{P}^*$ is CNS, the probability that no good i^* exists is bounded by $(1 - \lambda^{-c} + \mathrm{negl}(\lambda))^{\lambda^{c+1}}$,

which is negligible.

For each $b \in \{0,1\}$, the probability that $V_1^{\geq \lambda - r}(\mathsf{st}_{i^*}, \varphi_{i^*}^b, (A_{i^*}^b)_Q) = 0$ for a good i^* is equal to the conditional probability

$$\mathbb{E}\left[\Pr\left[V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^b, (A_{i^*}^b)_Q) = 0 \middle| Q, \mathsf{st}, \begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right]\right], \quad (4.7)$$

which we claim is negligibly close to

$$\Pr\left[V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^b, (A_{i^*}^b)_Q) = 0 \middle| \begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right] \quad (4.8)$$

in the probability space defined by sampling $(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda)$ and $(Q_{i^*}, \mathsf{st}_{i^*}) \leftarrow V_0^{\otimes\lambda}(1^\lambda)$ and $(\varphi_{i^*}^0, \varphi_{i^*}^1, A_{i^*}^0, A_{i^*}^1) \leftarrow P_\lambda^*(Q \cup Q_{i^*})$. Equation (4.7) is well-defined and close to Equation (4.8) because for any Q, st, we have that the absolute difference between

$$\Pr\left[\begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right]$$

and

$$\Pr\left[\begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array} \middle| Q, \mathsf{st}\right]$$

is negligible. This is because:

- P_λ^* convinces with probability λ^{-c} and is k_{max}-CNS, so

$$\Pr\left[\begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right] \geq \lambda^{-c} - \mathrm{negl}(\lambda),$$

  which is non-negligible; and

- $$\Pr\left[\left(\begin{array}{l} V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^0, (A_{i^*}^0)_Q) = 0 \ \vee \\ V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^1, (A_{i^*}^1)_Q) = 0 \end{array}\right) \wedge \left(\begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right)\right]$$

  is negligible because P_λ^* does not learn Q or Q_{i^*}; only $Q \cup Q_{i^*}$. Let us fix $Q \cup Q_{i^*}$ (but importantly not Q or Q_{i^*} individually), P_λ^*'s responses, and the randomness used by each instance of V_1. Suppose that a fraction p of the answers given by P_λ^* are accepted by both of the corresponding calls to V_1. Now

$$\Pr_{Q, Q_{i^*}}\left[\begin{array}{l} V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^0, (A_{i^*}^0)_{Q_{i^*}}) = 1 \ \wedge \\ V_1^{\otimes\lambda}(\mathsf{st}_{i^*}, \varphi_{i^*}^1, (A_{i^*}^1)_{Q_{i^*}}) = 1 \end{array}\right] \leq p^\lambda,$$

  and by Hoeffding's inequality[7]and a union bound,

$$\Pr_{Q, Q_{i^*}}\left[\begin{array}{l} V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^0, (A_{i^*}^0)_Q) = 0 \ \vee \\ V_1^{\geq \lambda - r}(\mathsf{st}, \varphi_{i^*}^1, (A_{i^*}^1)_Q) = 0 \end{array}\right] \leq e^{-2(\lambda - r - p\lambda)^2}$$

We claim that one of these bounds is negligible in λ. If $p < 1 - \frac{r}{2\lambda}$, then $p^\lambda < e^{-\omega(\log\lambda)}$. Otherwise, $\lambda - r - p\lambda \leq -r/2 = -\omega(\log\lambda)$, and so $e^{-2(\lambda-r-p\lambda)^2} \leq e^{-\omega(\log\lambda)}$. These two cases cover (non-disjointly) all possible p.

So the total probability that $V_1^{\geq\lambda-r}$ is not convinced for a good i^* is negligible. By the definition of conditional probability, $\Pr[A|B] = \Pr[A \wedge B] / \Pr[B]$. In particular, we have shown that the probability in Equation 4.8 is $\mathrm{negl}(\lambda) \cdot \lambda^{c'}$, which is negligible. □

Claim 4.21.2. $\{\mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, \cdot)\}_{\lambda\in\mathbb{N}}$ *is* $(k_{max} - \lambda \cdot k)$*-wise CNS.*

Proof. We need to show that for all sets S_λ and Q_λ with $S_\lambda \subset Q_\lambda$ and $|Q_\lambda| \leq k_{max} - \lambda \cdot k$, it holds that

$$\mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, Q_\lambda)_{S_\lambda} \approx \mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, S_\lambda).$$

We will show this by defining indistinguishable distributions B_λ and C_λ, and giving an efficiently computable randomized function f such that as distributions,

$$\mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, Q_\lambda)_{S_\lambda} \equiv f(B_\lambda) \tag{4.9}$$

and

$$\mathsf{Amplify}_c^{P_\lambda^*}(1^\lambda, S_\lambda) \equiv f(C_\lambda). \tag{4.10}$$

Here, $\equiv$ denotes equality of distributions.

Define B_λ as the distribution on $\{(i, \mathsf{st}_i, Q_i, \varphi_i^0, \varphi_i^1, (A_i^0)_{S_\lambda\cup Q_i}, (A_i^1)_{S_\lambda\cup Q_i})\}_{i=1}^{\lambda^{c+1}}$ in the probability space defined by sampling, for each i,

$$\begin{aligned}(\mathsf{st}_i, Q_i) &\leftarrow V_0^{\otimes\lambda}(1^\lambda)\\ (\varphi_i^0, \varphi_i^1, A_i^0, A_i^1) &\leftarrow P_\lambda^*(Q_\lambda \cup Q_i)\end{aligned}$$

Define C_λ as the distribution on $\{(i, \mathsf{st}_i, Q_i, \varphi_i^0, \varphi_i^1, A_i^0, A_i^1)\}_{i=1}^{\lambda^{c+1}}$ in the probability space defined by sampling, for each i,

$$\begin{aligned}(\mathsf{st}_i, Q_i) &\leftarrow V_0^{\otimes\lambda}(1^\lambda)\\ (\varphi_i^0, \varphi_i^1, A_i^0, A_i^1) &\leftarrow P_\lambda^*(S_\lambda \cup Q_i)\end{aligned}$$

Then, f is defined to find the first "good" i^*, and output $(\varphi_{i^*}^0, \varphi_{i^*}^1, (A_{i^*}^0)_{S_\lambda}, (A_{i^*}^1)_{S_\lambda})$. Equations (4.9) and (4.10) are then easy to verify.

[7] Hoeffding's inequality[Hoe63] states that if $X_1, \ldots, X_n$ are independent Bernoulli distributions with parameter p, and H denotes $\sum_i X_i$, then

$$\Pr[H \geq (p+\epsilon)n] \leq e^{-2\epsilon^2 n}.$$

Hoeffding also showed that this inequality holds even if $X_1, \ldots, X_n$ are not independent, but are chosen uniformly without replacement from a population of $\{0,1\}$'s, a fraction p of which are 1's.

It remains to show that $B_\lambda \approx C_\lambda$. This follows by a simple hybrid argument over the different i's. Indeed, we just need that for each i,

$$P^*_\lambda(Q_\lambda \cup Q_i)_{S_\lambda \cup Q_i} \approx P^*_\lambda(S_\lambda \cup Q_i),$$

which is guaranteed by the fact that $\mathcal{P}^*$ is k_{max}-wise CNS. □

This concludes the proof of Lemma 4.20. □

4.5 Adaptive RAM Delegation

In this section, we use Theorem 4.19 to construct a RAM delegation scheme. Our construction is essentially the same as in [KP16]: To prove that $M^{D \to D_{\mathsf{new}}} \to y$ for $\mathsf{d}_{\mathsf{new}} = \mathsf{Digest}(D_{\mathsf{new}})$ and $\mathsf{Time}(M, D) \leq T$, we use the PCP to prove the local satisfiability of a related 3-CNF $\varphi = \varphi_{M,\mathsf{d},y,\mathsf{d}_{\mathsf{new}},T}$ (where $\mathsf{d} = \mathsf{Digest}(D)$). The formula φ is defined in Section 4.5.1. It has $O(T \cdot \mathrm{poly}(\lambda))$ variables and verifies a transcript of M using a CRHF $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$. If $M^D \not\to y$, then a collision in h can be found from M, D, and any assignment to φ. In fact, even a *local* assignment generator for φ suffices, and this is the property that was used in [KP16].

First, we show how to make the verifier in this scheme run in fixed polynomial time, independent of $\mathsf{Time}(M, D)$. To do this, we show how to take advantage of the repetitive and local structure of φ to efficiently compute its clause indicator function's low-degree extension $\hat{\phi}$ (see Claim 4.23). In [KRR14], the verifier delegates the computation of $\hat{\phi}$ using the protocol of [GKR08a]. We note that this is not straight-forward: [KRR14] proved the soundness of this composition against *statistically* no-signaling provers, but the proof does not apply to provers which are only computationally no-signaling. We consider the verifier's direct computation of $\hat{\phi}$ to be a simplification as well as an optimization to the overall protocol.

Finally, we show adaptive soundness. Here we grapple with the fact that even if the PCP verifier accepts a 3-CNF φ given by an adaptive cheating prover P^*, there is not necessarily a partial assignment generator for φ. Rather, there is an adaptive partial assignment generator Assign which outputs $\varphi_{M,\mathsf{d},y,\mathsf{d}_{\mathsf{new}},T}$ for some distribution on $(M, \mathsf{d} = \mathsf{Digest}(D), y, \mathsf{d}_{\mathsf{new}})$. While this does not allow us to reconstruct the full execution transcript of M^D for any single (M, D), we are still able to find a fixed set of variables W such that by querying $\mathsf{Assign}(W)$, we observe a false hash tree proof (and hence obtain a hash collision) with non-negligible probability.

As mentioned in Section 4.3, we actually achieve a stronger notion of soundness: no prover can prove two different "correct" $(y, \mathsf{d}_{\mathsf{new}})$ outputs for any adversarially chosen digest d and machine M (in particular, there is not necessarily a D such that $\mathsf{d} = \mathsf{Digest}(D)$). This is stronger because if d were equal to $\mathsf{Digest}(D)$, then completeness guarantees we can prove the correct result, and hence cannot prove any incorrect result.

4.5.1 RAMs as 3-CNF Formulas

[KP16] gives a 3-CNF that succinctly verifies a RAM machine's execution transcript (under computational assumptions) by using Merkle trees. The following lemma describes its essential properties.

Lemma 4.22 ([KP16]). *There exist deterministic polynomial-time algorithms*

$$\mathsf{MakeCNF}, \mathsf{ProcessDB}, \mathsf{Transcript}, \mathsf{FindCollision},$$

such that for any hash function $h : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$, RAM machine M, digest d, output y, digest d', and time bound T, $\mathsf{MakeCNF}(h, M, \mathsf{d}, y, \mathsf{d}', T)$ is a 3-CNF $\varphi \triangleq \varphi_{in}^{M,\mathsf{d}} \wedge \varphi^{M,T,h} \wedge \varphi_{out}^{y,\mathsf{d}'}$ with the following structure:

1. *$\varphi^{M,T,h}$ has $T' = T \cdot |M| \cdot \mathrm{poly}(\lambda)$ variables. The set of variables V is partitioned into $T+1$ disjoint sets $V_0, \ldots, V_T$ with identical sets of clauses in $\varphi^{M,T,h}$ between adjacent layers. Each layer V_i contains (among other things):*
   - *Variables $V_i.q$ whose values encode the RAM machine's local state after i execution steps.*
   - *Variables $V_i.\mathsf{d}$ whose values encode a digest of the database contents after i execution steps.*
   - *Variables $V_i.y$ whose values encode the output of M if it has terminated by the i^{th} step, and encode $\perp$ otherwise.*
2. *$\varphi_{in}^{M,\mathsf{d}}$ has clauses only on V_0, which enforce that $V_0.q = M.q_0$, $V_0.\mathsf{d} = \mathsf{d}$, and $V_0.y = \perp$. $\varphi_{out}^{y,\mathsf{d}'}$ has clauses only on V_T, which enforce that $V_T.y = y$ and $V_T.\mathsf{d} = \mathsf{d}'$.*
3. *If $M^{D \to D_{\mathsf{new}}} \to y$ with $(\mathsf{d}, \mathsf{dt}) = \mathsf{ProcessDB}(h, D)$ and $\mathsf{d}' = \mathsf{Digest}(h, D_{\mathsf{new}})$ and $\mathsf{Time}(M^D) \leq T$, then $\mathsf{Transcript}^{\mathsf{dt}}(1^T, M)$ outputs a satisfying assignment A for φ in time $T \cdot |M| \cdot \mathrm{poly}(\lambda)$ such that $A(V_0.\mathsf{d}) = \mathsf{d}$ and $A(V_T.\mathsf{d}) = \mathsf{d}'$.*
4. *If two locally-consistent assignments A and A' for φ, both on $V_i \cup V_{i+1}$, agree on $V_i.q \cup V_i.\mathsf{d} \cup V_i.y$ but disagree on $V_{i+1}.q \cup V_{i+1}.\mathsf{d} \cup V_{i+1}.y$, then the output of $\mathsf{FindCollision}(1^T, \varphi, A, A')$ is a pair (x, x') such that $x \neq x$ and $h(x) = h(x')$ – in other words, a collision under h.*

Now fix $M, \mathsf{d}, T, h, y, \mathsf{d}'$, and let φ and T' be as above. Let $\phi : V^3 \times \{0,1\}^3$ denote the "clause indicator function" of φ. That is,

$$\phi(v_1, v_2, v_3, b_1, b_2, b_3) = \begin{cases} 1 & \text{if the clause } (v_1 = b_1 \vee v_2 = b_2 \vee v_3 = b_3) \text{ is in } \varphi \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbb{F}$ be any finite field with a subset H of size $\log T'$, and let m be $\frac{\log T'}{\log \log T'}$. Given any injective mapping $e : V \hookrightarrow H^m$, one can view ϕ as a function mapping $H^{3m+3} \to$

$\{0,1\}$.

$$\phi_e(i_1,i_2,i_3,b_1,b_2,b_3) = \begin{cases} 1 & \text{if } \varphi \text{ contains the clause } \bigvee_{j=1}^{3}(e^{-1}(i_j) = b_j) \\ 0 & \text{otherwise (including if } b_j \notin \{0,1\} \text{ or } i_j \notin \mathsf{Img}(e)) \end{cases}$$

Using property 1 of Lemma 4.22 above, we prove the following lemma, assuming that operations on $\mathbb{F}$ have unit cost.

Lemma 4.23. *There is an efficiently computable and invertible mapping $e : V \hookrightarrow H^m$ such that the low-degree extension*

$$\hat{\phi} : \mathbb{F}^{3m+3} \to \mathbb{F}$$

of

$$\phi : H^{3m+3} \to \{0,1\}$$

is efficiently computable at any point $z \in \mathbb{F}^{3m+3}$ in time $\mathrm{poly}(|M|,\lambda)$.

Proof. We first note that it suffices to efficiently compute the low-degree extension of $\phi^{M,T,h}$, because $\hat{\phi} = \hat{\phi}_{in}^{M,\mathsf{d}} + \hat{\phi}^{M,T,h} + \hat{\phi}_{out}^{y,\mathsf{d}'}$, and $\varphi_{in}^{M,\mathsf{d}}$ and $\varphi_{out}^{y,\mathsf{d}'}$ are on a small ($\mathrm{poly}(\lambda)$) number of variables W, so their LDEs can be computed in time $\tilde{O}(W^3) = \mathrm{poly}(\lambda)$.

The following three claims follow from Theorem 4.8.

Claim 4.23.1. *For any $h^* \in H$, there is a univariate polynomial χ_{h^*} over $\mathbb{F}$ with degree at most $|H|-1$ such that for all $h \in H$,*

$$\chi_{h^*}(h) = \begin{cases} 1 & \text{if } h = h^* \\ 0 & \text{otherwise} \end{cases}$$

and $\chi_{h^*}(x)$ for $x \in \mathbb{F}$ can be evaluated in $O(|H|)$ time.

Claim 4.23.2. *For all $k > 0$ and $n > 1$, there is an nk-variate polynomial δ over $\mathbb{F}$ with degree at most $|H|-1$ in each variable such that for $h_1,\ldots,h_n \in H^k$,*

$$\delta(h_1,\ldots,h_n) = \begin{cases} 1 & \text{if } h_1 = \ldots = h_n \\ 0 & \text{otherwise.} \end{cases}$$

and δ can be evaluated in time $O(nk \cdot |H|^2)$

Claim 4.23.3. *When H^k is identified with $[|H|^k]$ using a lexicographic ordering, there is a $3k$-variate polynomial s of degree at most $|H|-1$ in each variable such that for every $x,y,z \in H^k$,*

$$s(x,y,z) = \begin{cases} 1 & \text{if } x = y \wedge z = x+1 \\ 0 & \text{otherwise.} \end{cases}$$

and s is computable in time $O(k^3 \cdot |H|^3)$.

We now have the necessary tools to prove Lemma 4.23. Let $k = \lceil \log_{|H|} T \rceil$, and define $e : V \hookrightarrow H^k \times H^{m-k}$ so that the j^{th} variable in V_i maps to (i, j), with lexicographic correspondences between $[T]$ and H^k and between $[|V_i|]$ and H^{m-k}. We will let W denote the size of each layer. Note $W = \text{poly}(M, \lambda)$.

Now, recall that the clauses in φ are repeated and identical between adjacent layers. We consider separately the clauses which are entirely contained within a layer, and the clauses which have variables in both layers. We can assume without loss of generality that the latter type of clauses have one variable in layer $i + 1$ and two variables in layer i.

These clauses can be described by functions

$$\psi_A : [W]^3 \times \{0,1\}^3 \to \{0,1\}$$

and

$$\psi_B : [W]^3 \times \{0,1\}^3 \to \{0,1\}$$

such that φ contains the clause

$$(v_{t,w_1} = b_1 \vee v_{t,w_2} = b_2 \vee v_{t,w_3} = b_3)$$

if and only if $\psi_A(w_1, w_2, w_3, b_1, b_2, b_3) = 1$. Similarly, φ contains the clause

$$(v_{t,w_1} = b_1 \vee v_{t,w_2} = b_2 \vee v_{t+1,w_3} = b_3)$$

if and only if $\psi_B(w_1, w_2, w_3, b_1, b_2, b_3) = 1$.

Now, we claim that $\hat{\phi}$ can be evaluated efficiently in terms of the low-degree extensions $\hat{\psi}_A$ and $\hat{\psi}_B$, which themselves take only $\tilde{O}(W^3) = \text{poly}(\lambda)$ time. This follows from the following formula for $\hat{\phi}$, where each $i_j \in \mathbb{F}^m$ is parsed as a tuple $(t_j, w_j) \in \mathbb{F}^k \times \mathbb{F}^{m-k}$.

$$\begin{aligned}\hat{\phi}(i_1, i_2, i_3, b_1, b_2, b_3) = \delta(t_1, t_2, t_3) \cdot \hat{\psi}_A(w_1, w_2, w_3, b_1, b_2, b_3) \\ + s(t_1, t_2, t_3) \cdot \hat{\psi}_B(w_1, w_2, w_3, b_1, b_2, b_3).\end{aligned}$$

□

We will next combine this lemma with Theorem 4.19 to obtain an adaptive RAM delegation protocol.

4.5.2 RAM Delegation Protocol

First, we observe that without loss of generality we can assume a fixed polynomial poly such that the RAM machines delegated are of size at most $\text{poly}(\lambda)$. Otherwise, we can instead delegate a universal RAM machine, with a preceding sequence of delegated constant-sized computations that add the desired machine to the persistent database. For machines of this size, let $\ell(\lambda) = \max_i\{|V_i|\}$, where $\{V_i\}$ are the variables of the CNF given by Lemma 4.22. Let $(V_0, V_1, P_{\mathsf{PCP}})$ denote the PCP of Theorem 4.19 and let

$k_{max}(\lambda) = 2\ell(\lambda) \cdot \ell_0(\lambda) + \lambda^2$, where $\ell_0(\cdot)$ is defined in the statement of Theorem 4.19. We will use a succinct PIR scheme $(\mathsf{ScPIR.Send}, \mathsf{ScPIR.Respond}, \mathsf{ScPIR.Decode})$ as defined in Definition 2.13.

Construction 4.24. *The algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProcessDB}, \mathsf{Prove}, \mathsf{Verify})$ *are defined as below.*

Setup $\mathsf{Setup}(1^\lambda)$ *samples a collision-resistant hash function* $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ *and outputs* $\mathsf{pp} = h$.

Key Generation $\mathsf{KeyGen}(1^\lambda)$ *samples*

$$(Q, \mathsf{st}) \leftarrow V_0(1^\lambda),$$

and a random injection

$$\zeta : Q \hookrightarrow [k_{max}].$$

It then defines, for each $i \in [k_{max}]$,

$$(\tilde{q}_i, s_i) = \begin{cases} \mathsf{ScPIR.Send}(1^\lambda, q) & \text{if } \zeta(q) = i \text{ for some (unique) } q \in Q \\ \mathsf{ScPIR.Send}(1^\lambda, 0) & \text{otherwise} \end{cases}$$

KeyGen *outputs*

$$\mathsf{pk} = (\tilde{q}_1, \ldots, \tilde{q}_{k_{max}})$$

and

$$\mathsf{sk} = (\mathsf{st}, Q, \zeta, (s_1, \ldots, s_{k_{max}}))$$

Processing Database $\mathsf{ProcessDB}$ *is the same as in Lemma 4.22.*[8]

Proving $\mathsf{Prove}^{\mathsf{dt}}(\mathsf{pp}, \mathsf{pk}, M)$ *outputs* $(y, \mathsf{d}_{\mathsf{new}}, T, (\tilde{a}_1, \ldots, \tilde{a}_{k_{max}}))$ *after sampling*

$$\begin{aligned} &y \leftarrow M^{D \to D_{\mathsf{new}}} \\ &T = \mathsf{Time}(M, D) \\ &(\mathsf{d}', \mathsf{dt}') = \mathsf{ProcessDB}(D_{\mathsf{new}}) \\ &\varphi \leftarrow \mathsf{MakeCNF}(h, M, \mathsf{d}, y, \mathsf{d}', T) \\ &w \leftarrow \mathsf{Transcript}^{\mathsf{dt}}(1^T, M) \\ &\pi \leftarrow P_{\mathsf{PCP}}(1^\lambda, \varphi, w) \\ &\tilde{a}_i \leftarrow \mathsf{ScPIR.Respond}(\tilde{q}_i, \pi) \quad \text{for } i = 1, \ldots, k_{max} \end{aligned}$$

Verifying $\mathsf{Verify}(\mathsf{pp}, \mathsf{sk}, (M, \mathsf{d}, y, \mathsf{d}', T), \mathsf{pf} = (\tilde{a}_1, \ldots, \tilde{a}_{k_{max}}))$ *computes*

$$A : Q \to \Sigma$$

where for every $q \in Q$,

$$A(q) = \mathsf{ScPIR.Decode}(s_{\zeta(q)}, \tilde{a}_{\zeta(q)})$$

[8] For the curious reader, $\mathsf{ProcessDB}(D, h)$ simply computes the hash-tree of D with respect to h, and lets d be the root.

and outputs $V_1^{\hat{\phi}}(\mathsf{st}, A)$*, where* $\hat{\phi}$ *is the low-degree extension of the clause indicator function for the 3-CNF* $\varphi = \mathsf{MakeCNF}(h, M, \mathsf{d}, y, \mathsf{d}', T)$*. Queries to* $\hat{\phi}$ *are efficiently answerable by Lemma 4.23.*

Theorem 4.25. *Construction 4.24 is an adaptively secure non-interactive RAM delegation protocol.*

Proof. Completeness is easy to see; we therefore focus on proving soundness.

Suppose for contradiction that there is a poly-sized prover ensemble $\{P_\lambda^*\}_\lambda$ and a constant $c > 0$ such that for infinitely many λ (let Λ denote this set of λ),

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_0, \mathsf{d}_0', T), \mathsf{pf}_0) = 1 \ \wedge \\ \mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_1, \mathsf{d}_1', T), \mathsf{pf}_1) = 1\end{array}\right] > \lambda^{-c}$$

in the probability space defined by sampling

$$\begin{array}{l} h \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (M, \mathsf{d}, y_0, \mathsf{d}_0', \mathsf{pf}_0, y_1, \mathsf{d}_1', \mathsf{pf}_1, 1^T) \leftarrow P_\lambda^*(h, \mathsf{pk}) \end{array}$$

Let H_λ denote the set of h^* such that in the above probability space,

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_0, \mathsf{d}_0', T), \mathsf{pf}_0) = 1 \ \wedge \\ \mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_1, \mathsf{d}_1', T), \mathsf{pf}_1) = 1\end{array} \middle| h = h^*\right] > \frac{\lambda^{-c}}{2}.$$

By a simple probability argument, the above equations, together with the definition of H_λ, imply that for every $\lambda \in \Lambda$,

$$\Pr_{h \leftarrow \mathsf{Setup}(1^\lambda)}[h \in H_\lambda] \geq \frac{\lambda^{-c}}{2}$$

We define the ensemble $\{P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)\}_\lambda$ such that on input $Q \subseteq N$ (where N is the length of a proof), it does the following:

1. Pick a random injection $\zeta : Q \to [k_{max}]$ and, for each $i \in k_{max}$, define

$$(\tilde{q}_i, s_i) = \begin{cases} \mathsf{ScPIR.Send}(1^\lambda, q) & \text{if } \zeta(q) = i \text{ for some (unique) } q \in Q \\ \mathsf{ScPIR.Send}(1^\lambda, 0) & \text{otherwise} \end{cases}$$

2. Define a public-key $\mathsf{pk} = (\tilde{q}_1, \ldots, \tilde{q}_{k_{max}})$.

3. Compute $(M, \mathsf{d}, y_0, \mathsf{d}_0', \mathsf{pf}_0, y_1, \mathsf{d}_1', \mathsf{pf}_1, 1^T) \leftarrow P_\lambda^*(h, \mathsf{pk})$, where $\mathsf{pf}_b = (\tilde{a}_1^b, \ldots, \tilde{a}_{k_{max}}^b)$.

4. Decrypt $a_i^b \leftarrow \mathsf{ScPIR.Decode}(s_i, \tilde{a}_i^b)$ for each $b \in \{0, 1\}$ and $i \in \{1, \ldots, k_{max}\}$. Define

$$A_b : Q \to \Sigma$$

so that for every $q \in [Q]$,

$$A_b(q) = a_{\zeta(q)}^b$$

5. For $b \in \{0,1\}$, compute $\varphi_b \leftarrow \mathsf{MakeCNF}(h, M, \mathsf{d}, y_b, \mathsf{d}'_b, T)$.

6. Output $(\varphi_0, \varphi_1, A_0, A_1)$.

By definition, it follows immediately that

$$\Pr\left[V_1(\mathsf{st}, \varphi_0, A_0) = 1 \ \wedge \ V_1(\mathsf{st}, \varphi_1, A_1) = 1\right]$$

in the probability space defined by sampling

$$\begin{aligned}(Q, \mathsf{st}) &\leftarrow V_0(1^\lambda)\\ (\varphi_0, \varphi_1, A_0, A_1) &\leftarrow P^*_{\mathsf{PCP},\mathsf{h}}(Q)\end{aligned}$$

is the same as

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_0, \mathsf{d}'_0, T), \mathsf{pf}_0) = 1 \ \wedge\\ \mathsf{Verify}(h, \mathsf{sk}, (M, \mathsf{d}, y_1, \mathsf{d}'_1, T), \mathsf{pf}_1) = 1\end{array}\right]$$

in the probability space defined by sampling

$$\begin{aligned}(\mathsf{pk}, \mathsf{sk}) &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ (M, \mathsf{d}, y_0, \mathsf{d}'_0, \mathsf{pf}_0, y_1, \mathsf{d}'_1, \mathsf{pf}_1, 1^T) &\leftarrow P^*_\lambda(h, \mathsf{pk})\end{aligned}$$

If $h \in H_\lambda$, this probability is at least $\lambda^{-c}/2$. Furthermore, for each h, $P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)$ produces φ_0, φ_1 distributed as

$$\begin{aligned}(\mathsf{pk}, \mathsf{sk}) &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ (M, \mathsf{d}, y_0, \mathsf{d}'_0, y_1, \mathsf{d}'_1) &\leftarrow P^*_\lambda(h, \mathsf{pk})\\ \varphi_b &\leftarrow \mathsf{MakeCNF}(h, M, \mathsf{d}, y_b, \mathsf{d}'_b, T)\end{aligned}$$

Claim 4.25.1. *For all h, $\{P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)\}_\lambda$ is a k_{max}-computationally no-signaling ensemble.*

Proof. This follows from the security of the PIR scheme. Suppose otherwise – namely, suppose there exists $Q' \subseteq Q$ such that the distributions of

$$(\varphi_0, \varphi_1, (A_0)_{Q'}, (A_1)_{Q'}) \tag{4.11}$$

when sampling $(\varphi_0, \varphi_1, A_0, A_1) \leftarrow P^*_{\mathsf{PCP},\mathsf{h}}(Q)$ and of

$$(\varphi_0, \varphi_1, A_0, A_1) \tag{4.12}$$

when sampling $(\varphi_0, \varphi_1, A_0, A_1) \leftarrow P^*_{\mathsf{PCP},\mathsf{h}}(Q')$ are efficiently distinguishable.

By averaging, there exists some injection $\zeta : Q \to [k_{max}]$ such that using this ζ in the definition of $P^*_{\mathsf{PCP},\mathsf{h}}$ still yields distinguishable results. By a hybrid argument, we can assume without loss of generality that $|Q| = |Q'| + 1$. In particular, let q^* denote the sole element of $Q \setminus Q'$ and let i^* denote $\zeta(q^*)$. Now, given a PIR query $\tilde{q}^* = \mathsf{ScPIR.Send}(1^\lambda, q, N)$, we show how to distinguish the case when $q = q^*$ from the case when $q = 0$.

First, for all $i \in [k_{max}] \setminus \{i^*\}$, sample

$$(\tilde{q}_i, s_i) = \begin{cases} \mathsf{ScPIR.Send}(1^\lambda, q) & \text{if } i = \zeta(q) \text{ for some (unique) } q \in Q' \\ \mathsf{ScPIR.Send}(1^\lambda, 0) & \text{otherwise.} \end{cases}$$

Define $\tilde{q}_{i^*} = \tilde{q}^*$. Then, compute

$$(M, \mathsf{d}, y_0, \mathsf{d}'_0, \tilde{\mathbf{a}}^0, y_1, \mathsf{d}'_1, \tilde{\mathbf{a}}^1) \leftarrow P^*_\lambda(h, (T_\lambda, (\mathsf{pk}_1, \ldots, \mathsf{pk}_{k_{max}}), (\tilde{q}_1, \ldots, \tilde{q}_{k_{max}}))).$$

For each $i \in \zeta(Q')$, compute

$$a_i^b \leftarrow \mathsf{ScPIR.Decode}(s_i, \tilde{\mathbf{a}}_i^b).$$

From this, compute $(\varphi_0, \varphi_1, A_0|_{Q'}, A_1|_{Q'})$ as in $P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)$.

If q were q^*, then $(\varphi_0, \varphi_1, A_0|_{Q'}, A_1|_{Q'})$ would be distributed as in (4.11). If it were 0, then it would be distributed as in (4.12). By assumption, these distributions are efficiently distinguishable, so we have broken the security of the PIR scheme. □

Now, for $h \in H_\lambda$, we have a k_{max}-CNS PCP prover which with probability $\lambda^{-c}/2$ convinces the PCP verifier to accept two conflicting proofs. Thus, by Theorem 4.19, there is a PPT oracle algorithm Assign_c and a negligible function negl such that for all $\lambda \in \Lambda$ and all $h \in H_\lambda$, $\mathsf{Assign}_c^{P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)}$ is an adaptive $2\ell(\lambda)$-partial assignment generator. Furthermore, for any set of variables $V \subseteq [T'_\lambda]$ with $|V| \leq 2\ell(\lambda)$, when we sample

$$(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Assign}_c^{P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)}(1^\lambda, V),$$

we have that the distribution on (φ_0, φ_1) is indistinguishable from that obtained in the course of running $P^*_{\mathsf{PCP},\mathsf{h}}$, conditioned on $P^*_{\mathsf{PCP},\mathsf{h}}$ convincing the verifier. In particular, there are (with high probability) $(M, \mathsf{d}, y_0, \mathsf{d}'_0, y_1, \mathsf{d}'_1, T)$ such that:

For each $b \in \{0, 1\}$, φ_b is the formula

$$\varphi_{in}^{M,\mathsf{d}} \wedge \varphi^{M,T,h} \wedge \varphi_{out}^{y,\mathsf{d}'}.$$

We know that any locally consistent assignments to such pairs of formulas must agree on $V_0.q \cup V_0.\mathsf{d} \cup V_0.y$ and with non-negligible probability they must disagree on $V_T.q \cup V_T.\mathsf{d} \cup V_T.y$. Thus, there exists some i such that when querying

$$(\varphi_0, \varphi_1, A_0, A_1) \leftarrow \mathsf{Assign}_c^{P^*_{\mathsf{PCP},\mathsf{h}}(\cdot)}(1^\lambda, V_i \cup V_{i+1}),$$

A_0 and A_1 are two locally consistent assignments to $\varphi^{M,T_\lambda,h}$ such that A_0 and A_1 agree on $V_i.q \cup V_i.\mathsf{d} \cup V_i.y$ but disagree on $V_{i+1}.q \cup V_{i+1}.\mathsf{d} \cup V_{i+1}.y$.

Thus for $h \in H_\lambda$, with non-negligible probability, $\mathsf{FindCollision}(1^T, \varphi^{M,T,h}, A_0, A_1)$ outputs a collision in h. Since $h \in H_\lambda$ with non-negligible probability, this violates the assumption that h is sampled from a collision-resistant hash family. □

4.6 Batch Arguments of Knowledge for NP

In this section, we consider *batch arguments of knowledge* for NP languages, in which a prover wants to prove that each of $x_1, \ldots, x_k$ is in $\mathcal{L}$. We will show a 2-message protocol where the communication complexity and the running time of the verifier are both $m \cdot \text{poly}(\lambda)$, where m is the size of a witness that a single x_i is in $\mathcal{L}$, and as previously, λ is the security parameter.

We emphasize that we only prove non-adaptive soundness of this 2-message protocol, as opposed to RAM delegation where we proved adaptive soundness and hence got a non-interactive delegation scheme. Intuitively, the reason for this discrepancy is that in the case of NP, a polynomial size distinguisher cannot distinguish between $x_i \in \mathcal{L}$ and $x_i \notin \mathcal{L}$, whereas in the case of polynomial-time RAM computations, a polynomial time distinguisher can do the computation on its own. We elaborate on this later.

Definition 4.26 (BARKs). *A (non-adaptive, 1-message) Batch Argument of Knowledge (BARK) for an* NP *language $\mathcal{L}$ given by the relation $R_{\mathcal{L}}$ is a tuple of algorithms* (KeyGen, V, P) *satisfying the following properties:*

Completeness *If $x_1, \ldots, x_k$ and $w_1, \ldots, w_k$ are such that $R_{\mathcal{L}}(x_i, w_i) = 1$ for every $i \in [k]$, then it holds that*

$$\Pr\left[V(\mathsf{sk}, (x_1, \ldots, x_k), \pi) = 1\right] = 1$$

in the probability space defined by sampling

$$\begin{aligned}(\mathsf{pk}, \mathsf{sk}) &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ \pi &\leftarrow P(\mathsf{pk}, (x_1, \ldots, x_k), (w_1, \ldots, w_k))\end{aligned}$$

(Non-Adaptive) Proof of Knowledge *There exists an algorithm E such that for all poly-sized P^*_λ and all $x_1, \ldots, x_k$, if*

$$\Pr\left[V(\mathsf{sk}, (x_1, \ldots, x_k), \pi) = 1\right] = \epsilon \geq \frac{1}{\text{poly}(\lambda)}$$

in the probability space defined by sampling

$$\begin{aligned}(\mathsf{pk}, \mathsf{sk}) &\leftarrow \mathsf{KeyGen}(1^\lambda)\\ \pi &\leftarrow P^*_\lambda(\mathsf{pk})\end{aligned}$$

then $E(P^*_\lambda, (x_1, \ldots, x_k))$ outputs, with high probability, witnesses $(w_1, \ldots, w_k)$ in time $\text{poly}(|P^*_\lambda|, |x_1| + \ldots + |x_k|, \frac{1}{\epsilon})$ *such that for each i, $R_{\mathcal{L}}(x_i, w_i) = 1$.*

Remark 4.27. In our construction, the knowledge extractor uses P^*_λ as a black-box, and runs in time $\text{poly}(|x_1| + \ldots + |x_k|, \frac{1}{\epsilon})$. For the sake of generality, the definition above allows for non-black extractors.

Theorem 4.28. *Let $\mathcal{L} \in \mathsf{NP}$ be any language defined by a relation $R_{\mathcal{L}}$ such that $x \in \mathcal{L}$ if and only if there is a witness w such that $R_{\mathcal{L}}(x, w) = 1$. If a succinct PIR scheme exists as in Definition 2.13, then there is a BARK for $\mathcal{L}$ with the following efficiency:*

- *The length of a proof π when sampling*

$$\pi \leftarrow P(q, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$$

  is $(\max_i |w_i|) \cdot \mathrm{poly}(\lambda)$. Furthermore the length of q is also $(\max_i |w_i|) \cdot \mathrm{poly}(\lambda)$.

- *The total running time of (V_0, V_1) is $(\sum_i |x_i| + \max_i |w_i|) \cdot \mathrm{poly}(\lambda)$.*

Remark 4.29. If the verifier is given (or has pre-computed) the digests of each x_i, the verifier can actually run in time $(k + \max_i |w_i|) \cdot \mathrm{poly}(\lambda)$.

Remark 4.30. From now on, we write $f = \tilde{O}(g)$ if there is a polynomial poly such that $f(\lambda) \leq g(\lambda) \cdot \mathrm{poly}(\lambda)$, departing from the usual convention that $\tilde{O}$ hides logarithmic factors.

Proof Overview. Let us suppose for simplicity that all the x_i's have the same length n, and all w_i's have the same length m.

Recall that Theorem 4.19[9], together with a succinct PIR (as in Theorem 4.25), allows one to prove for any ℓ and any satisfiable 3-CNF φ, that φ is ℓ-locally satisfiable via a 1-message (adaptively sound) protocol with communication complexity $\tilde{O}(\ell)$, where the running time of the verifier is $\tilde{O}(\ell + T_{\hat{\phi}})$, where $T_{\hat{\phi}}$ is the time it takes to evaluate the low-degree extension of ϕ, where ϕ is the clause indicator function of φ. We denote by $\hat{\phi}$ the low-degree extension of ϕ.

It therefore suffices to define a 3-CNF $\varphi_{x_1,\ldots,x_k}$ such that:

- $\varphi_{x_1,\ldots,x_k}$ is $\tilde{O}(m)$-locally satisfiable if and only if $x_i \in \mathcal{L}$ for every i. Furthermore, for any i, a witness that $x_i \in \mathcal{L}$ can be efficiently extracted from any such partial assignment generator.

- $\hat{\phi}_{x_1,\ldots,x_k}$ is computable in time $\tilde{O}(kn + m)$.

First we construct φ_x that is satisfiable if and only if $x \in \mathcal{L}$, for a single x, and we ensure that given a $\tilde{O}(m)$-partial assignment generator for φ_x, one can efficiently extract a witness w such that $R_{\mathcal{L}}(x, w) = 1$. Once we have done this, we build $\varphi_{x_1,\ldots,x_k} = \bigwedge_i \varphi_{x_i}$, and give each φ_{x_i} a disjoint set of variables. One point that we must carefully address is that the LDE of the clause indicator function of $\varphi_{x_1,\ldots,x_k}$ needs to be efficiently computable (i.e. in time $\tilde{O}(kn + m)$). This is done in Lemma 4.33.

We first show the existence of such a φ_x when $m = \Omega(n)$. To this end, consider the RAM machine M which operates on a database whose initial contents are $x \| w$, and computes $R_{\mathcal{L}}(x, w)$. Let T be a bound on the running time of $R_{\mathcal{L}}$, let $h : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ be a hash function, and let d be the digest of $x \| w$ with respect to the hash

[9] In fact, a weaker variant of Theorem 4.19 without adaptivity suffices.

function h. Lemma 4.22 constructs a 3-CNF $\varphi \triangleq \varphi_{in}^{M,\mathsf{d}} \wedge \varphi^{M,T,h} \wedge \varphi_{out}$ of a particular structure,[10] so that (we show this "local-to-global" argument in Theorem 4.25) given a poly(λ)-partial assignment generator Assign for φ and a database $D \in \mathsf{Digest}^{-1}(\mathsf{d})$, either $M^D \to 1$ or one can efficiently find a collision in h. Importantly, this holds even if the database D is chosen adaptively as a (no-signaling) function of the queries that Assign receives. A stronger version of this "local-to-global" claim is part of Theorem 4.25.

We define φ_x as the 3-CNF consisting of $\varphi'_{in} \wedge \varphi^{M,T,h} \wedge \varphi_{out}$, where $\varphi^{M,T,h}$ and φ_{out} are the 3-CNFs given by Lemma 4.22 as above, and where φ'_{in} is a 3-CNF on $v = \tilde{O}(n+m)$ new variables, as well as the variables $V_0.\mathsf{d}$ from $\varphi^{M,T,h}$. The first $n+m$ variables, denoted by V_D, supposedly contain $x\|w$, and the other $v-(n+m)$ variables are auxiliary variables, denoted by V_{aux}. The clauses of φ'_{in} ensure that $V_0.\mathsf{d} = \mathsf{Digest}(V_D)$, and ensure that the first n variables of V_D are x.

Now suppose we are given a $\tilde{O}(n)$-partial assignment generator $\mathcal{G}$ for φ_x, and imagine querying $V_D \cup V_{aux} \cup V_0.\mathsf{d} \cup Q$ for any Q with $|Q| \leq \text{poly}(\lambda)$. We can view this as an adaptive poly(λ)-partial assignment generator for $\varphi^{M,T,h}$ on queries Q, which by local satisifiability, must sastify the following:

- Define D by the answers on V_D. Then D must be of the form $x\|w$.
- The answers on V_{d} must be equal to $\mathsf{Digest}(x\|w)$.
- The answers on Q must locally satisfy $\varphi_{M,\mathsf{Digest}(x\|w),h}$.

By the "local-to-global" argument of Theorem 2 referenced above, it means that $M^D \to 1$. Furthermore, the answers on V_D directly give us a witness to this fact, so extractability is straight-forward.

Note that above the locality of the partial assignment generator is $\tilde{O}(m+n)$. Note that if $m = o(n)$ then we do not get the efficiency we desire, of locality $\tilde{O}(m)$. To obtain our desired efficiency, we think of M as a RAM with two databases, one which initially holds x and the other which initially holds w. The point here is that instead of computing the digest of $x\|w$, φ_x will have the digest of x (computed by the verifier) hard-coded, and will only compute the digest of w. As a result we will need a $\tilde{O}(m)$-partial assignment generator $\mathcal{G}$ for φ_x, in order to extract w, as desired.

We proceed with a formal proof of Theorem 4.28.

Proof. Let $T(\cdot)$ and $m(\cdot)$ be polynomials such that there is a Turing machine M which computes $R_{\mathcal{L}}(x,w)$ in time $T(|x|)$, provided that $|w| \leq m(|x|)$. Without loss of generality, we can think of M as a two-tape machine, where the first tape initially holds x and the second tape initially holds w.

We begin with the following lemma, which sums up the required modifications to Lemma 4.22.

[10] In Lemma 4.22, the 3-CNF $\varphi_{out} = \varphi_{out}^{y,d'}$ checks that the output is indeed y and checks that the digest of the database at the end of the computation is d'. In our setting, we do not care about d', and we check that the output of the computation is 1, therefore we omit the notation of y, d' from φ_{out}.

Lemma 4.31. *There exist deterministic polynomial-time algorithms*

$$\mathsf{MakeCNF}, \mathsf{Transcript}, \mathsf{FindCollision},$$

such that for any hash function $h : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$*, any input* x *with digest* d_x*,* $\mathsf{MakeCNF}(h, x)$ *is a 3-CNF* φ *with the following structure: Let* $T = T(|x|)$ *and* $m = m(|x|)$*. Then,* $\varphi \triangleq \varphi_{in}^{M,\mathsf{d}_x} \wedge \varphi_{wit}^{h,m} \wedge \varphi_{main}^{M,T,h} \wedge \varphi_{out}$*. We will now define each component.*

1. *φ has $T' = \tilde{O}(T \cdot |M| + m)$ variables. The set of variables V contains $T+1$ disjoint sets $V_0, \ldots, V_T$ each of size $\tilde{O}(|M|)$, as well as a set V_{wit} with $\tilde{O}(m)$ variables. Each set V_i (for $i = 1, \ldots, T$) contains (in addition to $\tilde{O}(m)$ auxiliary variables):*

   - *Variables $V_i.q$ whose values encode the Turing machine's local state after i execution steps.*
   - *Variables $V_i.\mathsf{d}_1$ and variables $V_i.\mathsf{d}_2$ whose values are the digests of the two tapes after i execution steps. The first tape initially holds x and the second tape initially holds w.*
   - *Variables $V_i.y$ whose values encode the output of M if it has terminated by the i^{th} step, and encode $\perp$ otherwise.*

   V_{wit} has m variables $V_{wit}.w$ whose values encode a witness that $x \in \mathcal{L}$. V_{wit} also has auxiliary variables $V_{wit}.aux$.

2. *$\varphi_{in}^{M,\mathsf{d}_x}$ has clauses on V_0, which enforce that $V_0.q = M.q_0$, $V_0.\mathsf{d}_1 = \mathsf{d}_x$, and $V_0.y = \perp$.*

3. *$\varphi_{wit}^{h,m}$ has $\tilde{O}(m)$ clauses on $V_{wit} \cup V_0$ which enforce that $V_0.\mathsf{d}_2 = \mathsf{Digest}(h, V_{wit}.w)$,*

4. *$\varphi_{main}^{M,T,h}$ has clauses only within each V_i and between V_i and V_{i+1}. These clauses are identically repeated between every pair of (V_i, V_{i+1}).*

5. *φ_{out} has only one clause on V_T, which enforces that $V_T.y = 1$.*

Furthermore:

1. *If $R_{\mathcal{L}}(x, w) = 1$ then $\mathsf{Transcript}(x, w)$ outputs a satisfying assignment A for φ in time $T \cdot |M| \cdot \mathrm{poly}(\lambda)$ such that $A(V_{wit}.w) = w$. If $R_{\mathcal{L}}(x, w) = 0$, then $\mathsf{Transcript}(x, w)$ outputs an assignment which satisfies all clauses except for the one in φ_{out} (that is, $V_T.y = 0$).*

2. *For any two locally-consistent assignments A and A' for φ, both on $V_i \cup V_{i+1}$, such that:*

   - *$A|_{V_i.q \cup V_i.\mathsf{d}_1 \cup V_i.\mathsf{d}_2 \cup V_i.y} \equiv A'|_{V_i.q \cup V_i.\mathsf{d} \cup V_i.y}$ (i.e. are equal as functions) but*
   - *$A|_{V_{i+1}.q \cup V_{i+1}.\mathsf{d}_1 \cup V_{i+1}.\mathsf{d}_2 \cup V_{i+1}.y} \not\equiv A'|_{V_{i+1}.q \cup V_{i+1}.\mathsf{d} \cup V_{i+1}.y}$*

it holds that $\mathsf{FindCollision}(1^T, \varphi, A, A')$ *outputs* (z, z') *such that* $z \neq z'$ *and* $h(z) = h(z')$ *– in other words, a collision under* h*.*

Proof. (Sketch) An assignment to any layer V_i contains the two digests corresponding to the two work tapes of the Turing machine M at computation step i (where initially the first work tape contains x and the second work tape contains a corresponding witness w), it contains the memory operations (i.e. reads or writes) that are performed by M on the i^{th} step, as well as the alleged results of these operations (i.e. the value read from memory or the value written to memory). This is used to compute the internal state of M on the $i+1^{th}$ step, which in turn is used to compute the updated digests of the two work tapes in step $i+1$. The constraints on V_i use cryptographic machinery (based on Merkle trees) to ensure that the claimed memory operation results are consistent with the previous memory digests.

□

Claim 4.31.1. *Fix any* x *and sample a hash function* h *from a collision-resistant hash family. Let* φ *be the 3-CNF as in Lemma 4.31, corresponding to* x *and* h*. Given a (non-adaptive)* $\tilde{O}(m)$*-partial assignment generator* Assign *for* φ*, one can efficiently obtain a witness* w *such that* $R_{\mathcal{L}}(x, w) = 1$ *with overwhelming probability.*

Proof. We will show that querying $\mathsf{Assign}(V_{wit}.w)$ yields the desired witness with overwhelming probability. We know from local consistency and the definition of $\varphi_{in}^{M,\mathsf{d}_x}$ and $\varphi_{wit}^{h,m}$ that if we query $A \leftarrow \mathsf{Assign}(V_{wit} \cup V_0)$ and then compute $A' \leftarrow \mathsf{Transcript}(x, A(V_{wit}.w))$, then, as functions,

$$A|_{V_0.q \cup V_0.\mathsf{d}_1 \cup V_0.\mathsf{d}_2 \cup V_0.y} \equiv A'|_{V_0.q \cup V_0.\mathsf{d}_1 \cup V_0.\mathsf{d}_2 \cup V_0.y}$$

with overwhelming probability (and both are equal to $(q_0, d_x, d_w, \perp)$ with overwhelming probability, where d_w is the digest of $A(V_{wit}.w)$).

Similarly, by local consistency, and by the definition of φ_{out}, if we query $A \leftarrow \mathsf{Assign}(V_{wit} \cup V_T)$, then $A(V_T.y) = 1$ with overwhelming probability.

We next show that with overwhelming probability $A'(V_T.y) = 1$ where $A' \leftarrow \mathsf{Transcript}(x, A(V_{wit}.w))$, which implies that $R_{\mathcal{L}}(x, V_{wit}.w) = 1$. Suppose otherwise, i.e. that with non-negligible probability $A'(V_T.y) \neq 1$. Then there is some i such that when querying $A \leftarrow \mathsf{Assign}(V_{wit} \cup V_i \cup V_{i+1})$ and $A' \leftarrow \mathsf{Transcript}(x, A(V_{wit}.w))$, it holds with non-negligible probability that

$$A|_{V_i.q \cup V_i.\mathsf{d}_1 \cup V_i.\mathsf{d}_2 \cup V_i.y} \equiv A'|_{V_i.q \cup V_i.\mathsf{d}_1 \cup V_i.\mathsf{d}_2 \cup V_i.y}$$

but

$$A|_{V_{i+1}.q \cup V_{i+1}.\mathsf{d}_1 \cup V_{i+1}.\mathsf{d}_2 \cup V_{i+1}.y} \not\equiv A'|_{V_{i+1}.q \cup V_{i+1}.\mathsf{d}_1 \cup V_{i+1}.\mathsf{d}_1 \cup V_{i+1}.y},$$

where again $\equiv$ denotes equality as functions. This implies that we can efficiently find a collision in h by using $\mathsf{FindCollision}$, which is a contradiction.

In this argument, we only ever needed to query Assign at $\tilde{O}(m)$ variables – specifically, we queried the number of variables in V_{wit} plus the number of variables in any V_i. Hence, it is only required that Assign is a $\tilde{O}(m)$-partial assignment generator. □

Corollary 4.32. *For any $x_1, \ldots, x_k$, let $\varphi_1, \ldots, \varphi_k$ be as in Lemma 4.31. Given an $\tilde{O}(m)$-partial assignment generator for $\varphi = \bigwedge_i \varphi_i$, one can efficiently find witnesses $w_1, \ldots, w_k$ so that for each i, $R_{\mathcal{L}}(x_i, w_i) = 1$.*

Proof. This follows from the fact that an $\tilde{O}(m)$-partial assignment generator for $\varphi = \bigwedge_i \varphi_i$ is an $\tilde{O}(m)$-partial assignment generator for each φ_i. □

Lemma 4.33. *Let $x_1, \ldots, x_k \in \{0,1\}^n$ be any strings. Let $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ be any hash function. For each $i \in \{1, \ldots, k\}$, define*

$$\varphi_i = \mathsf{MakeCNF}(h, x_i).$$

Let V be the variables of $\varphi = \bigwedge_i \varphi_i$.

For any finite field $\mathbb{F}$ with $|\mathbb{F}| > \log|V|$, and any subset $H \subseteq \mathbb{F}$ with $|H| = \lceil \log|V| \rceil$, let $m = \lceil \log|V| / \log|H| \rceil$. There is an efficiently computable and invertible mapping $e : V \hookrightarrow H^m$ such that the low-degree extension

$$\hat{\phi} : \mathbb{F}^{3m+3} \to \mathbb{F}$$

of the clause-indicator function

$$\phi : H^{3m+3} \to \{0,1\}$$

$$\phi(i_1, i_2, i_3, b_1, b_2, b_3) = \begin{cases} 1 & \text{if } \varphi \text{ contains the clause } \bigvee_{j=1}^{3} (e^{-1}(i_j) = b_j) \\ 0 & \text{otherwise} \end{cases}$$

is efficiently computable at any point $z \in \mathbb{F}^{3m+3}$ in time $\tilde{O}(kn + m)$.

Proof. The following claims follow directly from Theorem 4.8.

Claim 4.33.1. *Associate H^m with $[|H|^m]$ by lexicographically extending an arbitrary correspondence of H with $[|H|]$. Then, for any $k \in [|H|^m]$, we can compute the low-degree extension of*

$$\mathsf{eq}^{\geq k} : H^{3m} \to \{0,1\}$$

$$\mathsf{eq}^{\geq k}(x, y, z) = \begin{cases} 1 & \text{if } x = y = z \geq k \\ 0 & \text{otherwise} \end{cases}$$

in $\mathrm{poly}(|H|, m)$ *time (assuming field operations have unit cost).*

Claim 4.33.2. *Associate H^m with $[|H|^m]$ by lexicographically extending an arbitrary correspondence of H with $[|H|]$. Then for any $k \in [|H|^m]$, we can compute the low-degree extension of*

$$s^{\geq k} : H^{3m} \to \{0,1\}$$

$$s^{\geq k}(x, y, z) = \begin{cases} 1 & \text{if } x = y = z - 1 \geq k \\ 0 & \text{otherwise} \end{cases}$$

in $\mathrm{poly}(|H|, m)$ *time (assuming field operations have unit cost).*

Armed with the ability to compute low-degree extensions of $\mathsf{eq}^{\geq k}$ and $s^{\geq k}$, we can now write the formula for $\hat{\phi} : \mathbb{F}^{3m+3} \to \mathbb{F}$.

Let $a = \lceil \log_{|H|} k \rceil$, let $b = \lceil \log_{|H|} T \rceil$, and let $\ell = \lceil \log_{|H|} |V_{wit}| \rceil$. We will define e to map the variables of each φ_i into $\{i\} \times H^{m-a}$, where $i \in [k]$ is interpreted in H^a as number in base $|H|$. In particular, the w^{th} variable of layer V_j in φ_i is mapped to $(i, w, j + \ell) \in H^a \times H^{m-a-b} \times H^b$. The variables of V_{wit} are mapped to

$$H^a \times H^{m-a-b} \times \{t \in H^b : t < \ell\}$$

in an arbitrary way.

Recall from the statement of Lemma 4.31 that the clauses of each φ_i are highly repetitive. In particular, there are "small" clause-indicator functions ψ_A and ψ_B such that for any $i \in [k]$ and any $t \geq \ell$, φ_i contains the clause

$$e^{-1}((i, j_1, t)) = b_1 \vee e^{-1}((i, j_2, t)) = b_2 \vee e^{-1}((i, j_3, t)) = b_3$$

iff $\psi_A(j_1, j_2, j_3, b_1, b_2, b_3) = 1$. Also, φ_i contains the clause

$$e^{-1}((i, j_1, t)) = b_1 \vee e^{-1}((i, j_2, t)) = b_2 \vee e^{-1}((i, j_3, t + 1)) = b_3$$

iff $\psi_B(j_1, j_2, j_3, b_1, b_2, b_3) = 1$.

Because φ_{wit} has $\tilde{O}(m)$ (efficiently enumerable) clauses, its low-degree extension $\hat{\phi}_{wit}$ can be computed in $\tilde{O}(m)$ time. $\varphi_{in}^{M,\mathbf{d}_{x_i}}$ has $\mathrm{poly}(\lambda)$ constraints which take $\tilde{O}(|x_i|)$ time to enumerate, and each of $\tilde{\psi}_A$ and $\tilde{\psi}_B$ is computable in time $\mathrm{poly}(\lambda)$. Thus, Lemma 4.33 follows from the following formula, where each z_j is interpreted as a tuple $(i_j, w_j, t_j) \in \mathbb{F}^a \times \mathbb{F}^{m-a-b} \times \mathbb{F}^b$.

$$\hat{\phi}(z_1, z_2, z_3, b_1, b_2, b_3) = \delta(i_1, i_2, i_3) \cdot \left(\begin{array}{l} \mathsf{eq}^{\geq t}(t_1, t_2, t_3)\hat{\psi}_A(j_1, j_2, j_3, b_1, b_2, b_3) + \\ s^{\geq t}(t_1, t_2, t_3)\hat{\psi}_B(j_1, j_2, j_3, b_1, b_2, b_3) + \\ \hat{\phi}_{wit}^{h,m}(w_1, t_1, w_2, t_2, w_3, t_3, b_1, b_3) \end{array} \right) + \sum_i \hat{\phi}_{in}^{M,\mathbf{d}_{x_i}}(z_1, z_2, z_3, b_1, b_2, b_3).$$

□

Now that we have shown how to efficiently compute the low-degree extension of ϕ, the construction and proof of Theorem 4.28 proceeds similarly to the proof of Theorem 4.25. We can now describe our BARK scheme. Let M be the Turing machine such that $x \in \mathcal{L} \cap \{0,1\}^n$ if and only if there is a witness $w \in \{0,1\}^{m(n)}$ such that $M(x, w) = 1$ in time $T(n)$. Let $(V_{\mathsf{PCP}}^0, V_{\mathsf{PCP}}^1, P_{\mathsf{PCP}})$ be the PCP of Theorem 4.19 and let

$$(\mathsf{ScPIR.Send}, \mathsf{ScPIR.Respond}, \mathsf{ScPIR.Decode})$$

be a succinct PIR scheme as defined in Definition 2.13.

Verifier's Message $V_0(1^\lambda, (x_1, \ldots, x_k))$ first computes

$$(Q, \mathsf{st}) \leftarrow V_{\mathsf{PCP}}^0(1^\lambda).$$

Let m be the maximum length of any witness corresponding to an x_i, and let m' be the locality in the statement of Claim 4.31.1. Let T be the corresponding running time of M. Let $k_{max} = m' \cdot \ell_0(\lambda) + \lambda^2$, where $\ell_0(\cdot)$ is defined as in the statement of Theorem 4.19.

V_0 samples a random injection

$$\zeta : Q \hookrightarrow [k_{max}].$$

It then defines, for each $i \in [k_{max}]$,

$$(\tilde{q}_i, s_i) = \begin{cases} \mathsf{ScPIR.Send}(1^\lambda, q) & \text{if } \zeta(q) = i \text{ for some (unique) } q \in Q \\ \mathsf{ScPIR.Send}(1^\lambda, 0) & \text{otherwise} \end{cases}$$

and sends $(\tilde{q}_1, \ldots, \tilde{q}_{m'})$ to the prover.

Prover's Message $P((h, \tilde{q}_1, \ldots, \tilde{q}_{m'}), (x_1, \ldots, x_k), (w_1, \ldots, w_k))$ uses $(w_1, \ldots, w_k)$ to compute a satisfying assignment w to the 3-CNF $\varphi = \bigwedge_i \varphi_i$, where $\varphi_i \leftarrow \mathsf{MakeCNF}(h, M, x_i, m(|x_i|), T(|x_i|))$. It generates $\pi \leftarrow P_{\mathsf{PCP}}(1^\lambda, \varphi, w)$ and computes $\tilde{a}_i \leftarrow \mathsf{ScPIR.Respond}(\tilde{q}_i, \pi)$ for $i = 1, \ldots, m'$.

The prover sends $(\tilde{a}_1, \ldots, \tilde{a}'_m)$ to the verifier.

Verifier Checks The verifier then computes $A : Q \to \Sigma$ where for every $q \in Q$,

$$A(q) = \mathsf{ScPIR.Decode}(s_{\zeta(q)}, \tilde{a}_{\zeta(q)})$$

and outputs $(V_{\mathsf{PCP}}^1)^{\hat{\phi}}(\mathsf{st}, A)$, where $\hat{\phi}$ is the low-degree extension of $\bigwedge_i \varphi_i$, and queries to $\hat{\phi}$ are efficiently answerable by Lemma 4.33.

Extraction First, the security of the PIR implies that any BARK prover P^* for which

$$\Pr[V_1(\mathsf{st}, \pi) = 1] \geq \epsilon(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned} (q, \mathsf{st}) &\leftarrow V_0(1^\lambda, x_1, \ldots, x_k) \\ \pi &\leftarrow P^*(q) \end{aligned}$$

can be turned into a k_{max}-wise CNS PCP prover P^*_{PCP} such that

$$\Pr\left[V_{\mathsf{PCP}}^1(\mathsf{st}, \varphi, A) = 1\right] \geq \epsilon(\lambda)$$

in the probability space defined by sampling

$$(Q, \mathsf{st}) \leftarrow V^0_{\mathsf{PCP}}(1^\lambda)$$
$$A \leftarrow P^*_{\mathsf{PCP}}(Q),$$

where $\varphi = \bigwedge_i \varphi_i$ and $\varphi_i = \mathsf{MakeCNF}(h, M, x_i, m, T)$. By Theorem 4.19, any such P^*_{PCP} can be turned into an $\tilde{O}(m)$-partial assignment generator for φ. By Corollary 4.32, this can be used, for each i, to extract a witness that $x_i \in \mathcal{L}$.

□

4.6.1 Barrier to Adaptively Sound BARKs

We argue that the non-adaptivity of the above scheme is inherent – namely, it is not possible to prove adaptive soundness of any BARK with a black box reduction to any falsifiable assumption. To show this, we sketch a black-box construction of a SNARK from any adaptively sound BARK. We then rely on the beautiful separation result of Gentry and Wichs [GW11], which shows that SNARKs cannot have a black-box reduction to falsifiable assumptions.

Our first observation is that a BARK allows one to succinctly prove that a Merkle digest d is honestly generated, i.e. prove knowledge of $D \in \{0,1\}^n$ such that $\mathsf{d} = \mathsf{Digest}(D)$. The prover just needs to prove knowledge of an opening of $D[i]$ that is consistent with d, for each $i \in$ Âă$\{1, \ldots, n\}$. But simultaneously proving knowledge of n different witnesses each of length $\tilde{O}(\log n)$ is something a BARK can accomplish with a proof of length $\tilde{O}(\log n)$.

Building on this observation, we build a designated-verifier SNARK as follows. The SNARK public key consists of a collision-resistant hash function h, as well as a public key pk_{RAM} for our adaptive RAM delegation protocol. A proof for x consists of a digest d under h, a proof of knowledge as described above of a witness $w \in \{0,1\}^n$ such that $\mathsf{d} = \mathsf{Digest}(w)$, and a proof (using the RAM protocol) that the NP relation accepts (x, w).

Chapter 5

Memory Delegation

This chapter is based on the TCC 2016 paper "Adaptive Succinct Garbled RAM, or: How to Delegate Your Database" by Canetti, Chen, Holmgren, and Raykova [CCHR16].

The starting point of our construction is the garbling scheme of Chapter 3. We briefly recap that construction, and then explain where the issues with adaptivity come up and how we solve them.

Statically Secure Garbling Scheme for RAMs - an overview. Our Chapter 3 construction consisted of four main steps. We started with a garbling scheme satisfying *same transcript indistinguishability*, i.e. a garbling scheme which guarantees indistinguishability of the garbled machines and inputs as long as the entire transcripts of the communication with the external memory, as well as the local states kept between the RAM computation steps are the same in the two computations. In other words, if the computation of machine M_1 on input x_1 has the same transcript as that of M_2 on input x_2, then the garbled machines $\tilde{M}_1$, $\tilde{M}_2$ and the garbled inputs $\tilde{x}_1$, $\tilde{x}_2$ are computationally indistinguishable: $(\tilde{M}_1, \tilde{x}_1) \approx (\tilde{M}_2, \tilde{x}_2)$.

The construction of this weak garbling scheme closely follows the scheme of Koppula, Lewko and Waters [KLW15] for garbling Turing machines. The garbled program is essentially an obfuscated CPU circuit, which takes as input a local state and a memory symbol, and outputs an updated local state, as well as a memory operation. The main challenge here is to guarantee the authenticity and freshness of the values read from the memory. This is done using a number of mechanisms, namely splittable signatures, iterators and positional accumulators.

The next step extends the construction to a garbling scheme satisfying *same address indistinguishability*, namely a scheme that guarantees indistinguishability of the garbled machines as long as the sequence of *memory addresses* accessed is the same in the two computations. To obtain this security property, we modified the obfuscated CPU circuit so that it operates on an encrypted local state and encrypted memory contents.

The final step is to use an obfuscation-friendly ORAM in order to hide the program's memory access pattern. Specifically, we use the ORAM of Chung and Pass [CP13].

The challenge of adaptive security. In obtaining adaptive security, we are able to follow the same basic approach to bootstrapping security. It turns out that the biggest

challenge is the first step – obtaining an adaptively secure variant of same transcript indistinguishability. The challenge involves the *positional accumulator*, which is an iO-friendly variant of a Merkle hash tree. This is used to hash the contents of memory hashed down to a short root (called the accumulator value ac). This value is then signed by the CPU circuit, together with its current local state, and is kept (in memory) for subsequent verification of memory accesses. Using the accumulator, the evaluator is later able to efficiently convince the CPU circuit of the true contents of an arbitrary memory location. We call this operation "opening" the accumulator value ac. Intuitively, the main security property is that it should be computationally infeasible to open an accumulator to an incorrect value.

To be useful with indistinguishability obfuscation, the accumulator needs an additional property, called *enforceability*. In [KLW15], this property allows to generate, given memory location L^* and symbol v^*, a "rigged" public key for the accumulator along with a "rigged" accumulator value ac^*. The rigged public key and accumulator look indistinguishable from honestly generated public key and accumulator value, and also have the property that *there does not exist* a way to open ac^* at location L^* to any value other than v^*.[1]

The fact that the special values v^*, L^*, and ac^* are encoded in the rigged public key forces these values to be known before the adversary sees the public key. This suffices for the case of static garbling, since the special values depend only on the underlying computation, and this computation is fixed in advance and does not depend on adversary's view. However, in the adaptive setting, this is not the case. This is so since the adversary can choose new computations — and thus new special values v^*, L^* — depending on its view so far, which includes the public key of the accumulator.

Adaptive Accumulators. We get around this problem by defining and constructing a new primitive, called *adaptive accumulators,* which are an adaptive alternative to positional accumulators. In our adaptive accumulators there are no "rigged" public keys. Instead, correctness of an opening of a hash value at some location is verified using a *verification key* which can be generated later. In addition to the usual computational binding guarantees, it should be possible to generate, given a special accumulator value ac^*, value v^* and location L^*, a "rigged" verification key vk^* that looks indistinguishable from an honestly generated one, and such that vk^* does not verify an opening of ac^* at location L^* to any value other than v^*. Furthermore, it is possible to generate multiple verification keys, that are all rigged to enforce the same

[1] To get an idea of why enforceability is needed, consider two programs C_0 and C_1, such that $C_0(L^*, v^*) = C_1(L^*, v^*)$, but whose functionality may differ elsewhere, and let $C'_b(L, v)$ be the program "if L, v are consistent with ac^* then run C_b, else output $\perp$". Let $i\mathcal{O}$ be an indistinguishability obfuscator, i.e. it is guaranteed that $i\mathcal{O}(A) \approx i\mathcal{O}(B)$ whenever A and B are equal sized programs that have the same functionality everywhere. We would like to argue that $i\mathcal{O}(C'_0) \approx i\mathcal{O}(C'_1)$; however, we cannot do it directly using a plain Merkle hash tree, since collisions exist and so C'_0 and C'_1 have very different functionalities. Positional accumulators get around this difficulty: Using enforceability it is possible to argue that, when C'_0 and C'_1 use the rigged public key for the accumulator, the two programs have exactly the same functionality, and so indistinguishability holds. Due to the indistinguishability of rigged public accumulator keys from honest ones, indistinguishability holds even for the case of non-rigged accumulator keys.

accumulator value ac^* to different values v^* at different locations L^*, where all are indistinguishable from honest verification keys.

We then use adaptive accumulators as follows: There is a single set of public parameters that is posted together with the garbled database and is used throughout the lifetime of the system. Now, each new garbled machine is given a different, independently generated verification key. This allows us, at the proof of security, to use a different rigged verification key for each machine. Since the key is determined only when a machine is being garbled (and its computation and output values are already fixed), we can use a rigged verification key that enforces the correct values, and obtain the same tight security reduction as in the static setting.

Adaptive accumulators from adaptive puncturable hash functions. We build adaptive accumulators from a new primitive called an *adaptively puncturable (AP) hash family*. In this primitive a standard collision resistant hash family $\{\mathcal{H}_\lambda\}$ is augmented with three algorithms Verify, GenVK, $\mathsf{GenBindingVK}$. GenVK generates a verification key vk, which can be later used in $\mathsf{Verify}(\mathsf{vk}, x, y)$ to check that $h(x) = y$. $\mathsf{GenBindingVK}(x^*)$ produces a binding key vk^* such that $\mathsf{Verify}(\mathsf{vk}^*, x, y = h(x^*))$ accepts if *and only if* $x = x^*$. Finally, we require that real and binding verification keys should be indistinguishable even for the adversary which chooses x^* adaptively after seeing h.

The construction of adaptive accumulators from AP hash functions proceeds as follows. The public key is an AP hash function h, and the initial accumulator value ac_0 is the root of a Merkle tree on the initial data store (which can be thought of as empty, or the all-0 string) using h. We maintain the invariant that at every moment the root value ac is the result of hashing down the memory store. In order to write a new symbol v to a position L the evaluator recomputes all hashes on the path from the root to L. The "opening information" for v at L is all sibling hashes of the path from the root to L.

The verification key is a sequence of $d = \log |S|$ (honest) verification keys for h - one for each level of the tree. The "rigged" verification key for accumulator value ac^* and value v^* at location L^* consists of a sequence of d rigged verification keys for the AP hash, where each key forces the opening of a single value along the path from the root to leaf L^*. Security of the adaptive accumulator follows from the security of the AP hash via standard reduction.

Constructing AP hash. We construct adaptively puncturable hash function ensembles from indistinguishability obfuscation for circuits, plus collision-resistant hash functions with the property that any image has at most polynomially many preimages. (This implies that the CRHF shrinks at most logarithmically many bits). We say that a hash function is *c-bounded* if the number of preimages for any image is no more than c. To be usable in the Merkle-Damgård construction, we will also need that the hash functions have domain $\{0,1\}^\lambda$ and range $\{0,1\}^{\lambda'}$ for some $\lambda' < \lambda$. For simplicity we focus on the setting where $\lambda = \lambda' + 1$. We construct 4-bounded CRHFs assuming hardness of discrete log and 64-bounded CRHFs assuming hardness of factoring.

Our construction of an AP hash family can be understood in two steps.

1. First we construct a c-bounded AP hash family from any c-bounded hash family $\{\mathcal{H}_\lambda\}$. This is done as follows: The public key is a hash function $h \leftarrow \mathcal{H}_\lambda$. A verification key vk is $\mathsf{i}\mathcal{O}(V)$, where V is the program that on input x, y outputs 1 if $h(x) = y$. A "rigged" verification key vk^* that is binding for input x^* is $\mathsf{i}\mathcal{O}(V_{x^*})$ where V_{x^*} is the program that on input (x, y) does the following:

   - if $y = h(x^*)$, it accepts if and only if $x = x^*$;
   - otherwise it accepts if and only if $y = h(x)$.

   Since h is c-bounded, the functionality of V and V_{x^*} differ only on polynomially many (difficult to find) inputs. Therefore, the real and "rigged" verification keys are indistinguishable following the $\mathsf{di}\mathcal{O}$-$\mathsf{i}\mathcal{O}$ equivalence for circuits with polynomially many differing inputs [BCP14].

2. Next we construct AP hash families which are length halving (and are thus not polynomially bounded) from bounded AP hashing. This is done in the natural way by extending the hash function's domain using Merkle-Damgård. Suppose we start with a function $h' : \{0,1\}^{\lambda+1} \to \lambda$, and build $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$. A verification key vk for h is an obfuscated circuit C which takes x and y, and directly checks that $h(x) = y$.

The proof of security involves a sequence of hybrids, in which C is modified to contain a verification key for h'. This implies that in the real world, C must also be padded to this same size. In other words, the verification key vk must be as large as twice-obfuscated circuit computing h'. We note that it is possible to avoid this overhead by instead distributing λ different verification keys for h', but we avoid this approach for conceptual simplicity.

Adaptive Same Transcript Indistinguishability. We return to the challenges encountered when trying to use the [CH16] construction in our adaptive setting. With adaptive accumulators in hand, the additional modifications made on the use of iterator and splittable signatures are relatively local. Since these primitives do not access the long-lived shared memory, it suffices to generate a fresh instance of each primitive for each new query.

Adaptive Same Address Indistinguishability. Next we upgrade the next two layers in the [CH16] construction, namely the fixed-access and fixed-address garbling schemes, to adaptively secure ones. This is done with relatively local changes from the original construction. Specifically we include the index and time step in the domain of puncturable PRF that is used to derive the randomness of the one-time-pad-like encryption on the state and memory. The technical details can be found in the main construction.

Adaptive Full Security. Recall that in [CH16] full garbling is achieved by applying an Oblivious RAM scheme on top of the fixed-access garbling. The randomness for the ORAM accesses is sampled using a PRF. This leads to a situation where a PRF key is first used inside a program M_i for some execution i. Later, the key needs to be punctured at a point that may depend on the PRF values. This leads to another adaptivity problem.

We get around this problem by noticing that the Chung-Pass ORAM has a special property which allows us to guess which points to puncture with only polynomial security loss. This property, which we call *strong localized randomness*, is sketched as follows. Let R be the randomness used by the ORAM. Let $\vec{A_i} = \vec{a_{i1}}, \ldots, \vec{a_{im}}$ be a set of locations accessed by the ORAM during emulation of access i. The strong localized randomness property guarantees that there exists a set of intervals $I_{11}, \ldots, I_{Tm}$ with each $I_{ij} \subseteq [1, |R|]$ such that:

1. Each $\vec{a_{ij}}$ depends only on $R_{I_{ij}}$, i.e., the part of the randomness R indexed with I_{ij}; furthermore, $\vec{a_{ij}}$ is efficiently computable from I_{ij};
2. All I_{ij} are mutually disjoint;
3. All I_{ij} are efficiently computable given the sequence of memory operations.

To see that the Chung-Pass ORAM has strong localized randomness, observe that in its non-recursive form, each virtual access of addr touches two paths: one is the path used for the eviction, which is purely random, and the other is determined by the randomness chosen in the previous virtual access of addr. Therefore, the set of accessed locations is determined by two randomness intervals. When the ORAM is applied recursively, the sequence of accesses is determined by $O(\log S)$ intervals. Since the number of intervals in the range $[1, \ldots, |R|]$ is only polynomial in the security parameter, the reduction can guess the interval (and therefore the points to puncture at) with only polynomial security loss.

In contrast, the localized randomness property used in [CH16] differs from property 1 above, requiring only that each $\mathbf{A}_i$ depends on polylogarithmically many bits of R. This does not suffice for us, because there are superpolynomially many possible dependencies, and so the reduction cannot guess correctly with any non-negligible probability.

Concurrent and independent work. A potential alternative to our adaptive positional accumulators is to build on the *somewhere statistically binding (SSB) hash* of Hubáček and Wichs [HW15] or Okamoto et al. [OPWW15]. SSB hashes have a similar flavor to positional accumulators, but they allow rigging to be statistically binding at a hidden location L^*. However it turns out that SSB hashes alone do not suffice for positional accumulators, even in the non-adaptive case! In concurrent and independent work, Ananth et al. [ACC+16] give a stronger definition of SSBs which *does* suffice, and then show that a known construction [OPWW15] satisfies this stronger property. Their reduction can then be made adaptive by guessing L^*, at the price of reducing the reduction's winning probability by a factor proportional to the database size. In all, their construction uses a somewhat stronger assumption than ours (DDH vs. discrete log) and their security reduction is somewhat less efficient than ours.

RAM Machine Concatenation

For RAM machines $M_1, \ldots, M_t$, we let $M_1; \ldots; M_t$ denote the RAM machine which on input x, sequentially executes M_1 through M_t, and then outputs whatever M_t

outputs.

5.1 Adaptive Garbled RAM Definition

Syntax. A garbling scheme for RAM programs is a tuple of p.p.t. algorithms (Setup, GbPrg, GbMem, Eval).

> $\mathsf{Setup}(1^\lambda)$ takes the security parameter λ in unary and outputs a secret key $\mathbf{K}$.
>
> $\mathsf{GbMem}(\mathbf{K}, x)$ takes a secret key $\mathbf{K}$ and an input x, and then outputs a memory configuration $\tilde{\mathbf{x}}$.
>
> $\mathsf{GbPrg}(SK, M_i, T_i, i)$ takes a secret key SK, a RAM machine M_i, a running time bound T_i, and a sequence number i, and outputs a garbled RAM machine $\tilde{M}_i$.
>
> $\mathsf{Eval}(\tilde{M}, \tilde{x})$: takes a garbled RAM $\tilde{M}$ and gabled input $\tilde{x}$ and evaluates the machine on the input, which we denote $\tilde{M}(\tilde{x})$.

We are interested in garbling schemes which are *correct*, *efficient*, and *secure*.

Correctness. A garbling scheme is said to be correct if for all p.p.t. adversaries $\mathcal{A}$ and every $t = \text{poly}(\lambda)$

$$\Pr\left[\tilde{M}_t(\tilde{s}_{t-1}) = M_t(s_{t-1})\right] \geq 1 - \text{negl}(\lambda),$$

in the probability space defined by sampling

$$\begin{array}{l}
(s_0, S) \leftarrow \mathcal{A}(1^\lambda) \\
SK \leftarrow \mathsf{Setup}(1^\lambda, S) \\
\tilde{s}_0 \leftarrow \mathsf{GbMem}(SK, s_0) \\
\text{for } i = 1, \ldots, t \\
\quad M_i, T_i \leftarrow \mathcal{A}(\tilde{s}_0, \tilde{M}_1, \ldots \tilde{M}_{i-1}) \\
\quad \tilde{M}_i \leftarrow \mathsf{GbPrg}(SK, M_i, T_i, i) \\
\quad s_i = \mathsf{NextMem}(M_i, s_{i-1}) \\
\quad \tilde{s}_i = \mathsf{NextMem}(\tilde{M}_i, \tilde{s}_{i-1}),
\end{array}$$

where

- $\sum T_i \leq \text{poly}(\lambda)$, $|s_0| \leq S \leq \text{poly}(\lambda)$;
- $\mathsf{Space}(M_i, s_{i-1}) \leq S$ and $\mathsf{Time}(M_i, s_{i-1}) \leq T_i$ for each i.

Efficiency. A garbling scheme is said to be efficient if:

1. Setup, GbPrg, GbMem and Eval are probabilistic polynomial-time algorithms. Furthermore, GbMem runs in time linear in the length of x. We require *succinctness* for the garbled programs, which means that the size of a garbled program $\tilde{M}$ is linear in the description length of the plaintext program M. The bounds T_i and S are encoded in binary, so the time to garble does not significantly depend on either of these quantities.

2. With $\tilde{M}_i$ and $\tilde{s}_i$ defined as above, it holds that $\mathsf{Time}(\tilde{M}_i, \tilde{s}_{i-1}) = \tilde{O}(\mathsf{Time}(M_i, s_{i-1}))$ and $\mathsf{Space}(\tilde{M}_i, \tilde{s}_{i-1}) = \tilde{O}(S)$ (hiding polylogarithmic factors in S).

Security. We define the security property of an adaptive RAM garbler as follows.

Definition 5.1. *Let* $\mathcal{GRAM} = (\mathsf{Setup}, \mathsf{GbMem}, \mathsf{GbPrg}, \mathsf{Eval})$ *be a garbling scheme. We define the following two experiments, where each* M_i *is a program with time and space complexity* T_i *and* S *that is evaluated with memory* s_{i-1} *and* $y_i = M_i(s_{i-1})$, $s_i = \mathsf{NextMem}(M_i, s_{i-1})$, *and* $T_i = \mathsf{Time}(M_i, s_{i-1})$.

Experiment $REAL_{\mathcal{A}}(1^\lambda)$	***Experiment*** $IDEAL_{\mathcal{A}}(1^\lambda)$
$(s_0, S) \leftarrow \mathcal{A}(1^\lambda)$	$(s_0, S) \leftarrow \mathcal{A}(1^\lambda)$
$\mathbf{K} \leftarrow \mathsf{Setup}(1^\lambda, S)$	
$\tilde{s}_0 \leftarrow \mathsf{GbMem}(SK, s_0)$	$\tilde{s}_0 \leftarrow \mathsf{Sim}(1^\lambda, \lVert s_0 \rVert)$
$(M_1, 1^{T_1}) \leftarrow \mathcal{A}(\tilde{s}_0)$	$(M_1, 1^{T_1}) \leftarrow \mathcal{A}(\tilde{s}_0)$
$\tilde{M}_1 \leftarrow \mathsf{GbPrg}(SK, M_1, T_1, 1)$	$\tilde{M}_1 \leftarrow \mathsf{Sim}(y_1, T_1)$
for $i = 1$ *to* $\ell = \mathsf{poly}(\lambda)$	*for* $i = 1$ *to* $\ell = \mathsf{poly}(\lambda)$
$(M_{i+1}, 1^{T_{i+1}}) \leftarrow \mathcal{A}(\tilde{M}_i)$	$(M_{i+1}, 1^{T_{i+1}}) \leftarrow \mathcal{A}(\tilde{M}_i)$
$\tilde{M}_{i+1} \leftarrow \mathsf{GbPrg}(\mathbf{K}, M_{i+1}, T_{i+1}, i+1)$	$\tilde{M}_{i+1} \leftarrow \mathsf{Sim}(y_{i+1}, t_{i+1})$
Output : $b \leftarrow \mathcal{A}(\tilde{M}_{n+1})$	**Output** : $b' \leftarrow \mathcal{A}(\tilde{M}_{n+1})$

The garbling scheme $\mathcal{GRAM}$ *is* ϵ-adaptively secure *if*

$$\left|\mathsf{Pr}[1 \leftarrow REAL_{\mathcal{A}}(1^\lambda)] - \mathsf{Pr}[1 \leftarrow IDEAL_{\mathcal{A}}(1^\lambda)]\right| \leq \epsilon.$$

5.2 c-Bounded Collision-Resistant Hash Functions

We say that a hash function ensemble $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$ with $\mathcal{H}_\lambda = \{h_k : D_\lambda \to R_\lambda\}_{k \in \mathcal{K}_\lambda}$ is $c(\cdot)$-bounded if

$$\Pr_{h \leftarrow \mathcal{H}_\lambda}[\forall y \in R_\lambda, \#\{x : h(x) = y\} \leq c(\lambda)] \geq 1 - \mathrm{negl}(\lambda)$$

That is, with high probability, every element in the codomain of h has at most $c(\lambda)$ pre-images. In our adaptively secure garbling scheme, we need $c(\cdot)$ to be any polynomial (smaller is better for the security reduction), and we need $D_\lambda = \{0,1\}^{\lambda'}$ and $R_\lambda = \{0,1\}^{\lambda'-1}$ for some $\lambda' = \mathrm{poly}(\lambda)$. For both of the constructions in this section, we obtain constant $c(\cdot)$.

The starting point for our constructions is the construction of Damgård [Dam89] that uses a claw-free pair of permutations (π_0, π_1) on a domain $\mathcal{D}_\lambda$. In this construction, for some fixed y_0, a hash function h is defined so that

$$h(x) \stackrel{\text{def}}{=} (\pi_{x_0} \circ \cdots \circ \pi_{x_n})(y_0).$$

Unfortunately, while this construction allows one to directly construct arbitrarily-compressing hash families, it in general may not be poly(n)-bounded even if n is chosen so that the hash family compresses by a constant number of bits (i.e., $n = \log|\mathcal{D}_\lambda| + O(1)$). .

However, a slight modification of this construction allows us to take any injective functions $\iota_{in} : \{0,1\}^n \hookrightarrow \mathcal{D}_\lambda$ and $\iota_{out} : \mathcal{D}_\lambda \hookrightarrow \{0,1\}^m$, and produce a 2^k-bounded collision-resistant function mapping $\{0,1\}^{n+k} \to \{0,1\}^m$. As long as such injections exist with $m - n = O(\log\lambda)$, this yields a poly(λ)-bounded collision-resistant hash family.

Theorem 5.2. *If for a random λ-bit prime p, it is hard to solve the discrete log problem in $\mathbb{Z}_p^*$, then there exists a 4-bounded CRHF ensemble $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathcal{H}_\lambda$ consists of functions mapping $\{0,1\}^{\lambda+1} \to \{0,1\}^\lambda$.*

Proof. Let p be a random λ-bit prime, and let g and h be randomly chosen generators of $\mathbb{Z}_p^*$. Our hash function is keyed by p, g, h. It is well-known that the permutations $\pi_0(x) = g^x$ and $\pi_1(x) = g^x h$ are claw-free. It is easy to see there is an injection $\iota_{in} : \{0,1\}^{\lambda-1} \to \mathbb{Z}_p^*$ and an injection $\iota_{out} : \mathbb{Z}_p^* \to \{0,1\}^\lambda$. Define a hash function

$$f : \{0,1\}^{\lambda-1} \times \{0,1\} \times \{0,1\} \to \{0,1\}^\lambda$$

$$a, b, c \mapsto \iota_{out}(\pi_c(\pi_b(\iota_{in}(a))))$$

Clearly given $x \neq x'$ such that $f(x) = f(x')$, one can find a claw (and therefore find $\log_g h$), so f is collision-resistant. Also for any given image, there is at most one corresponding pre-image per choice of b, c, so f is 4-bounded. □

Theorem 5.3. *If for random λ-bit primes p and q, with $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$, it is hard to factor $N = pq$, then there exists a 64-bounded CRHF ensemble $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathcal{H}_\lambda$ consists of functions mapping $\{0,1\}^{2\lambda+1} \to \{0,1\}^{2\lambda}$.*

Proof. First, we construct injections $\iota_0 : \{0,1\}^{2\lambda-4} \to [N/6]$ and $\iota_1 : [N/6] \to \mathbb{Z}_N^* \cap [N/2]$, using the fact that for sufficiently large p and q, for any integer $x \in [N/6]$, at least one of $3x, 3x+1$, and $3x+2$ is relatively prime to N. $\iota_1(x)$ is therefore well-defined as the smallest of $\{3x, 3x+1, 3x+2\} \cap \mathbb{Z}_n^*$. Let $\iota_{in} : \{0,1\}^{2\lambda-4} \to \mathbb{Z}_N^* \cap [N/2]$ denote $\iota_1 \circ \iota_0$. Let ι_{out} denote an injection from $\mathbb{Z}_N^* \to \{0,1\}^{2\lambda}$.

Next, we use a construction of claw-free permutations due to Goldwasser, Micali, and Rivest [GMR88]. Specifically, in this construction the permutations π_0 and π_1 are defined such that $\pi_0(x) = x^2 \pmod N$ and $\pi_1(x) = 4x^2 \pmod N$, where the domain of π_0 and π_1 is the set of quadratic residues mod N.

Now we define the hash function

$$f : \{0,1\}^{2\lambda-4} \times \{0,1\}^5 \to \{0,1\}^{2\lambda}$$

$$f(x,y) = (\iota_{out} \circ \pi_{y_5} \circ \cdots \circ \pi_{y_1})(\iota_{in}(x)^2 \bmod N)$$

This is 64-bounded because for any given image, there is at most one pre-image under $\iota_{out} \circ \pi_{y_5} \circ \cdots \circ \pi_{y_1}$ per possible y value. This accounts for a factor of 32. The remaining factor of 2 comes from the fact that every quadratic residue has four square roots, two of which are in $[N/2]$ (the image of ι_{in}). The collision resistance of $x \mapsto \iota_{in}(x)^2 \pmod N$ follows from the fact that the two square roots are nontrivially related, i.e., neither is the negative of the other, so given both it would be possible to factor N. □

Notation. For a function $h : \{0,1\}^{\lambda+1} \to \{0,1\}^{\lambda}$, we let h^0 denote the identity function and for $k > 0$ inductively define

$$h^k : \{0,1\}^{\lambda+k} \to \{0,1\}^{\lambda}$$
$$h^k(x) = h(x_1 \| h^{k-1}(x_2 \| \cdots \| x_{\lambda+k}))$$

5.3 Adaptively Puncturable Hash Functions

We say that an collision-resistant hash family $\mathcal{H} = \{\mathcal{H}_\lambda\}$ is *adaptively puncturable* if there are algorithms Verify, GenVK, and ForceGenVK such that:

Correctness

For all $x, y \in \{0,1\}^*$ and all $h \in \mathcal{H}_\lambda$, it holds with probability 1 that

$$\mathsf{Verify}(\mathsf{vk}, x, y) = 1 \iff y = h(x)$$

in the probability space defined by sampling $\mathsf{vk} \leftarrow \mathsf{GenVK}(1^\lambda, h)$.

Forced Verification

For all $x, x^* \in \{0,1\}^*$ and $h \in \mathcal{H}$, it holds with probability 1 that

$$\mathsf{Verify}(\mathsf{vk}, x, h(x^*)) = 1 \iff x = x^*$$

in the probability space defined by sampling $\mathsf{vk} \leftarrow \mathsf{ForceGenVK}(1^\lambda, h, x^*)$.

Indistinguishability

For all polynomial-sized circuit ensembles $\{\mathcal{A}_\lambda^{(1)}\}$ and $\{\mathcal{A}_\lambda^{(2)}\}$, it holds for some negligible function ν that

$$\Pr\left[\mathcal{A}_\lambda^{(2)}(s, \mathsf{vk}_b) = b\right] \leq \frac{1}{2} + \nu(\lambda)$$

in the probability space defined by sampling

$$
\begin{aligned}
&h \leftarrow \mathcal{H}_\lambda \\
&(x^*, s) \leftarrow \mathcal{A}_\lambda^{(1)}(h) \\
&\mathsf{vk}_0 \leftarrow \mathsf{GenVK}(1^\lambda, h) \\
&\mathsf{vk}_1 \leftarrow \mathsf{ForceGenVK}(1^\lambda, h, x^*) \\
&b \leftarrow \{0,1\}
\end{aligned}
$$

Theorem 5.4. *If $i\mathcal{O}$ for circuits exists and there is a $\mathrm{poly}(\lambda)$-bounded CRHF ensemble mapping $\{0,1\}^{\lambda+1} \to \{0,1\}^\lambda$, then there is an adaptively puncturable hash function ensemble mapping $\{0,1\}^{2\lambda}$ to $\{0,1\}^\lambda$.*

Let $\mathcal{H} = \{\mathcal{H}_\lambda\}$ be a $\mathrm{poly}(\lambda)$-bounded CRHF ensemble, where $\mathcal{H}_\lambda$ is a family of functions mapping $\{0,1\}^{\lambda+1} \to \{0,1\}^\lambda$. We define an adaptively puncturable hash function ensemble $\mathcal{F} = \{\mathcal{F}_\lambda\}$, where $\mathcal{F}_\lambda$ is a family of functions that map $\{0,1\}^{2\lambda}$ to $\{0,1\}^\lambda$.

Setup

The key space for $\mathcal{F}_\lambda$ is the same as the key space for $\mathcal{H}_\lambda$.

Evaluation

For a key $h \in \mathcal{H}_\lambda$ and a string $x \in \{0,1\}^{2\lambda}$, we define

$$f_h(x) = h^\lambda(x)$$

Verification

$\mathsf{GenVK}(1^\lambda, f_h)$ outputs an $i\mathcal{O}$-obfuscation of a circuit which directly computes

$$x, y \mapsto \begin{cases} 1 & \text{if } f_h(x) = y \\ 0 & \text{otherwise} \end{cases}$$

$\mathsf{ForceGenVK}(1^\lambda, f_h, x^*)$ outputs an $i\mathcal{O}$-obfuscation of a circuit which directly computes

$$x, y \mapsto \begin{cases} 1 & \text{if } y \neq f_h(x^*) \wedge y = f_h(x) \\ 1 & \text{if } (x,y) = (x^*, f_h(x^*)) \\ 0 & \text{otherwise} \end{cases}$$

$\mathsf{Verify}(\mathsf{vk}, x, y)$ simply evaluates and outputs $\mathsf{vk}(x, y)$.

Claim 5.4.1. *No p.p.t. adversary which adaptively chooses x^* after seeing $h \leftarrow \mathcal{H}_\lambda$ can distinguish between $\mathsf{GenVK}(1^\lambda, h)$ and $\mathsf{ForceGenVK}(1^\lambda, h, x^*)$.*

Proof. We present $\lambda+1$ hybrid games $H_0, \ldots, H_\lambda$. In each game h is sampled from $\mathcal{H}_\lambda$, but the circuit given by the challenger to the adversary depends on the game and on x^*. In hybrid H_i, the challenger computes $y^* = h^\lambda(x^*)$ and $y_{\lambda-i} = h^{\lambda-i}(x^*_{i+1} \| \cdots \| x^*_{2\lambda})$.

The challenger then sends $i\mathcal{O}(C_i)$ to the adversary, where C_i has y^*, $y_{\lambda-i}$, and $x_1^*, \ldots, x_i^*$ hard-coded and is defined as

$$C_i(x,y) = \begin{cases} 1 & \text{if } y \neq y^* \wedge y = h^\lambda(x) \\ 1 & \text{if } y = y^* \wedge x_1 = x_1^* \wedge \cdots \wedge x_i = x_i^* \wedge h^{\lambda-i}(x_{i+1} \| \cdots \| x_{2\lambda}) = y_{\lambda-i} \\ 0 & \text{otherwise} \end{cases}$$

The challenger sends $i\mathcal{O}(C_i)$ to the adversary.

It is easy to see that C_0 is functionally equivalent to the circuit produced by GenVK, and C_λ is functionally equivalent to the circuit produced by $\mathsf{ForceGenVK}$. So we only need to show that $H_i \approx H_{i+1}$ for $0 \leq i < \lambda$. We give a sequence of indistinguishable changes to the challenger – specifically, changes which affect the circuit C that the challenger obfuscates and sends to the adversary. The result of these changes will be to transform the functionality of C from that of C_i to that of C_{i+1}.

1. We first change C so that when $y = y^*$, it computes the intermediate value $y' = h^{\lambda-i-1}(x_{i+2} \| \cdots \| x_{2\lambda})$ and outputs 1 if:

   - $h(x_{i+1} \| y') = y_{\lambda-i}$
   - For all $1 \leq j \leq i$, $x_i = x_i^*$.

   When $y \neq y^*$, the behavior of C is unchanged.

   This change preserves functionality (we only introduced a name y' for an intermediate value in the computation) and hence is indistinguishable by $i\mathcal{O}$.

2. Now we change C so that instead of directly checking whether $h(x_{i+1} \| y') = y_{\lambda-i}$, it uses a hard-coded helper circuit $\tilde{V} = i\mathcal{O}(V)$, where

   $$V : \{0,1\} \times \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$$
   $$V(a,b,c) = \begin{cases} 1 & \text{if } c = h(a \| b) \\ 0 & \text{otherwise} \end{cases}$$

   This is functionally equivalent and hence indistinguishable by $i\mathcal{O}$.

3. Now we change V. The challenger computes $y_{\lambda-i-1} = h^{\lambda-i-1}(x_{i+2}^* \| \cdots \| x_{2\lambda}^*)$ and $y_{\lambda-i} = h(x_{i+1}^* \| y_{\lambda-i-1})$, and define

   $$V(a,b,c) = \begin{cases} 1 & \text{if } c \neq y_{\lambda-i} \wedge c = h(a \| b) \\ 1 & \text{if } (a,b,c) = (x_{i+1}^*, y_{\lambda-i-1}, y_{\lambda-i}) \\ 0 & \text{otherwise} \end{cases},$$

   with $y_{\lambda-i}$, $y_{\lambda-i-1}$, and x_{i+1}^* hard-coded. The old and new $\tilde{V}$'s are indistinguishable because:

- By the collision-resistance of h, it is difficult to find an input on which they differ.
- Because $\mathcal{H}_\lambda$ is poly(λ)-bounded, they differ on only polynomially many points.
- $\mathsf{i}\mathcal{O}$ is equivalent to $\mathsf{di}\mathcal{O}$ for circuits which differ on polynomially many points.

C is now functionally equivalent to C_{i+1}. □

5.4 Adaptively Secure Positional Accumulators

In this section we define and construct adaptive positional accumulators (APA). We use this primitive for memory authentication in our garbling construction. A garbled program will be an obfuscated functionality where one input is a succinct commitment ac to some memory contents, another is a piece of data v allegedly resulting from a memory operation op, and another is a commitment ac', allegedly to the resulting memory configuration. Informally, APAs provide a way for the garbled program to check the consistency of v and ac' with ac (given a short proof),

As described so far, Merkle trees satisfy our needs, and indeed our construction is built around a Merkle tree. However, we require more. As in the positional accumulators of [KLW15], we need a way to indistinguishably "rig" the public parameters so that for some ac and op, there is exactly one (ac', v) with any accepting proof. We deviate from [KLW15] by separating the parameters used for proof verification from those used for updating the accumulator, and allowing the rigged $(\mathsf{ac}, \mathsf{op})$ to be chosen adaptively as an adversarial function of the update parameters.

We now formally define the algorithms of the APA primitive.

$\mathsf{SetupAcc}(1^\lambda, S) \to \mathsf{PP}, \mathsf{ac}_0, \mathsf{store}_0$

The setup algorithm takes as input the security parameter λ in unary and a bound S (in binary) on the memory addresses accessed. SetupAcc produces as output public parameters PP, an initial accumulator value ac_0, and an initial data store store_0.

$\mathsf{Update}(\mathsf{PP}, \mathsf{store}, \mathsf{op}) \to \mathsf{store}', \mathsf{ac}', v, \pi$

The update algorithm takes as input the public parameters PP, a data store store, and a memory operation op. Update then outputs a new store store', a memory value v, a succinct accumulator ac', and a succinct proof π.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{ac}, \mathsf{op}, \mathsf{ac}', v, \pi) \to \{0, 1\}$

The verification algorithm takes as inputs a verification key vk, an initial accumulator value ac, a memory operation op, a resulting accumulator ac', a memory value v, and a proof π. Verify then outputs 0 or 1. Intuitively, Verify checks the following statement:

π is a proof that the operation op*, when applied to the memory configuration corresponding to* ac*, yields a value* v *and results in a memory configuration corresponding to* ac'*.*

Looking ahead, Verify will be run by a garbled program to authenticate the memory values that the evaluator gives it.

$\mathsf{SetupVerify}(\mathsf{PP}) \to \mathsf{vk}$

SetupVerify generates a regular verification key for checking Update's proofs. This is the verification key that is used in the "real world" garbled programs.

$\mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_1, \ldots, \mathsf{op}_k)) \to \mathsf{vk}$

SetupEnforceVerify takes a sequence of memory operations, and generates a verification key which is perfectly sound when verifying the action of op_k in the sequence $(\mathsf{op}_1, \ldots, \mathsf{op}_k)$. This type of verification key is used in the hybrid garbled programs in our security proof.

An adaptive positional accumulator must satisfy the following properties.

Correctness

Let $\mathsf{op}_0, \ldots, \mathsf{op}_k$ be any arbitrary sequence of memory operations, and let v_i^* denote the result of the i^{th} memory operation when $(\mathsf{op}_0, \ldots, \mathsf{op}_{k-1})$ are sequentially executed on an initially empty memory.

Correctness requires that, when sampling

$$\begin{aligned}
&\mathsf{PP}, \mathsf{ac}_0, \mathsf{store}_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, S)\\
&\mathsf{vk} \leftarrow \mathsf{SetupVerify}(\mathsf{PP})\\
&\text{For } i = 0, \ldots, k\text{:}\\
&\quad \mathsf{store}_{i+1}, \mathsf{ac}_{i+1}, v_i, \pi_i \leftarrow \mathsf{Update}(\mathsf{PP}, \mathsf{store}_i, \mathsf{op}_i)\\
&\quad b_i \leftarrow \mathsf{Verify}(\mathsf{vk}, \mathsf{ac}_i, \mathsf{op}_i, \mathsf{ac}_{i+1}, v_i, \pi_i)
\end{aligned}$$

it holds (with probability 1) that for all $j \in \{0, \ldots, k\}$, $v_j = v_j^*$ and $b_j = 1$

Enforcing

Enforcing requires that for all space bounds S, all sequences of operations $\mathsf{op}_0, \ldots, \mathsf{op}_{k-1}$, when sampling

$$\begin{aligned}
&\mathsf{PP}, \mathsf{ac}_0, \mathsf{store}_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, S)\\
&\mathsf{vk} \leftarrow \mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_0, \ldots, \mathsf{op}_{k-1}))\\
&\text{For } i = 0, \ldots, k-1\\
&\quad \mathsf{store}_{i+1}, \mathsf{ac}_{i+1}, v_i, \pi_i \leftarrow \mathsf{Update}(\mathsf{PP}, \mathsf{store}_i, \mathsf{op}_i)
\end{aligned}$$

it holds (with probability 1) that for all accumulators $\hat{\mathsf{ac}}$, all values $\hat{v}$, and all proofs $\hat{\pi}$, if $\mathsf{Verify}(\mathsf{vk}, \mathsf{ac}_{k-1}, \mathsf{op}_{k-1}, \hat{\mathsf{ac}}, \hat{v}, \hat{\pi}) = 1$, then $(\hat{v}, \hat{\mathsf{ac}}) = (v_{k-1}, \mathsf{ac}_k)$

Indistinguishability of Enforcing Verify

Now we require that the output of $\mathsf{SetupVerify}(\mathsf{PP})$ is indistinguishable from

the output of $\mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_1, \ldots, \mathsf{op}_k))$, even when $(\mathsf{op}_1, \ldots, \mathsf{op}_k)$ are chosen adaptively as a function of PP.

More formally, for all p.p.t. $\mathcal{A}_1$ and $\mathcal{A}_2$,

$$\Pr\left[\mathcal{A}_2(s, \mathsf{vk}_b) = b\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

in the probability space defined by sampling

$$\begin{aligned}
&\mathsf{PP}, \mathsf{ac}_0, \mathsf{store}_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, S)\\
&(\mathsf{op}_0, \ldots, \mathsf{op}_{k-1}), s \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{PP})\\
&\mathsf{vk}_0 \leftarrow \mathsf{SetupVerify}(\mathsf{PP})\\
&\mathsf{vk}_1 \leftarrow \mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_0, \ldots, \mathsf{op}_{k-1}))\\
&b \leftarrow \{0,1\}
\end{aligned}$$

Efficiency

In addition to all the algorithms being polynomial-time, we require that:

- The size of an accumulator is $\mathrm{poly}(\lambda)$.
- The size of proofs is $\mathrm{poly}(\lambda, \log S)$.
- The size of a store is $O(S)$

Theorem 5.5. *If there is an adaptively puncturable hash function ensemble $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ with $\mathcal{H}_\lambda = \{H_k : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda\}_{k\in\mathcal{K}_\lambda}$, then there exists an adaptive positional accumulator.*

Proof. We construct an adaptive positional accumulator in which stores are low-depth binary trees, each node of which contains a λ-bit value. The accumulator corresponding to a given store is the value held by the root node. The public parameters for the accumulator consist of an adaptively puncturable hash $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$, and we preserve the invariant that the value in any internal node is equal to the hash h applied to its children's values. It will be convenient for us to assume the existence of a $\perp$, which is represented as a λ-bit string not in the image of h. Without loss of generality, h can be chosen to have such a value.

$\mathsf{Setup}(1^\lambda, S) \to \mathsf{PP}, \mathsf{ac}_0, \mathsf{store}_0$

Setup samples $h \leftarrow \mathcal{H}_\lambda$, and sets $\mathsf{PP} = h$, $\mathsf{ac}_0 = h(\perp\|\perp)$, and store_0 to be a root node with value $h(\perp\|\perp)$.

$\mathsf{Update}(h, \mathsf{store}, \mathsf{op}) \to \mathsf{store}', \mathsf{ac}', v, \pi$

Suppose op is $\mathsf{ReadWrite}(\mathsf{addr} \mapsto v')$. There is a unique leaf node in store which is indexed by a prefix of addr. Let v be the value of that leaf, and let π be the values of all siblings on the path from the root to that leaf.

Update adds a leaf node indexed by the entirety of addr to store if no such node already exists, and sets the value of the leaf to v'. Then Update updates the value of ancestor of that leaf to preserve the invariant.

$\mathsf{SetupVerify}(h) \to \mathsf{vk}$

For $i = 1, \ldots, \log S$, $\mathsf{SetupVerify}$ samples

$$vk_i \leftarrow \mathsf{GenVK}(1^\lambda, h)$$

and sets $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_{\log S})$.

$\mathsf{Verify}((\mathsf{vk}_1, \ldots, \mathsf{vk}_{\log S}), \mathsf{ac}, \mathsf{op}, \mathsf{ac}', v, (w_1, \ldots, w_d)) \to \{0, 1\}$

Define $z_d := v$. Let $b_1 \cdots b_{d'}$ denote the bit representation of the address on which op acts. For $0 \leq i < d$, Verify computes

$$z_i = \begin{cases} h(w_{i+1} \| z_{i+1}) & \text{if } b_{i+1} = 1 \\ h(z_{i+1} \| w_{i+1}) & \text{otherwise} \end{cases}$$

For all i such that $b_i = 1$, Verify checks that $\mathsf{vk}_i(w_{i+1} \| z_{i+1}, z_i) = 1$. For all i such that $b_i = 0$, Verify checks that $\mathsf{vk}_i(z_{i+1} \| w_{i+1}, z_i) = 1$. If all these checks pass, then Verify outputs 1; otherwise, Verify outputs 0.

$\mathsf{SetupEnforceVerify}(h, (\mathsf{op}_1, \ldots, \mathsf{op}_k)) \to \mathsf{vk}$

Computes the store_{k-1} which would result from processing $\mathsf{op}_1, \ldots, \mathsf{op}_{k-1}$. Suppose op_k accesses address $\mathsf{addr}_k \in \{0, 1\}^{\log S}$. Then there is a unique leaf node in store_{k-1} which is indexed by a prefix of addr_k; write this prefix as $b_1 \cdots b_d$.

For each $i \in \{1, \ldots, d\}$, define z_i as the value of the node indexed by $b_1 \cdots b_i$, and let w_i denote the value of that node's sibling. If $b_i = 0$, sample

$$\mathsf{vk}_i \leftarrow \mathsf{ForceGenVK}(1^\lambda, h, z_i \| w_i).$$

Otherwise, sample

$$\mathsf{vk}_i \leftarrow \mathsf{ForceGenVK}(1^\lambda, h, w_i \| z_i).$$

For $i \in \{d+1, \ldots, \log S\}$, just sample $\mathsf{vk}_i \leftarrow \mathsf{GenVK}(1^\lambda, h)$.

Finally we define the total verification key to be $(\mathsf{vk}_1, \ldots, \mathsf{vk}_{\log S})$.

All the requisite properties of this construction are easy to check. □

5.5 Fixed-Transcript Garbling

Next we present the first step in our construction, a garbling scheme that provides adaptive security for RAM programs that have the same transcript. The notion extends the first stage of Canetti-Holmgren scheme into the adaptive setting. The construction employs the adaptive positional accumulators, in addition to minor changes to the other primitives.

We define fixed transcript security via the following game.

1. The challenger samples $\mathsf{SK} \leftarrow \mathsf{Setup}(1^\lambda, S)$ and $b \leftarrow \{0, 1\}$.

2. The adversary sends a memory configuration s to the challenger. The challenger sends back $\mathsf{GbMem}(\mathsf{SK}, s)$.

3. The adversary repeatedly sends pairs of RAM programs (M_i^0, M_i^1) along with a time bound T_i, and the challenger sends back $\tilde{M}_i^b \leftarrow \mathsf{GbPrg}(\mathsf{SK}, M_i^b, T_i, i)$. Each pair (M_i^0, M_i^1) is chosen adaptively after seeing $\tilde{M}_{i-1}^b$.

4. The adversary outputs a guess b'.

Let $((M_1^0, M_1^1), \ldots, (M_\ell^0, M_\ell^1))$ denote the sequence of pairs of machines output by the adversary. The adversary is said to win if $b' = b$ and:

- Sequentially executing $M_1^0, \ldots, M_\ell^0$ on initial memory configuration s yields the same transcript as executing $M_1^1, \ldots, M_\ell^1$.
- Each M_i^b runs in time at most T_i and space at most S.

Definition 5.6. *A garbling scheme is* fixed-transcript secure *if for all p.p.t. algorithms $\mathcal{A}$, there is a negligible function* negl *so that $\mathcal{A}$'s probability of winning the game is at most $\frac{1}{2} + \mathrm{negl}(\lambda)$.*

Theorem 5.7. *Assuming the existence of indistinguishability obfuscation and an adaptive positional accumulator, there is a fixed-transcript secure garbling scheme.*

Proof. Our construction follows closely the fixed-transcript garbling scheme of [CH16], using our *adaptive* positional accumulator in place of [KLW15]'s positional accumulator. Let $\{\mathcal{F}_\lambda\}$ denote a puncturable PRF family.

$\mathsf{Setup}(1^\lambda, S) \to \mathsf{SK}$: sample

$$(\mathsf{Acc.PP}, \mathsf{ac}_{\mathsf{init}}, \mathsf{store}_{\mathsf{init}}) \leftarrow \mathsf{Acc.Setup}(1^\lambda, S)$$
$$(\mathsf{Itr.PP}, \mathsf{itr}_{\mathsf{init}}) \leftarrow \mathsf{Itr.Setup}(1^\lambda)$$
$$F \leftarrow \mathcal{F}_\lambda$$

and set $\mathsf{SK} = (\mathsf{Acc.PP}, \mathsf{ac}_{\mathsf{init}}, \mathsf{store}_{\mathsf{init}}, \mathsf{Itr.PP}, \mathsf{itr}_{\mathsf{init}}, F)$.

$\mathsf{GbMem}(\mathsf{SK}, s) \to \tilde{s}$: GbMem updates the APA $(\mathsf{ac}_{\mathsf{init}}, \mathsf{store}_{\mathsf{init}})$ to set the underlying memory to s (via a sequence of calls to Update) and let $\mathsf{ac}_0, \mathsf{store}_0$ denote the result. It then generates $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F(1,0))$, where $(1,0)$ represents the initial index number i and initial time-step number 0[2]. Finally, GbMem computes $\sigma_0 \leftarrow \mathsf{Spl.Sign}(\mathsf{sk}, (\bot, \bot, \mathsf{ac}_0, \mathsf{ReadWrite}(0 \mapsto 0)))$. Here the first $\bot$ represents an initial local state q_0 for M_1, and the second $\bot$ represents an initial iterator value itr_0. GbMem outputs $\tilde{s} = (\sigma_0, \mathsf{ac}_0, \mathsf{store}_0)$.

$\mathsf{GbPrg}(\mathsf{SK}, M_i, T_i, i) \to \tilde{M}_i$: GbPrg first transforms M_i so that its initial state is $\bot$. Note this can be done without loss of generality by hard-coding the "real" initial state in the transition function. GbPrg then computes $\tilde{C}_i \leftarrow i\mathcal{O}(C_i)$, where C_i is described in Algorithm 11.

[2]Looking ahead, all the intermediate $(\mathsf{sk}, \mathsf{vk})$ key pairs are generated by applying F to the (index, time-step) tuple.

Input: Time t, state q, iterator itr, accumulator ac, operation op, signature σ, memory value v, new accumulator ac', proof π

Data: Puncturable PRF F, RAM machine M_i with transition function δ_i, Accumulator verification key $\mathsf{vk}_{\mathsf{Acc}}$, index i, iterator public parameters $\mathsf{Itr.PP}$, time bound T_i

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F(i,t))$;
if $t > T_i$ *or* $\mathsf{Spl.Verify}(\mathsf{vk}, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}), \sigma) = 0$ *or* $\mathsf{Acc.Verify}(\mathsf{vk}_{\mathsf{Acc}}, \mathsf{ac}, \mathsf{op}, \mathsf{ac}', v, \pi) = 0$ **then return** $\perp$;
$\mathsf{out} \leftarrow \delta_i(q, v)$;
if $\mathsf{out} \in Y$ **then**
  $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F(i+1, 0))$;
  return $\mathsf{out}, \mathsf{Sign}(\mathsf{sk}', (\perp, \perp, \mathsf{ac}', \mathsf{ReadWrite}(0 \mapsto 0))$
else
  Parse out as (q', op');
  $\mathsf{itr}' \leftarrow \mathsf{Itr.Iterate}(\mathsf{Itr.PP}, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}))$;
  $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F(i, t+1))$;
  return $(q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'), \mathsf{Sign}(\mathsf{sk}', (q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'))$

Algorithm 11: Transition function for M_i, with memory verified by a signed accumulator.

Finally, GbPrg defines and outputs a RAM machine $\tilde{M}_i$, which has $\tilde{C}_i$ hard-coded as part of its transition function, such that $\tilde{M}_i$ does the following:

1. Reads $(\mathsf{ac}_0, \sigma_0)$ from memory. under the names $(\mathsf{ac}_s, \sigma_s)$). Define $\mathsf{op}_0 = \mathsf{ReadWrite}(0 \mapsto 0)$, $q_0 = \perp$, and $\mathsf{itr}_0 = \perp$.
2. For $t = 0, 1, 2, \ldots$:
   (a) Compute $\mathsf{store}_{t+1}, \mathsf{ac}_{t+1}, v_t, \pi_t \leftarrow \mathsf{Acc.Update}(\mathsf{Acc.PP}, \mathsf{store}_t, \mathsf{op}_t)$.
   (b) Compute $\mathsf{out}_t \leftarrow \tilde{C}_i(t, q_t, \mathsf{itr}_t, \mathsf{ac}_t, \mathsf{op}_t, \sigma_t, v_t, \mathsf{ac}_{t+1}, \pi_t)$.
   (c) If out_t parses as (y, σ), then write $(\mathsf{ac}_{t+1}, \sigma)$ to memory, output y, and terminate.
   (d) Otherwise, parse out_t as $(q_{t+1}, \mathsf{itr}_{t+1}, \mathsf{ac}_{t+1}, \mathsf{op}_{t+1}), \sigma_{t+1}$ or terminate if out_t is not of this form.

We note that GbPrg can efficiently produce $\tilde{M}_i$ from $\tilde{C}_i$ and $\mathsf{Acc.PP}$. This means that later, when we prove security, it will suffice to analyze a game in which the adversary receives $\tilde{C}_i$ instead of $\tilde{M}_i$.

$\mathsf{Eval}(\tilde{M}, \tilde{s})$ The evaluation algorithm runs $\tilde{M}$ on the garbled memory $\tilde{s}$, and outputs $\tilde{M}(\tilde{s})$.

Correctness and efficiency are easy to verify. We show that the challenger in the real security game is indistinguishable from one for which the adversary's view is independent of b. We present a sequence of hybrid games $H_0, \ldots, H_{\ell+1}$, and show that all p.p.t. algorithms $\mathcal{A}$ have negligibly different advantages in adjacent games. In each hybrid, the memory query is answered as in the real game.

Hybrid H_0 In hybrid H_0, we add a "B"-track for execution to C_i. Instead of just checking that $((q, \mathsf{op}, \mathsf{ac}, \mathsf{itr}), \sigma)$ is accepted under vk_t^A, we also allow it to be accepted under vk_t^B, which is derived from a different puncturable PRF F_B. In this second case, we proceed as before except that we compute with δ_i^0 instead of δ_i^b, and we sign the eventual outputs using sk_{t+1}^B instead of sk_{t+1}^A.

The indistinguishability of this change follows by $O(t)$ applications of the indistinguishability of punctured keys, together with the security of $\mathsf{i}\mathcal{O}$. In particular, we can add any functionality we want (by IO) under an always-rejecting $\mathsf{vk}_{i,j,\emptyset}^B$ verification key, and then indistinguishably replace $vk_{i,j,\emptyset}^B$ with vk^B. We start by modifying the last time step, and work backwards because under $\mathsf{vk}_{i,j}^B$, we use the signing key $\mathsf{sk}_{i,j+1}^B$. By working backwards, we avoid the issue that $\mathsf{vk}_{i,j,\emptyset}^B$ is *not* indistinguishable from $\mathsf{vk}_{i,j}^B$ if also given $\mathsf{sk}_{i,j}^B$.

Hybrids H_i In hybrid H_i for $1 \leq i \leq \ell + 1$, the first $i - 1$ program queries are answered differently. For $1 \leq j \leq i-1$, the circuits C_j have hard-coded the transition function for M_j^0 instead of M_j^b. The challenger computes $s_j = \mathsf{NextMem}(M_j^0(s_{j-1}))$, and hard-codes the corresponding accumulator ac_{s_j} into the circuit C_j. The resulting circuit is illustrated in Algorithm 13.

It remains to show that $H_i \approx H_{i+1}$. This is shown using the techniques of [KLW15]. The main difference is that in our setting the positional accumulator needs to be adaptively secure.

1. We hard-code $\mathsf{vk}_{i,0}^A$ and $\mathsf{vk}_{i,0}^B$, and puncture F_A and F_B at $\{(i, 0)\}$. This change preserves functionality and is hence indistinguishable by $\mathsf{i}\mathcal{O}$.

2. We replace $\mathsf{vk}_{i,0}^A$ and $\mathsf{vk}_{i,0}^B$ by keys punctured on the sets $\mathcal{M}\backslash\{(q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})\}$ and $\{(q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})\}$ respectively. These changes are indistinguishable by the (selective) indistinguishability of punctured keys.

3. We modify the way that the accumulator verification key $\mathsf{vk}_{\mathsf{Acc}}$ is generated – namely, we generate it as the output of $\mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_{i,0}))$ instead of $\mathsf{SetupVerify}(\mathsf{PP})$. This guarantees that if $\mathsf{Acc.Verify}(\mathsf{vk}_{\mathsf{Acc}}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0}, \mathsf{ac}', v, \pi) = 1$, then $\mathsf{ac}' = \mathsf{ac}_{i,1}$ and $v = v_{i,0}$. This is indistinguishable by the positional accumulator's indistinguishability of enforcing setup. We note this holds even though $\mathsf{op}_{i,0}$ and $\mathsf{ac}_{i,0}$ may be chosen adversarially after observing the positional accumulator's public parameters.

4. At time 0, we use δ_i^0 instead of δ_i^b (on both tracks A and B). By the hypothesis that M_i^0 and M_i^1 have the same transcripts, we know that $\delta_i^0(q_{i,0}, v_{i,0}) = \delta_i^1(q_{i,0}, v_{i,0})$. Because in steps 2 and 3 we have already made our verification keys perfectly binding, this change is indistinguishable by $\mathsf{i}\mathcal{O}$.

5. The verification key for the accumulator $\mathsf{vk}_{\mathsf{Acc}}$ is generated normally as the output of $\mathsf{SetupVerify}(\mathsf{PP})$ instead of $\mathsf{SetupEnforceVerify}(\mathsf{PP}, (\mathsf{op}_{i,0}))$. This is again indistinguishable by the positional accumulator's adaptively enforcing setup.

6. We modify C_i so that at time 0, instead of deciding to sign with $\mathsf{sk}^A_{i,1}$ or $\mathsf{sk}^B_{i,1}$ based on which branch we are in, we decide by looking at $(q, \mathsf{itr}, \mathsf{ac}, \mathsf{op})$. Namely, we use $\mathsf{sk}^A_{i,1}$ if and only if $(q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}) = (q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})$. This is functionally equivalent because of how we have punctured the verification keys $\mathsf{vk}^A_{i,0}$ and $\mathsf{vk}^B_{i,0}$, and hence is indistinguishable by $\mathsf{i}\mathcal{O}$. Note the 'A' branch and 'B' branch are now identical.

7. We generate $\mathsf{Itr.PP}$ using $\mathsf{SetupEnforce}$ so that $\mathsf{itr}' = \mathsf{itr}_{i,1}$ if and only if the tuple $(q, \mathsf{itr}, \mathsf{ac}, \mathsf{op})$ is equal to $(q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})$. This change is indistinguishable by the iterator's (selective) setup indistinguishability.

8. Instead of choosing whether to use $\mathsf{sk}^A_{i,1}$ or $\mathsf{sk}^B_{i,1}$ based on the value of $(q, \mathsf{itr}, \mathsf{ac}, \mathsf{op})$, we choose based on the value of $(q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}')$. This is functionally equivalent because itr' is equal to $\mathsf{itr}_{i,1}$ (and in fact $(q', \mathsf{ac}', \mathsf{op}')$ is equal to $(q_{i,1}, \mathsf{ac}_{i,1}, \mathsf{op}_{i,1})$) if and only if $(q, \mathsf{itr}, \mathsf{ac}, \mathsf{op})$ is equal to $(q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})$, and therefore this change is indistinguishable by the security of $\mathsf{i}\mathcal{O}$.

9. We generate $\mathsf{Itr.PP}$ normally, which is indistinguishable by the iterator's (selective) indistinguishability of setup.

10. Instead of checking whether the signature σ on $(q, \mathsf{ac}, \mathsf{itr})$ verifies under one of vk^A_0 (which is punctured at $\mathcal{M} \setminus \{(q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0})\}$) and vk^B_0 (which is punctured at $\{q_{i,0}, \mathsf{itr}_{i,0}, \mathsf{ac}_{i,0}, \mathsf{op}_{i,0}\}$), we only check that it verifies under the *unpunctured* $\mathsf{vk}^A_{i,0}$. This is indistinguishable by the splittable signature's splitting indistinguishability property.

11. We unpuncture F_A and F_B at $(i, 0)$ and un-hardcode $\mathsf{vk}^A_{i,0}$ and $\mathsf{vk}^B_{i,0}$. This is functionally equivalent and hence indistinguishable by $\mathsf{i}\mathcal{O}$.

12. We repeat steps 1 through 11 for timestamps 1 through the worst-case running time bound T instead of just for timestamp 0 as was described above. In this way, we progressively change the computation from using δ^0_i (M^0_i's transition function) to δ^1_i (M^1_i's transition function), starting at the beginning of the computation. □

5.6 Fixed-Access Garbling

Fixed-access security is defined in the same way as fixed-transcript security, but the left and right machines produced by $\mathcal{A}$ do not need to have the same transcripts for $\mathcal{A}$ to win - they may not have the same intermediate states, but only need to perform the same memory operations.

Definition 5.8 (Fixed-access security). *We define fixed-access security via the following game.*

1. *The challenger samples* $SK \leftarrow \mathsf{Setup}(1^\lambda, S)$ *and* $b \leftarrow \{0, 1\}$.

Input: Time t, state q, iterator itr, accumulator ac, operation op, signature σ, memory value v, new accumulator ac', proof π

Data: Puncturable PRFs F_A and F_B, RAM machine M_i with transition function δ_i, Accumulator verification key $\mathsf{vk}_{\mathsf{Acc}}$, index i, iterator public parameters $\mathsf{Itr.PP}$, time bound T_i

$(\mathsf{sk}_A, \mathsf{vk}_A) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_A(i,t))$;
$(\mathsf{sk}_B, \mathsf{vk}_B) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_B(i,t))$;
if $t > T_i$ *or* $\mathsf{Acc.Verify}(\mathsf{vk}_{\mathsf{Acc}}, \mathsf{ac}, \mathsf{op}, \mathsf{ac}', v, \pi) = 0$ **then return** $\perp$;
if $\mathsf{Spl.Verify}(\mathsf{vk}_A, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}), \sigma) = 1$ **then** track:='A';
else if $\mathsf{Spl.Verify}(\mathsf{vk}_B, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}), \sigma) = 1$ **then** track:='B';
else return $\perp$;
$\mathsf{out} \leftarrow \delta_i(q, v)$;
if $\mathsf{out} \in Y$ **then**
    $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_{track}(i+1, 0))$;
    return $\mathsf{out}, \mathsf{Sign}(\mathsf{sk}', (\perp, \perp, \mathsf{ac}', \mathsf{ReadWrite}(0 \mapsto 0)))$
else
    Parse out as (q', op');
    $\mathsf{itr}' \leftarrow \mathsf{Itr.Iterate}(\mathsf{Itr.PP}, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}))$;
    $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_{track}(i, t+1))$;
    return $(q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'), \mathsf{Sign}(\mathsf{sk}', (q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'))$

Algorithm 12: Transition function for hybrid M_i, with memory verified by an accumulator.

Input: Time t, state q, iterator itr, accumulator ac, operation op, signature σ, memory value v, new accumulator ac', proof π

Data: Puncturable PRFs F_A and F_B, RAM machine M_j^0 with transition function δ_j^0, Accumulator verification key $\mathsf{vk}_{\mathsf{Acc}}$, index i, iterator public parameters $\mathsf{Itr.PP}$, accumulator ac_{s_j}, time bound T_i.

$(\mathsf{sk}_A, \mathsf{vk}_A) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_A(i,t))$;
$(\mathsf{sk}_B, \mathsf{vk}_B) \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_B(i,t))$;
if $t > T_i$ *or* $\mathsf{Acc.Verify}(\mathsf{vk}_{\mathsf{Acc}}, \mathsf{ac}, \mathsf{op}, \mathsf{ac}', v, \pi) = 0$ **then return** $\perp$;
if $\mathsf{Spl.Verify}(\mathsf{vk}_A, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}), \sigma) = 1$ **then** track:='A';
else if $\mathsf{Spl.Verify}(\mathsf{vk}_B, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}), \sigma) = 1$ **then** track:='B';
else return $\perp$;
$\mathsf{out} \leftarrow \delta_j^0(q, v)$;
if $\mathsf{out} \in Y$ **then**
    $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_{track}(i+1, 0))$;
    return $\mathsf{out}, \mathsf{Sign}(\mathsf{sk}', (\perp, \perp, \mathsf{ac}_{s_j}, \mathsf{ReadWrite}(0 \mapsto 0))$
else
    Parse out as (q', op');
    $\mathsf{itr}' \leftarrow \mathsf{Itr.Iterate}(\mathsf{Itr.PP}, (q, \mathsf{itr}, \mathsf{ac}, \mathsf{op}))$;
    $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{Spl.Setup}(1^\lambda; F_{track}(i, t+1))$;
    return $(q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'), \mathsf{Sign}(\mathsf{sk}', (q', \mathsf{itr}', \mathsf{ac}', \mathsf{op}'))$

Algorithm 13: Response to j^{th} program query, with hard-coded final accumulator value.

2. *The adversary sends a memory configuration s to the challenger. The challenger sends back* $\mathsf{GbMem}(SK, s)$.

3. *The adversary repeatedly sends pairs of RAM programs* (M_i^0, M_i^1) *to the challenger, together with a time bound* 1^{T_i}*, and the challenger sends back* $\tilde{M}_i^b \leftarrow \mathsf{GbPrg}(SK, M_i^b, T_i, i)$. *Each pair* (M_i^0, M_i^1) *is chosen adaptively after seeing* $\tilde{M}_{i-1}^b$.

4. *The adversary outputs a guess* b'.

Let $((M_1^0, M_1^1), \ldots, (M_\ell^0, M_\ell^1))$ *denote the sequence of pairs of machines output by the adversary. The adversary is said to win if* $b' = b$ *and:*

- *Sequentially executing* $M_1^0, \ldots, M_\ell^0$ *on initial memory configuration* s *yields the same transcript as executing* $M_1^1, \ldots, M_\ell^1$*, except that the local states can be different.*
- *Each* M_i^b *runs in time at most* T_i *and space at most* S.

A garbling scheme is said to have fixed-access security if all p.p.t. adversaries $\mathcal{A}$ *win in the game above with probability less than* $1/2 + \mathrm{negl}(\lambda)$.

To achieve fixed-access security, we adapt the exact same technique from [CH16]: xor-ing the state with a pseudorandom function applied on the local time t. The PRF keys used in different machines are sampled independently.

Theorem 5.9. *If there is a fixed-transcript garbling scheme, then there is a fixed-access garbling scheme.*

Proof. From a fixed-transcript garbling scheme $(\mathsf{Setup}', \mathsf{GbMem}', \mathsf{GbPrg}', \mathsf{Eval}')$, we construct a fixed-access garbling scheme $(\mathsf{Setup}, \mathsf{GbMem}, \mathsf{GbPrg}, \mathsf{Eval})$.

$\mathsf{Setup}(1^\lambda, S)$ samples $SK' \leftarrow \mathsf{Setup}'(1^\lambda, S)$, sets it as SK.

$\mathsf{GbMem}(SK, s)$ outputs $\tilde{s}' \leftarrow \mathsf{GbMem}'(SK', s)$.

$\mathsf{GbPrg}(SK, M_i, T_i, i)$ samples a PPRF F_i, outputs $\tilde{M}_i' \leftarrow \mathsf{GbPrg}'(SK', M_i', T_i, i)$, where M_i' is defined as in Algorithm 14. If M_i's initial state is q_0, the initial state of M_i' is $(0, q_0 \oplus F_i(0))$.

$\mathsf{Eval}(\tilde{M}, \tilde{s})$ outputs $\mathsf{Eval}'(\tilde{M}', \tilde{s}')$.

We introduce hybrid games H_ℓ through H_0, starting with the real security game, and ending with one in which the adversary's view is independent of b. In hybrid H_i, the j^{th} query (M_i^0, M_i^1) is answered with $\tilde{M}_i^b$ if $j \leq i$ and $\tilde{M}_i^0$ otherwise. It remains to show that hybrid H_i is indistinguishable from H_{i+1}.

To show this, we introduce intermediate hybrids $\{H_{i,j}\}_{j=0,\ldots,T_i}$, each of which differs from H_i only in the answer to the i^{th} query. In $H_{i,j}$, the answer to the i^{th} machine query is answered by $\mathsf{GbPrg}'(SK', M_{i,j}', T_i, i)$, where the machine $M_{i,j}'$ is defined in Algorithm 15. Informally, $M_{i,j}'$ executes M_i^b for the first $T_i - j$ steps, and executes the next j steps with machine M_i^0.

Input: State (t, c_q), memory symbol σ
Data: RAM machine M_i, puncturable PRF F_i

$q \leftarrow c_q \oplus F_i(t)$;
$\mathsf{out} \leftarrow M_i(q, \sigma)$;
if $\mathsf{out} \in Y$ **then return** out;
Parse out as (q', op);
return $((t+1, q' \oplus F_i(t+1)), \mathsf{op})$;

Algorithm 14: M'_i, the modified version of M_i which encrypts its state.

Input: State (t, c_q), memory symbol σ
Data: RAM machines M_i^0, M_i^1, punctured PRF $F'_i = F_i\{T_i - j\}$, hard-coded state q^*, hard-coded ciphertext c^*, bit b

if $t = T_i - j$ **then** $q \leftarrow q^*$;
else $q \leftarrow c_q \oplus F'_i(t)$;
if $t < T_i - j$ **then** $M_i \leftarrow M_i^b$;
else $M_i \leftarrow M_i^0$;
$\mathsf{out} \leftarrow M_i(q, \sigma)$;
if $\mathsf{out} \in Y$ **then return** out;
Parse out as (q', op);
if $t = T_i - j - 1$ **then return** $((t+1, c^*), \mathsf{op})$;
else return $((t+1, q' \oplus F'_i(t+1)), \mathsf{op})$;

Algorithm 15: $M'_{i,j}$ executes M_i^b for $t_i - j$ steps, and then executes M_i^0.

Claim 5.9.1. *$H_i \approx H_{i,0}$ and $H_{i-1} \approx H_{i,T_i}$.*

Proof. This follows from the underlying fixed-transcript garbling. □

Claim 5.9.2. *For every $j \in \{0, \ldots, T_i - 1\}$, $H_{i,j} \approx H_{i,j+1}$*

Proof. We introduce another intermediate hybrid $H_{i,j,0}$, in which $c^* = q^1_{i,T_i-j} \oplus F_i(T_i - j)$. The indistinguishability of $H_{i,j}$ and $H_{i,j,0}$ follows from the pseudorandomness of the (selectively) puncturable PRF F_i on $T_i - j$. The indistinguishability of $H_{i,j,0}$ and $H_{i,j+1}$ follows from the underlying fixed-transcript garbling. So we have shown that $H_{i,j} \approx H_{i,j,0} \approx H_{i,j+1}$. □

This completes the proof of Theorem 5.9. □

5.7 Fixed-Address Garbling

Fixed-address security is defined in the same way as fixed-access security, but the left and right machines produced by $\mathcal{A}$ do not need to make the same memory operations for $\mathcal{A}$ to win - their memory operations only need to access the same addresses. Additionally, the adversary $\mathcal{A}$ now provides not only a single memory configuration s_0, but two memory configurations s^0_0 and s^1_0. The challenger returns $\mathsf{GbMem}(SK, s^b_0)$. In keeping with the spirit of fixed-address garbling, we require s^0_0 and s^1_0 to have the same set of addresses storing non-ϵ values.

Definition 5.10 (Fixed-address security)**.** *We define fixed-address security via the following game.*

1. *The challenger samples $SK \leftarrow \mathsf{Setup}(1^\lambda, S)$ and $b \leftarrow \{0, 1\}$.*
2. *The adversary sends the initial memory configurations s^0_0, s^1_0 to the challenger. The challenger sends back $\tilde{s}^b_0 \leftarrow \mathsf{GbMem}(SK, s^b_0)$.*
3. *The adversary repeatedly sends pairs of RAM programs (M^0_i, M^1_i) to the challenger, together with a time bound 1^{T_i}, and the challenger sends back $\tilde{M}^b_i \leftarrow \mathsf{GbPrg}(SK, M^b_i, T_i, i)$. Each pair (M^0_i, M^1_i) is chosen adaptively after seeing $\tilde{M}^b_{i-1}$.*
4. *The adversary outputs a guess b'.*

Let $((s^0_0, s^1_0), (M^0_1, M^1_1), \ldots, (M^0_\ell, M^1_\ell))$ denote the sequence of pairs of memory configurations and machines output by the adversary. The adversary is said to win if $b' = b$ and:

- $\{a : s^0_0(a) \neq \epsilon\} = \{a : s^1_0(a) \neq \epsilon\}$.
- *The sequence of addresses accessed and the outputs during the sequential execution of $M^0_1, \ldots, M^0_\ell$ on initial memory configuration s^0_0 is the same as from executing $M^1_1, \ldots, M^1_\ell$ on s^1_0.*

- *Each M_i^b runs in time at most T_i and space at most S.*
- $|M_i^0| = |M_i^1|$, $i = 1, \ldots, \ell$.

A garbling scheme is said to have fixed-address security if all p.p.t. adversaries $\mathcal{A}$ win in the game above with probability less than $1/2 + \mathrm{negl}(\lambda)$.

Our construction of fixed-address garbling is almost the same with the two-track solution in [CH16], with a slight modification at the way to "encrypt" the memory configuration. In [CH16], the memory configurations are xor-ed with different puncturable PRF values in the two tracks, where the PRFs are applied on the time t and address a. In this work, the PRFs are applied on the execution index i and time t, not on the address a. This is enough for our purpose, because in each execution index i and step t, the machine only writes on a single address (for the initial memory configuration, the index is assigned as 0, and different timestamps will be assigned on different addresses). By this modification, we are able to prove adaptive security based on selective secure puncturable PRF, and adaptively secure fixed-access garbling.

We note that, even if the address a is included in the domain of PRF, as in [CH16], the construction is still adaptively secure if the underlying PRF is based on GGM's tree construction. Here we choose to present the simplified version which suffices for our purpose.

Construction 5.11. *Suppose* $(\mathsf{Setup}', \mathsf{GbMem}', \mathsf{GbPrg}', \mathsf{Eval}')$ *is a fixed-access garbling scheme, we construct a fixed-address garbling scheme* $(\mathsf{Setup}, \mathsf{GbMem}, \mathsf{GbPrg}, \mathsf{Eval})$*:*

$\mathsf{Setup}(1^\lambda)$ *samples* $SK' \leftarrow \mathsf{Setup}'(1^\lambda)$ *and puncturable PRFs* F_A *and* F_B.

$\mathsf{GbMem}(SK, s)$ *outputs* $\tilde{s}'_0 \leftarrow \mathsf{GbMem}'(SK', s'_0)$*, where*

$$s'_0(a) = \begin{cases} (0, -a, F_A(0,-a) \oplus s_0(a), F_B(0,-a) \oplus s_0(a)) & \text{if } s_0(a) \neq \epsilon \\ \epsilon & \text{otherwise} \end{cases}$$

$\mathsf{GbPrg}(SK, M_i, T_i, i)$ *outputs* $\tilde{M}'_i \leftarrow \mathsf{GbPrg}'(SK', M'_i, T_i, i)$*, where* M'_i *is defined as in Algorithm 16. If the initial state of* M_i *was* q_0*, the initial state of* M'_i *is* $(0, q_0, q_0)$.

$\mathsf{Eval}(\tilde{M}, \tilde{s})$ *outputs* $\mathsf{Eval}'(\tilde{M}', \tilde{s}'_0)$.

Theorem 5.12. *If* $(\mathsf{Setup}', \mathsf{GbMem}', \mathsf{GbPrg}')$ *is a fixed-access garbling scheme, then Construction 5.11 is a fixed-address garbling scheme.*

Proof. We give a sequence of hybrid games, starting with the real game H^b, and ending with one in which the adversary's view is independent of b. We show that the adversary's advantage differs negligibly in each pair of adjacent games. This

Input: State (t_q, q_A, q_B), memory symbol $(i_{in}, t_{in}, c_A, c_B)$
Data: RAM machine M_i, puncturable PRFs F_A, F_B
1 out $\leftarrow M_i(q_A, F_A(i_{in}, t_{in}) \oplus c_A)$;
2 **if** out $\in Y$ **then return** out;
3 Parse out as $(q', \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'))$;
4 $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v', F_B(i, t_q) \oplus v')$;
5 **return** $(t_q + 1, q', q'), \mathsf{op}'$;

Algorithm 16: M_i': Modified version of M_i which encrypts its memory twice in parallel.

will imply that in the real security game, all adversaries have advantage at most $1/2 + \mathrm{negl}(\lambda)$.

The hybrid structure follows closely from [CH16]. Purely for ease of informal exposition, we think of the machines $M_1^b, \ldots, M_\ell^b$ as being concatenated into one RAM machine $M = M^b$ with running time at most T. Recall that in our construction, if M^b would write v_t^b to address a at time t, then $\tilde{M}$ writes $(F_A(t) \oplus v_t^b, F_B(t))$ to a. Our hybrids make the following changes to the way in which the challenger generates $\tilde{M}$ and $\tilde{s}_0$:

1. $\tilde{s}_0$ is now defined as

$$\tilde{s}_0(a) = \begin{cases} (F_A(-a) \oplus s_0^b(a), F_B(-a) \oplus s_0^0(a)) & \text{if } s_0^b(a) \neq \epsilon \\ \epsilon & \text{otherwise} \end{cases}$$

   This is indistinguishable by the puncturable PRF security of F_B, because the contents on "B"-track are not decrypted at all in the real garbled program.

2. Let $v_1^b, \ldots, v_T^b$ denote the values that M^b would write when executed on s^b. For $i = 1, \ldots, T$, we have a hybrid in which:

   - On timesteps $t < i$, $\tilde{M}$ writes $(F_A(t) \oplus v_t^b, F_B(t) \oplus v_t^0)$.
   - On subsequent timesteps, $\tilde{M}$ writes $(F_A(t) \oplus v_t^b, F_B(t))$.

   Here, the addresses which $\tilde{M}$ accesses are determined by the implicit internal execution of M^b.

   These hybrids are indistinguishable by puncturable PRF security together with fixed-access security: one can freely puncture F_B at i and hard-code its value because no other point in the computation uses $F_B(i)$.

3. Now that M^b and M^0 are both being implicitly executed in parallel, we determine where $\tilde{M}$ writes by following M^0. This is indistinguishable by fixed-access security because M^0 and M^b access the same addresses.

4. Symmetrically to step 1, we define another sequence of hybrids for $i = T, \ldots, 1$, in which:

- On timesteps $t < i$, $\tilde{M}$ writes $(F_A(t) \oplus v_t^b, F_B(t) \oplus v_t^0)$.
- On subsequent timesteps, $\tilde{M}$ writes $(F_A(t), F_B(t) \oplus v_t^0)$.

5. Finally, we remove M^b from $\tilde{M}$ altogether. As it is no longer used, this change is indistinguishable by security of the fixed-access garbling scheme. Thus, in this hybrid the adversary's view is independent of b.

Formally we define the hybrids with full exposition.

Hybrids $H_{i,z,b,0}$ In $H_{i,z,b,0}$, where the subscripts represent the execution index $i \in \{1, 2, ..., \ell\}$, timestep $z \in \{0, 1, ..., T_i\}$, initial memory configuration on track-"A" and "B" being the encryption of s_0^b and s_0^0.

1. The challenger samples $SK' \leftarrow \mathsf{Setup}'(1^\lambda, S)$ and $b \leftarrow \{0, 1\}$.

2. The adversary sends the initial memory configurations s_0^0, s_0^1 to the challenger. The challenger sends back $\tilde{s}'_{0,0,b,0} \leftarrow \mathsf{GbMem}'(SK', s'_{0,0,b,0})$, where $s'_{0,0,b,0}(a)$ is constructed as:

$$s'_{0,0,b,0}(a) = \begin{cases} (0, -a, F_A(0, -a) \oplus s_0^b(a), F_B(0, -a) \oplus s_0^0(a)) & \text{if } s_0^b(a) \neq \epsilon \\ \epsilon & \text{otherwise} \end{cases}$$

3. The adversary sends pairs of RAM programs $(M_1^0, M_1^1), \ldots, (M_\ell^0, M_\ell^1)$ to the challenger, each pair chosen adaptively after seeing the garbling of previous programs. In the RAM machine $M'_{i,z,b,0}$ defined by algorithm 17, for the first z steps, the resulting memory configurations of M_i^b evaluated on s_{i-1}^b are written on track A, those of M_i^0 evaluated on s_{i-1}^0 are written on track B; for the next $T_i - z$ steps, the resulting memory configuration of M_i^b on s_{i-1}^b is written on both tracks.

   The response of the challenger is the fixed-access garbling of machine $M'_{j,z,b,0}$, set up in different ways depending on the relation of j and i:

   (a) For $j \in \{1, \ldots, i-1\}$, the challenger sends back

$$\tilde{M}'_{j,T_1,b,0} \leftarrow \mathsf{GbPrg}'(SK, M'_{j,T_1,b,0}, j);$$

   (b) For $j = i$, the challenger sends back

$$\tilde{M}'_{i,z,b,0} \leftarrow \mathsf{GbPrg}'(SK, M'_{i,z,b,0}, i);$$

   (c) For $j \in \{i+1, \ldots, \ell\}$, the challenger sends back

$$\tilde{M}'_{j,0,b,0} \leftarrow \mathsf{GbPrg}'(SK, M'_{j,0,b,0}, j).$$

4. The adversary outputs a guess b'.

Input: State (t_q, q_A, q_B), memory symbol $(i_{in}, t_{in}, c_A, c_B)$
Data: i, z, RAMs M_i^b, M_i^0, PPRFs F_A, F_B
$\mathsf{out}_A \leftarrow M_i^b(q_A, F_A(i_{in}, t_{in}) \oplus c_A)$;
if $\mathsf{out}_A \in Y$ **then return** out_A;
Parse out_A as $(q'_A, \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'_A))$;
if $t_q < z$ **then**
  $\mathsf{out}_B \leftarrow M_i^0(q_B, F_B(i_{in}, t_{in}) \oplus c_B)$;
  Parse out_B as $(q'_B, \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'_B))$;
  $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v'_A, F_B(i, t_q) \oplus v'_B)$;
else
  $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v'_A, F_B(i, t_q) \oplus v'_A)$;
return $(t_q + 1, q'_A, q'_B), \mathsf{op}'$;

Algorithm 17: $M'_{i,z,b,0}$

Lemma 5.13. $H^b \approx H_{1,0,b,0}$.

Proof. The initial memory configurations in H^b and $H_{0,0,b,0}$ differ in the "B"-track. Because $M_1'^b$ and $M'_{1,0,b,0}$ don't "decrypt" the contents in the "B"-track, $s^b(a) \oplus F_B(0, -a)$ and $s^0(a) \oplus F_B(0, -a)$ are indistinguishable by the pseudorandomness of F_B on $(0, -a)$, $a \in \{a : s^0(a) \neq \epsilon\}$.

The RAM machine $M_1'^b$ and $M'_{1,0,b,0}$ have the exact same functionality and are accessing the same memory configuration, so the garbling of them are indistinguishable following the fixed-access security. □

Lemma 5.14. *For* $i \in \{2, ..., \ell\}$, $H_{i-1,T_{i-1},b,0} \approx H_{i,0,b,0}$.

Proof. This follows directly from the underlying fixed-access security. □

Lemma 5.15. *For* $i \in \{1, 2, ..., \ell\}$, $z \in \{0, 1, ..., T_i - 1\}$, $H_{i,z,b,0} \approx H_{i,z+1,b,0}$.

Proof. For each i and z, we introduce one more intermediate hybrid $H_{i,z,b,0,0}$, where the adversary receives

$$\tilde{s}'_{0,0,b,0},\ \tilde{M}'_{1,T_1,b,0},\ ...,\ \tilde{M}'_{i-1,T_{i-1},b,0},\ \tilde{M}'_{i,z,b,0,0},\ \tilde{M}'_{i+1,0,b,0},\ ...,\ \tilde{M}'_{\ell,0,b,0}.$$

The RAM machine $M'_{i,z,b,0,0}$ is defined by Algorithm 18. The hard-coded ciphertext c^* in $H_{i,z,b,0,0}$ is $F_B(i, z) \oplus v'_B$. The difference of $H_{i,z,b,0}$ and $H_{i,z,b,0,0}$ are

1. The i^{th} RAM machine $M'_{i,z,b,0}$ versus $M'_{i,z,b,0,0}$.

2. In the other RAM machines, F_B is also punctured on (i, z). Note that this won't change the functionality of $M'_{1,T_1,b,0}$, ..., $M'_{i-1,T_{i-1},b,0}$, $M'_{i+1,0,b,0}$, ..., $M'_{\ell,0,b,0}$, since the first $i-1$ machines won't read or write with index i, and the last $\ell - i$ ones won't read "B"-track or write with index i.

Input: State (t_q, q_A, q_B), memory symbol $(i_{in}, t_{in}, c_A, c_B)$
Data: i, z, RAMs M_i^b, M_i^0, PPRFs F_A, $F'_B = F_B\{i, z\}$, ciphertext c^*.

$\mathsf{out}_A \leftarrow M_i^b(q_A, F_A(i_{in}, t_{in}) \oplus c_A)$;
if $\mathsf{out}_A \in Y$ **then return** out_A;
Parse out_A as $(q'_A, \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'_A))$;
if $t_q < z$ **then**
  $\mathsf{out}_B \leftarrow M_i^0(q_B, F'_B(i_{in}, t_{in}) \oplus c_B)$;
  Parse out_B as $(q'_B, \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'_B))$;
  $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v'_A, F'_B(i, t_q) \oplus v'_B)$;
else if $t_q = z$ **then**
  $\mathsf{out}_B \leftarrow M_i^0(q_B, F'_B(i_{in}, t_{in}) \oplus c_B)$;
  Parse out_B as $(q'_B, \mathsf{ReadWrite}(\mathsf{addr}' \mapsto v'_B))$;
  $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v'_A, c^*)$;
else
  $\mathsf{op}' := \mathsf{ReadWrite}(\mathsf{addr}' \mapsto (i, t_q, F_A(i, t_q) \oplus v'_A, F'_B(i, t_q) \oplus v'_A)$;
return $(t_q + 1, q'_A, q'_B), \mathsf{op}'$;

Algorithm 18: $M'_{i,z,b,0,0}$.

Note that $H_{i,z,b,0,0} \approx H_{i,z+1,b,0}$ by the underlying fixed-access garbling. If we define $c^* = F_B(i, z) \oplus v'_A$, the RAM machines has the same functionality with those in hybrid $H_{i,z,b,0}$. By the pseudorandomness of the punctured PRF F_B on (i, z), $H_{i,z,b,0,0} \approx H_{i,z,b,0}$. This shows that $H_{i,z,b,0} \approx H_{i,z,b,0,0} \approx H_{i,z+1,b,0}$. □

Combining Lemmas 5.13 to 5.15, we obtain that $H^b \approx H_{1,0,b,0} \approx ... \approx H_{\ell,0,b,0} \approx H_{\ell,T_\ell,b,0}$.

The rest of the proof can be done symmetrically: First, instead of returning out_A, return out_B, by the underlying fixed-access garbling. Then switch the computation on "A"-track from running M^b on s^b into running M^0 on s^0, and prove the indistinguishability of them analogously via the puncturability F_A and the underlying fixed-access security. Finally b is not in the view of the adversary. □

5.8 Full Garbling

In order to construct a fully secure garbling scheme, we will need to make use of an oblivious RAM to hide the addresses accessed by the machine.

5.8.1 Oblivious RAMs with Strong Localized Randomness

An ORAM is a probabilistic scheme for memory storage and access that provides obliviousness for access patterns with sublinear access complexity. It is convenient for us to model an ORAM scheme as follows. We define a deterministic algorithm OProg so that for a security parameter 1^λ, a memory operation op, and a space

bound S, $\mathsf{OProg}(1^\lambda, \mathsf{op}, S)$ outputs a probabilistic RAM machine M_{op}. More generally, for a RAM machine M, we can define $\mathsf{OProg}(1^\lambda, M, S)$ as one which executes $\mathsf{OProg}(1^\lambda, \mathsf{op}, S)$ for every operation op output by M.

We also define OMem, a procedure for making a memory configuration oblivious, in terms of OProg, as follows: Given a memory configuration s with n non-empty addresses $a_1, \ldots, a_n$, all less than or equal to a space bound S, $\mathsf{OMem}(1^\lambda, s, S)$ iteratively samples

$$s'_0 \leftarrow \epsilon^{\mathbb{N}}$$

and

$$s'_i = \mathsf{NextMem}(\mathsf{OProg}(1^\lambda, \mathsf{ReadWrite}(a_i \mapsto s(a_i)), S), s'_{i-1})$$

and outputs s'_n.

Correctness An ORAM is said to be correct if for all memory operations $\mathsf{op}_1, \ldots, \mathsf{op}_\ell$ accessing addresses less than or equal to S, it holds with high probability that

$$(M_{\mathsf{op}_1}; \ldots; M_{\mathsf{op}_\ell})(\epsilon^{\mathbb{N}}) = (\mathsf{op}_1; \ldots; \mathsf{op}_\ell)(\epsilon^{\mathbb{N}})$$

That is, when one sequentially executes $M_{\mathsf{op}_1}, \ldots, M_{\mathsf{op}_\ell}$ on the initially empty memory, M_{op_ℓ} outputs the same result as op_n when executing $\mathsf{op}_1, \ldots, \mathsf{op}_\ell$ from the initially empty memory.

Efficiency An ORAM is said to have multiplicative space overhead $\zeta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ if for all memory operations op accessing an address less than or equal to a space bound S, and for all memory configurations s, it holds with probability 1 that

$$\mathsf{Space}(M_{\mathsf{op}}, s) \leq \zeta(S, \lambda) \cdot S$$

An ORAM is said to have multiplicative time overhead $\eta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ if for all memory operations op and all memory configurations s, it holds with probability 1 that

$$\mathsf{Time}(M_{\mathsf{op}}, s) = \eta(S, \lambda).$$

Security (Strong Localized Randomness). We now define a notion of strong localized randomness[3] for an ORAM, which is satisfied by the ORAM construction of [CP13].

Informally, we consider obliviously executing operations $\mathsf{op}_1, \ldots, \mathsf{op}_t$ on a memory of size S, i.e. executing machines $M_{\mathsf{op}_1}; \ldots; M_{\mathsf{op}_t}$ using a random tape $R \in \{0,1\}^{\mathbb{N}}$. This yields a sequence of addresses $\vec{A} = \vec{a}_1 \| \cdots \| \vec{a}_t$. There should be a natural way to decompose each $\vec{a}_i$ (in the Chung-Pass ORAM, we consider each recursive level of the construction) such that we can write $\vec{a}_i = \vec{a}_{i,1} \| \cdots \| \vec{a}_{i,m}$. Our notion of strong localized randomness requires that (after having fixed $\mathsf{op}_1, \ldots, \mathsf{op}_t$), each $\vec{a}_{i,j}$ depends on some small substring of R, which does not influence any other $\vec{a}_{i',j'}$. In other words:

- There is some $\alpha_{i,j}, \beta_{i,j} \in \mathbb{N}$ such that $0 < \beta_{i,j} - \alpha_{i,j} \leq \mathrm{poly}(\log S)$ and such that $\vec{a}_{i,j}$ is a function of $R_{\alpha_{i,j}}, \ldots, R_{\beta_{i,j}}$.

[3]This notion is similar but stronger to the "localized randomness" defined in [CH16]

- The collection of intervals $[\alpha_{i,j}, \beta_{i,j})$ for $i \in \{1, \ldots, t\}$, $j \in \{1, \ldots, m\}$ is pairwise disjoint.

Formally, we say that an ORAM with multiplicative time overhead η has strong localized randomness if:

- For all λ and S, there exists m and $\tau_1 < \tau_2 < \cdots < \tau_m$ with $\tau_1 = 1$ and $\tau_m = \eta(S, \lambda) + 1$, and there exist circuits $C_1, \ldots, C_m$, such that for all memory operations $\mathsf{op}_1, \ldots, \mathsf{op}_t$, there exist pairwise disjoint intervals $I_1, \ldots, I_m \subset \mathbb{N}$ such that:
  - If we write
    $$\vec{A}_1 \| \cdots \| \vec{A}_t \leftarrow \mathsf{addr}(M_{\mathsf{op}_1}^{R_1}; \ldots; M_{\mathsf{op}_t}^{R_t}, \epsilon^{\mathbb{N}})$$
    where $R = R_1 \| \cdots \| R_t$ denotes the randomness used by the oblivious accesses and each $\vec{A}_i$ denotes the addresses accessed by $M_{\mathsf{op}_i}^{R_i}$, then $(\vec{A}_t)_{[\tau_j, \tau_{j+1})} = C_j(R_{I_j})$ with high probability over R. Here R_{I_j} denotes the contiguous substring of R indexed by the interval $I_j \subset [|R|]$.
  - With high probability over the choice of $R_{\mathbb{N} \setminus I_j}$, $\vec{A}_1, \ldots, \vec{A}_{t-1}$ does not depend on R_{I_j} as a function.
- τ_j and the circuits C_j are computable in polynomial time given 1^λ, S, and j.
- I_j is computable in polynomial time given 1^λ, S, $\mathsf{op}_1, \ldots, \mathsf{op}_t$, and j.

Definition 5.16. *An S-ORAM is an ORAM where correctness, efficiency, and security need hold only if the space bound is at most S.*

Claim 5.16.1. *For any c, there is a c-ORAM with multiplicative space overhead of 1 and multiplicative time overhead of c.*

Proof. This is just the brute-force ORAM which accesses the entire memory for each underlying memory operation. Since this ORAM is deterministic, the localized randomness property is trivial. □

Claim 5.16.2. *There is an ORAM with polylogarithmic time and space overhead and localized randomness.*

Proof. Suppose we have an S-ORAM with multiplicative space overhead ζ and time overhead η. We show how to build a $2S$-ORAM with multiplicative space overhead $\zeta'(N) = \zeta(N) + \mathrm{poly}(\log N, \lambda)$, and multiplicative time overhead $\eta'(N) = \eta(N) + \mathrm{poly}(\log N, \lambda)$. Our base case will be the brute-force ORAM.

Next we construct an ORAM with strong localized randomness.

Construction 5.17 (Chung-Pass). *Given a memory operation* $\mathsf{op} = \mathsf{ReadWrite}(a \mapsto v')$ *with alphabet* Σ*, we construct a RAM machine* M_{op}*, which we describe with pseudocode:* M_{op}*'s view of memory has two parts:*

- *A memory* $\mathcal{M}$ *with* $\zeta(S) \cdot S$ *addresses, which we think of as a smaller ORAM.*

- *A complete binary tree $\mathcal{T}$ of depth $\log S$. Each node in $\mathcal{T}$ is a bucket with capacity sufficient to hold λ tuples of the form $(\mathsf{addr}, \mathsf{pos}, \sigma_0, \sigma_1) \in [S] \times [S] \times \Sigma \times \Sigma$.*

M_{op} does the following:

1. *Sample a random position pos', and execute $\mathsf{pos} \leftarrow \mathsf{OProg}(1^\lambda, \mathsf{ReadWrite}(\lfloor \frac{a}{2} \rfloor \mapsto \mathsf{pos}'), S)$ on $\mathcal{M}$.*
2. *On the path from the root of $\mathcal{T}$ to the pos^{th} leaf, search for a tuple $(\mathsf{addr}, \mathsf{pos}, \sigma_0, \sigma_1)$ such that $\mathsf{addr} = \lfloor \frac{a}{2} \rfloor$. If such a tuple is found, record σ_0 and σ_1, and then overwrite the tuple with $\perp$.*
3. *Add a tuple $(\lfloor \frac{a}{2} \rfloor, \mathsf{pos}', \sigma_0', \sigma_1')$ to the root bucket, where*

$$\sigma_b' = \begin{cases} v' & \text{if } b \equiv a \pmod 2 \\ \sigma_b & \text{otherwise} \end{cases}$$

4. *Traverse a path from the root to a random leaf of T, moving every encountered tuple $(\mathsf{addr}, \mathsf{pos}, \sigma_0, \sigma_1)$ to the deepest node on the path that is a prefix of pos.*
5. *Return $\sigma_{a \bmod 2}$.*

Correctness and efficiency of Construction 5.17 are easy to see, assuming the following lemma, which is proved in [CP13].

Lemma 5.18. *For all memory operations $\mathsf{op}_1, \ldots, \mathsf{op}_t$, with high probability, Construction 5.17 will not exceed the capacity of any of its buckets.*

Claim 5.18.1. *Construction 5.17 has strong localized randomness.*

Proof. We show how each of the chunks of addresses accessed by Construction 5.17 are functions of prior contiguous chunks of randomnesses.

- In step 1, we access the addresses that the S-ORAM would access; hence we get localized randomness for free.
- In step 2, we access all nodes on the path to pos, where pos was retrieved from the S-ORAM in step 1. pos was chosen at random and written to the S-ORAM on the last time that M accessed a (or a' with $\lfloor \frac{a'}{2} \rfloor = \lfloor \frac{a}{2} \rfloor$).
- In step 4, we simply choose a fresh random path and access all of the nodes on that path. □

□

Remark 5.19. A more usual definition of obliviousness requires that if two machines M_0 and M_1 have the same running time, then the addresses accessed by $\mathsf{OProg}(M_0)$ and $\mathsf{OProg}(M_1)$ will be statistically close. Our definition of strong localized randomness in fact implies this definition.

5.8.2 Full Garbling Construction

Theorem 5.20. *If there is an efficient fixed-address garbling scheme, then there is an efficient full garbling scheme.*

Proof. Given a fixed-address garbling scheme $(\mathsf{Setup}', \mathsf{GbMem}', \mathsf{GbPrg}', \mathsf{Eval}')$ and an oblivious RAM OProg with space overhead ζ and time overhead η. We construct a full garbling scheme $(\mathsf{Setup}, \mathsf{GbMem}, \mathsf{GbPrg}, \mathsf{Eval})$.

$\mathsf{Setup}(1^\lambda, T, S)$ samples $SK' \leftarrow \mathsf{Setup}'(1^\lambda, \eta(S,\lambda)\cdot T, \zeta(S,\lambda)\cdot S)$ and samples a PPRF $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{\ell_R}$, where ℓ_R is the length of randomness needed to obliviously execute one memory operation. We will sometimes think of the domain of F as $[2^{2\lambda}]$.

$\mathsf{GbMem}(\mathsf{SK}, s_0)$ outputs $\tilde{s}'_0 \leftarrow \mathsf{GbMem}'(\mathsf{SK}', \mathsf{OMem}(1^\lambda, s_0, S))$.

$\mathsf{GbPrg}(\mathsf{SK}, M_i, i)$ outputs $\tilde{M}'_i \leftarrow \mathsf{GbPrg}'(\mathsf{SK}', \mathsf{OProg}(1^\lambda, M_i, S)^{F(i,\cdot)}, i)$.

$\mathsf{Eval}(\tilde{M}, \tilde{s})$ outputs $\mathsf{Eval}'(\tilde{M}', \tilde{s}'_0)$.

Simulator To show security of this construction, we define the following simulator.

1. The adversary provides S, and an initial memory configuration s_0. Say that s_0 has n non-ϵ addresses. The simulator is given S and n, and samples $SK' \leftarrow \mathsf{Setup}'(1^\lambda, \zeta(S,\lambda)\cdot S)$ and sends $\mathsf{GbMem}'(SK', \mathsf{OMem}(1^\lambda, 0^n, S))$ to the adversary.
2. When the adversary makes a query $M_i, 1^{T_i}$, the simulator is given $y_i = M_i(s_{i-1})$ and $t_i = \mathsf{Time}(M_i, s_{i-1})$, where $s_i = \mathsf{NextMem}(M_i, s_{i-1})$. The simulator then outputs $\mathsf{GbPrg}'(SK', D_i, \eta(S,\lambda)\cdot T_i, i)$, where D_i is a "dummy program". As described in Algorithm 19, D_i independently samples addresses to access for t_i steps, and then outputs y_i.

Data: Underlying running time t_i, output value y_i, PPRF G_i, circuits $C_1, \ldots, C_m$ guaranteed by localized randomness

```
for t = 1, ..., t_i do
    for k = 1, ..., m do
        r_k ← G_i(t, k);
        Access addresses given by C_k(r_k)
return y_i.
```

Algorithm 19: Pseudocode for a dummy RAM machine which simulates pseudorandom addresses to access using the circuits $C_1, \ldots, C_m$ given in the definition of localized randomness, and then outputs y_i.

The rest of this section is devoted to proving that this simulator is indistinguishable from the real challenger.

Hybrid $H_{i,j}$ We show indistinguishability by giving a sequence of "hybrid" challengers $H_{i,j}$ for $i = 1, \ldots, \ell$ and $j = 1, \ldots, T$, and show that they are all indistinguishable. In hybrid $H_{i,j}$, the challenger:

- Answers the memory query s_0 with $\mathsf{GbMem}'(SK', \mathsf{OMem}(1^\lambda, s_0, S))$ as in the real game.

- For $k < i$, if the k^{th} query is $M_k, 1^{T_k}$, then it is answered with the output of $\mathsf{GbPrg}'(SK', \mathsf{OProg}(1^\lambda, M_k, S), \eta(S, \lambda) \cdot T_k, k)$, just as in the real game.

- The i^{th} query $M_i, 1^{T_i}$ is answered with $\mathsf{GbPrg}(SK', N_{i,j}, \eta(S, \lambda) \cdot T_i, i)$, where N_i is a RAM machine which acts like M_i for the first j underlying steps, and acts like D_i for the rest of the steps. $N_{i,j}$ is described more precisely in Algorithm 20.

- For $k > i$, the k^{th} query $M_k, 1^{T_k}$ is answered with $\mathsf{GbPrg}'(SK', D_k, \eta(S, \lambda) \cdot T_k, k)$ for a dummy program D_k, just as in the simulator.

Data: RAM machine M_i, Underlying running time t_i, output value y_i, PPRFs F and G_i

```
op := ReadWrite(0 ↦ 0);
for t = 1, ..., j do
    Execute OProg(op)^{F(i,t)}, yielding a result v;
    Run one step of M_i with memory input v, yielding a new value for op;
for t = j + 1, ..., t_i do
    for k = 1, ..., m do
        r_k := G_i(t, k);
        Access addresses given by C_k(r_k)
return y_i.
```

Algorithm 20: Pseudocode for a RAM machine $N_{i,j}$ which starts acting like a dummy machine after j steps.

$H_{\ell+1,0}$ is identical to the real world, so it remains to show the following three claims:

Claim 5.20.1. $H_{i,T} \approx H_{i+1,0}$.

Proof. This follows directly from fixed-address security, because the semi-dummy machine $N_{i,0}$ accesses the same addresses and has the same output as the dummy machine D_i. □

Claim 5.20.2. $H_{1,0}$ *is indistinguishable from the simulator.*

Proof. This follows from fixed-address security, and from the fact that the set of non-empty addresses in $\mathsf{OMem}(1^\lambda, s, S)$ is simulatable given $\|s\|_0$. □

Our main claim is the following:

Claim 5.20.3. $H_{i,j} \approx H_{i,j+1}$.

Proof. Recall the definition of an ORAM with strong localized randomness. The addresses accessed in the oblivious exeution of $\mathsf{op}_{i,j}$ consist of m different chunks $\vec{a}_1, \ldots, \vec{a}_m$. Each $\vec{a}_k$ depends on some contiguous substring of the random tape R, indexed by an interval I_k, via a circuit C_k. The interval I_k depends on the underlying operations being executed.

We present $m+1$ hybrids $H_{i,j,m}$ through $H_{i,j,0}$. In hybrid $H_{i,j,k}$, the addresses $\vec{a}_1, \ldots, \vec{a}_k$ are generated honestly, and addresses $\vec{a}_{k+1}, \ldots, \vec{a}_m$ are simulated as

$$C_{k+1}(r_{k+1}), \ldots, C_m(r_m)$$

for pseudorandomly chosen $r_{k+1}, \ldots, r_m$.

We prove that no adversary $\mathcal{A}$ can distinguish between $H_{i,j,k}$ and $H_{i,j,k-1}$ if $\mathcal{A}$ first commits to $2\log(T \cdot \ell_R)$ bits about what it is going to do. Specifically, we suppose that $\mathcal{A}$ initially sends I_k (which depends on the machines $M_1, \ldots, M_i$). In this case, we can show the indistinguishability of $H_{i,j,k}$ and $H_{i,j,k-1}$ by making a sequence of indistinguishable changes.

1. The puncturable PRF F sampled during Setup is punctured at I_k, and has the values $F(I_k)$ hard-coded in all machines. This is indistinguishable because of fixed-address security of the underlying garbling scheme.

2. The machine M_i' has $\vec{a}_k$, the addresses accessed in the k^{th} chunk of M_i's j^{th} operation, hard-coded. This also is indistinguishable because of fixed-address security.

3. The hard-coded values $F(I_k)$ are replaced by truly random values r_k, and $\vec{a}_k$ is replaced by $C_k(r_k)$. This is indistinguishable by the security of the punctured PRF F.

4. r_k is replaced by $F(I_k)$ and F is unpunctured. By localized randomness properties – namely, no other $\vec{A}_i$ depends on R_{I_k} and $I_1, \ldots, I_m$ are pairwise disjoint – this doesn't affect the addresses accessed by $M_1', \ldots, M_i'$. So this is indistinguishable by fixed-address security.

5. $C_k(r_k)$ is replaced by $C_k(G_i(j,k))$. This is indistinguishable by the puncturable PRF security of G_i.

It suffices to analyze this semi-selective game because (by a usual complexity leveraging technique) if no adversary has advantage ϵ in this game, then no adversary has advantage $\epsilon' = \epsilon/(T\ell_R)^2$ in distinguishing $H_{i,j,k}$ from $H_{i,j,k-1}$. Since T, ℓ_R, and S are polynomial in the security parameter, if ϵ is negligible then ϵ' is as well. □

□

Appendix A

The RAM Model

When studying fine-grained algorithmic complexity, the choice of computational model is important. Most models of computation can simulate each other with polynomial blow-up in running-time and constant factor blow-up in space usage, but we want to simulate real-world computations with nearly constant overhead. We therefore focus on the word RAM model of computation, which is standard in practice.

In this chapter we formally define the RAM machine model that we consider. We give two equivalent definitions, each of which is convenient in different settings. Our definitions allow for flexibility in the concrete set of (word) operations that the RAM supports.

A.1 The Assembly Program Representation

Definition A.1 (RAM Machine). *A* RAM machine M *relative to a finite operation set*[1] $\mathcal{O}$ *of functions* $\{\star_i : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}\}_i$ *is a finite list of* ℓ *instructions* $(\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$*, each taking one of the following forms (for some* $i, j, k \in \mathbb{N}$*).*

- *Input:* $R[i] \leftarrow$ `INPUT`$[R[j]]$
- *Load/Store instructions: One of*
  - $R[i] \leftarrow c$ *for some constant* $c \in \mathbb{Z}_{\geq 0}$
  - $R[i] \leftarrow R[j]$
  - $R[i] \leftarrow R[R[j]]$
  - $R[R[i]] \leftarrow R[j]$
- *Word operations:* $R[i] \leftarrow R[j] \star R[k]$ *for some operation* $\star \in \mathcal{O}$.
- *Control flow instructions, either*
  - `GOTO` i, *or*

[1] The set of allowed operations is a common source of variation between different RAM models, but for us it only matters that the set is finite and each operation is computable in polynomial time.

– *GOTO* i *IF* $R[j] \neq 0$.

- *Halting instructions:* *ACCEPT* *or* *REJECT*.

A.1.1 Execution Semantics

Loosely speaking, the execution of a RAM machine M on a given input string x proceeds as follows. We start with a memory tape which is set to an infinite sequence of 0's. We use $R[i]$ to refer to the i^{th} memory cell. At each time step, we execute the relevant RAM instruction. For example, $R[i] \leftarrow \texttt{INPUT}[R[j]]$ means that i^{th} memory cell $R[i]$ obtains the value of the $(R[j])$-th bit of x. The other operations are similarly defined in the natural way. Throughout the execution we keep track of a *program counter* $\mathsf{pc} \in [\ell]$ that points to the current instruction being executed by the RAM program. The program counter advances linearly, except in the case of `GOTO` instructions (or the halting instructions `ACCEPT` and `REJECT`). We say that $M(x) = 1$ (resp., $M(x) = 0$) if the RAM machine halts at the `ACCEPT` (resp, `REJECT`) instruction, given input x. We proceed to the formalization of the foregoing discussion.

Definition A.2. *A* configuration *of a RAM machine* $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ *is a tuple* $(\mathsf{pc}, \mathcal{T})$*, where*

- $\mathsf{pc} \in [\ell]$ *is a "program counter" indicating the next instruction to be executed, and*
- $\mathcal{T} : \mathbb{N} \to \mathbb{Z}_{\geq 0}$ *is the memory contents of* M*.*

If $\mathsf{ins}_{\mathsf{pc}} =$ *ACCEPT, the configuration is said to be an* accepting *configuration. If* $\mathsf{ins}_{\mathsf{pc}} =$ *REJECT, the configuration is said to be a* rejecting *configuration. In either case, the configuration is said to be a* halting *configuration.*

Definition A.3. *The* initial *configuration of a RAM machine* $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ *is the configuration* $(1, \mathcal{T}_0)$*, where* $\mathcal{T}_0(\alpha) = 0$ *for all* $\alpha \in \mathbb{N}$*.*

The following definition formalizes the "evolution" of a RAM machine computation.

Definition A.4. *A non-halting configuration* $(\mathsf{pc}, \mathcal{T})$ *for a RAM machine* $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ *is said to* yield *the configuration* $(\mathsf{pc}', \mathcal{T}')$ *on input* $x = x_1 \cdots x_n$*, denoted* $(\mathsf{pc}, \mathcal{T}) \overset{M,x}{\to} (\mathsf{pc}', \mathcal{T}')$*, if the following holds:*

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[i] \leftarrow$ *INPUT*$[R[j]]$*, then*

$$\mathcal{T}'(\alpha) = \begin{cases} x_{\mathcal{T}(j)} & \text{if } \alpha = i \text{ and } 1 \leq \mathcal{T}(j) \leq n \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

and $\mathsf{pc}' = \mathsf{pc} + 1$.[2]

[2]The behavior of this instruction on $R[j] > n$ can be used to determine the value of n via a binary search.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[i] \leftarrow c$, *then*

$$\mathcal{T}'(\alpha) = \begin{cases} c & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[i] \leftarrow R[j]$, *then*

$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[i] \leftarrow R[R[j]]$, *then*

$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(\mathcal{T}(j)) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[R[i]] \leftarrow R[j]$, *then*

$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) & \text{if } \alpha = \mathcal{T}(i) \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* $R[i] \leftarrow R[j] \star R[k]$ *for some operation* $\star \in \mathcal{O}$, *then*

$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) \star \mathcal{T}(k) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* `GOTO` i, *then* $\mathcal{T}' = \mathcal{T}$, *and* $\mathsf{pc}' = i$.

- *If* $\mathsf{ins}_{\mathsf{pc}}$ *is of the form* `GOTO` i `IF` $R[j] \neq 0$, *then* $\mathcal{T}' = \mathcal{T}$, *and*

$$\mathsf{pc}' = \begin{cases} i & \text{if } \mathcal{T}(j) \neq 0 \\ \mathsf{pc} + 1 & \text{otherwise.} \end{cases}$$

When M and x are clear from the context we omit them from the notation and simply write $\mathcal{C} \to \mathcal{C}'$.

Definition A.5. *If $\mathcal{C}_0$ is the initial configuration of M, and if there is a finite sequence of configurations $\mathcal{C}_0 \overset{M,x}{\to} \cdots \overset{M,x}{\to} \mathcal{C}_T$ with $\mathcal{C}_T$ a halting configuration, then $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ is said to be the* transcript *of M on x, and we say that M* halts *at $\mathcal{C}_T$ on x. We denote this symbolically by writing $\mathcal{C}_0 \overset{M,x}{\Rightarrow} \mathcal{C}_T$.*

Definition A.6. *The evaluation of M on x is defined as*

$$M(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } M \text{ halts at } \mathcal{C} \text{ on } x \text{ for some accepting configuration } \mathcal{C} \\ 0 & \text{if } M \text{ halts at } \mathcal{C} \text{ on } x \text{ for some rejecting configuration } \mathcal{C} \\ \bot & \text{otherwise.} \end{cases}$$

We next define a *combined access pattern* as the list of all memory and input locations that are accessed by the RAM program on a given input. Formally, this is defined as follows.

Definition A.7. *If the transcript for a RAM machine $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ on an input x is $((\mathsf{pc}_0, \mathcal{T}_0), \ldots, (\mathsf{pc}_T, \mathcal{T}_T))$, then the* combined access pattern *(access pattern for short) of M on x is the concatenated tuple $\mathbf{a}_1 \| \cdots \| \mathbf{a}_T$, where for each $t \in [T]$,*

- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[i] \leftarrow$ `INPUT`$[R[j]]$, then $\mathbf{a}_t$ is the tuple $(j, \mathcal{T}_t(j), i)$.*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[i] \leftarrow c$, then $\mathbf{a}_t$ is the singleton tuple (i).*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[i] \leftarrow R[j]$, then $\mathbf{a}_t = (j, i)$.*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[i] \leftarrow R[R[j]]$, then $\mathbf{a}_t = (j, \mathcal{T}_t(j), i)$.*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[R[i]] \leftarrow R[j]$, then $\mathbf{a}_t = (j, i, \mathcal{T}(i))$.*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form $R[i] \leftarrow R[j] \star R[k]$, then $\mathbf{a}_t = (j, k, i)$.*
- *If $\mathsf{ins}_{\mathsf{pc}_t}$ is of the form* `GOTO` *i* `IF` *$R[j] \neq 0$, then $\mathbf{a}_t$ is the singleton tuple (j).*
- *Otherwise, $\mathbf{a}_t$ is the empty tuple.*

A.1.2 Time and Space Complexity

We next define the time and space complexity measures for RAM machines.

Definition A.8. *When a transcript $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ exists[3] for a RAM machine M on input x, then the* running time *of M on x is defined as* $\mathsf{Time}(M, x) \overset{\text{def}}{=} T$.

Modeling the space usage of a RAM program is a bit trickier. In particular, RAM programs can, in time $O(t)$ access the 2^t-th memory cell (assuming their instruction set includes addition). Given that, it may first seem natural to define the space usage as the number of memory cells that were ever written to by the time the program halts. That definition is problematic since the program can implicitly use the maximal memory address used as free space.

[3]Recall that a machine does not necessarily halt for every input.

Definition A.9. *The* space usage *of a tape $\mathcal{T}$ of a RAM machine is defined as*

$$\mathsf{Space}(\mathcal{T}) \stackrel{\mathsf{def}}{=} \max\{i : \mathcal{T}(i) \neq 0\}.$$

The space usage *or a configuration $\mathcal{C} = (\mathsf{pc}, \mathcal{T})$ is defined as $\mathsf{Space}(\mathcal{C}) = \mathsf{Space}(\mathcal{T})$.*

When a transcript $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ exists for a RAM machine M on input x, the space usage *of M on x is defined as*

$$\mathsf{Space}(M, x) \stackrel{\mathsf{def}}{=} \max_{i \in [T]} \mathsf{Space}(\mathcal{C}_i).$$

So far we have not bounded the amount of information that can be stored in each one of the memory cells. The standard RAM model allows $O(\log n)$ bits of information per cell, where n is the input length. Likewise, we have not placed any restrictions on the complexity of the RAM operations. We define a word RAM as a RAM that uses bounded size words as its memory and whose supported binary operations are computable in polynomial time. More formally:

Definition A.10 (Word RAM). *Let $w = w(n) \in \mathbb{N}$. A RAM machine M with operations $\mathcal{O}$ is a $w(\cdot)$-*bit word RAM *if:*

- *Each $\star \in \mathcal{O}$ is computable (by, say, a Turing machine) in polynomial time (in the bit length of its input).*
- *For each $x \in \{0,1\}^n$ with corresponding transcript $((\mathsf{pc}_0, \mathcal{T}_0), \ldots, (\mathsf{pc}_T, \mathcal{T}_T))$, it holds that $\mathcal{T}_t(i) < 2^{w(n)}$, for all $t \in \{0, \ldots, T\}$ and all $i \in \mathbb{N}$.*

For natural RAM programs the word size typically either is $\Theta(\log(S))$ or can be reduced to $\Theta(\log(S))$ without asymptotic loss in running time.

A.1.3 RAM Machines

In this work, a RAM machine M is defined as a tuple (Σ, Q, Y, C), where:

- Σ is a finite set, which is the possible contents of a memory cell. For example, $\Sigma = \{0,1\}$.
- Q is the set of all possible "local states" of M, containing some initial state q_0. (We think of Q as a set that grows polynomially as a function of the security parameter. That is, a state $q \in Q$ can encode cryptographic keys, as well as "local memory" of size that is bounded by some fixed polynomial in the security parameter.)
- Y is the output space of M.
- C is a circuit implementing a transition function which maps $Q \times (\Sigma \cup \{\epsilon\}) \to (Q \times O_\Sigma) \cup Y$. Here O_Σ denotes the set of memory operations with Σ as the alphabet of possible memory symbols. Precisely, $O_\Sigma = (\mathbb{N} \times \Sigma)$. That is, C takes the current state and the value returned by the memory access function,

and returns a new state, a memory address, a read/write instruction, and a value to be written in case of a write.

We write $|M|$ to denote the tuple $(\ell_\Sigma, \ell_Q, \ell_Y, |C|)$, where ℓ_Σ is the length of a binary encoding of Σ, and similarly for ℓ_Q and ℓ_Y.

A.1.4 Memory Configurations

A memory configuration on alphabet Σ is a function $s : \mathbb{N} \to \Sigma \cup \{\epsilon\}$. Let $\|s\|_0$ denote $\big|\{a : s(a) \neq \epsilon\}\big|$ and let $\|s\|_\infty$ denote $\max\big(\{a : s(a) \neq \epsilon\}\big)$, which we will call the *length* of the memory configuration. A memory configuration s can be implemented (say with a balanced binary tree) by a data structure of size $O\big(\|s\|_0\big)$, supporting updates to any index in $O\big(\log \|s\|_\infty\big)$ time.

We can naturally identify a string $x = x_1 \dots x_n \in \Sigma^*$ with the memory configuration s_x, defined by

$$s_x(i) = \begin{cases} x_i & \text{if } i \leq n \\ \epsilon & \text{otherwise} \end{cases}$$

Looking ahead, efficient representations of sparse memory configurations (in which $\|s\|_0 < \|s\|_\infty$) are convenient for succinctly garbling computations in which the space usage is larger than the input length.

A.1.5 Execution

We now define what it means to execute a RAM machine $M = (\Sigma, Q, Y, C)$ on an initial memory configuration $s_0 \in \Sigma^{\mathbb{N}}$ to obtain $M(s_0)$.

Define $a_0 = 0$. For $i > 0$, iteratively define $(q_i, a_i, v_i) = C(q_{i-1}, s_{i-1}(a_{i-1}))$ and define the i^{th} memory configuration s_i as

$$s_i(a) = \begin{cases} v_i & \text{if } a = a_i \\ s_{i-1}(a) & \text{otherwise} \end{cases}$$

If $C(q_{t-1}, s_{t-1}(a_{t-1})) = y \in Y$ for some t, then we say that $M(s_0) = y$. If there is no such t, we say that $M(s_0) = \perp$. When $M(s_0) \neq \perp$, it is convenient to define the following functions:

- Define the *running time* of M on s_0 as the above t, and denote it $\mathsf{Time}(M, s_0)$.
- Define the *space usage* of M on s_0 as $\max_{i=0}^{t-1}(\|s_i\|_\infty)$, and denote it $\mathsf{Space}(M, s_0)$.
- Define the *execution transcript* of M on s_0 as $((q_0, a_0, v_0), \dots, (q_{t-1}, a_{t-1}, v_{t-1}), y)$, and denote it $\mathcal{T}(M, s_0)$.
- Define the *resulting memory configuration* of M on s_0 as s_t, and denote it $\mathsf{NextMem}(M, s_0)$.

A.2 The CPU Circuit Representation

We often work with RAM programs that are restricted to inputs of bounded length. We call these bounded RAM machines. These are often conveniently represented via a small "CPU circuit".

Definition A.11. *A* CPU circuit *is a boolean circuit*

$$C : Q \times W \to Q \times \mathsf{Ops}_{A,W},$$

where:

- $Q = \{0,1\}^{\ell_Q}$ *for some explicitly given* ℓ_A. Q *represents the set of control states (along with register contents, etc.) for the RAM machine, and is called the* state set *of* C.
- $W = \{0,1\}^{\ell_W}$ *for some explicitly given* ℓ_W. W *represents the set of values (words) that can be stored in any cell of memory, and is called the* word set *of* C.
- $A = \{0,1\}^{\ell_A}$ *for some explicitly given* ℓ_A. A *represents the set of addresses that indexes the RAM machine's memory cells, and is called the* address set *of* C. *The* worst-case space usage of C *is defined to be* 2^{ℓ_A}.
- $\mathsf{Ops}_{A,W} = A \times W$, *and represents the set of atomic memory operations that can be performed by a RAM machine with address set* A *and word set* W *(see discussion below).*

A RAM machine M*'s* CPU circuit representation *is a tuple* $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, *where* q_0, q_{acc}, *and* q_{rej} *are members of the state set of* C. q_0 *is called the* initial state *of* M, q_{acc} *is called the* accept state *of* M, *and* q_{rej} *is called the* reject state *of* M.

In the above definition, the set $\mathsf{Ops}_{A,W}$ represents allowed "memory operations". For simplicity, we allowed only a single type of memory operation, namely a "read and write" operation. This operation is parameterized by an address $a \in A$ and a word $w \in W$ and is represented by the tuple (a, w). The action of this operation is to (1) write w to the a^{th} memory cell, and (2) return the value that was previously there. Either a read operation or a write operation can be simulated by two of these operations.

Definition A.12. *Let* $M = (C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ *be the CPU circuit representation of a bounded RAM machine, where*

$$C : Q \times W \to Q \times \mathsf{Ops}_{A,W},$$

A computation transcript *of* M *on* $x \in W^*$ *(which may or may not exist, but is unique if it does exist) is a sequence*

$$(q_0, a_0, w_0), \ldots, (q_T, a_T, w_T)$$

such that

- $a_0 = 0$ *and* $w_0 = 0$.[4]
- $q_T \in \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$, *and for all* $0 \leq i < T$, $q_i \notin \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$.
- *For* $1 \leq i < T$,

$$(q_i, (a_i, w_i)) = C(q_{i-1}, w_{\mathsf{read}}),$$

where $w_{\mathsf{read}} \in W$ *is defined as follows. If there exists some* $j \in [0, i-1)$ *for which* $a_j = a_{i-1}$, *then let* j^* *be the largest such* j, *and define* $w_{\mathsf{read}} \stackrel{\mathsf{def}}{=} w_{j^*}$. *Otherwise, define* $w_{\mathsf{read}} \stackrel{\mathsf{def}}{=} x_{a_{i-1}}$ *(or* 0 *if* a_{i-1} *exceeds the length of* x*).*

When a transcript exists, T *is said to be the* running time *of* M *on* x. *The* memory transcript *of* M *on* x *is the sequence* $((a_0, w_0), \ldots, (a_T, w_T))$. *The* address transcript *of* M *on* x *is the sequence* $(a_0, \ldots, a_T)$.

When a transcript for M on x exists, we say that

$$M(x) = \begin{cases} 1 & \text{if } q_T = q_{\mathsf{acc}} \\ 0 & \text{if } q_T = q_{\mathsf{rej}}. \end{cases}$$

When no transcript exists, we say that $M(x) = \perp$, and that the running time of M on x is infinite.

A.2.1 Probablistic RAMs

One natural extension of the deterministic RAM machine is the ability to make random choices throughout a computation. The precise power of randomness is a topic of immense interest in complexity, and is well beyond the scope of this thesis. Here, we simply give syntax for describing probabilistic RAM machines.

Definition A.13. *A* probabilistic CPU circuit *is a boolean circuit*

$$C : Q \times W \times R \to Q \times \mathsf{Ops}_{A,W},$$

where Q, W, A, *and* $\mathsf{Ops}_{A,W}$ *are as in Definition A.11, and*

- $R \stackrel{\mathsf{def}}{=} \{0,1\}^{\ell_R}$ *for some explicitly given* ℓ_R. R *represents a set of possible random outcomes in a single step of computation, and is called the* randomness alphabet *of* C.

The CPU circuit representation *of a probabilistic RAM machine is a tuple* $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, *where* C *is a probabilistic CPU circuit, and* q_0, q_{acc}, *and* q_{rej} *are members of the state set of* C. q_0 *is called the* initial state *of* M, q_{acc} *is called the* accept state *of* M, *and* q_{rej} *is called the* reject state *of* M.

[4]This choice is somewhat arbitrary; any initial memory operation could be used instead.

Let $M = (C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ be a probabilistic RAM machine. The transcript of M is now a function of a "random tape" $r = (r_1, r_2, \ldots)$ sampled uniformly at random from $R^{\mathbb{N}}$. Once the random tape is fixed, the transcript is defined analogously to Definition A.12.

Definition A.14. *Let $M = (C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ be the CPU circuit representation of a bounded probabilistic RAM machine, where*

$$C : Q \times W \times R \to Q \times \mathsf{Ops}_{A,W},$$

A computation transcript *of M on $x \in W^*$ with randomness $r \in R^{\mathbb{N}}$ is a sequence*

$$(q_0, a_0, w_0), \ldots, (q_T, a_T, w_T)$$

such that

- $a_0 = 0$ *and* $w_0 = 0$.
- $q_T \in \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$, *and for all* $0 \leq i < T$, $q_i \notin \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$.
- *For* $1 \leq i < T$,
$$(q_i, (a_i, w_i)) = C(q_{i-1}, w_{\mathsf{read}}, r_i),$$
where $w_{\mathsf{read}} \in W$ is defined as follows. If there exists some $j \in [0, i-1)$ for which $a_j = a_{i-1}$, then let j^* be the largest such j, and define $w_{\mathsf{read}} \stackrel{\mathsf{def}}{=} w_{j^*}$. Otherwise, define $w_{\mathsf{read}} \stackrel{\mathsf{def}}{=} x_{a_{i-1}}$ (or 0 if a_{i-1} exceeds the length of x).

A computation transcript may or may not exist, but it is unique when it does exist. When a transcript exists, T is said to be the running time *of M on x. The* memory transcript *of M on x with randomness r is the sequence $((a_0, w_0), \ldots, (a_T, w_T))$. The* address transcript *of M on x with randomness r is the sequence $(a_0, \ldots, a_T)$.*

When a transcript for M on x with randomness r exists, we say that

$$M(x; r) = \begin{cases} 1 & \text{if } q_T = q_{\mathsf{acc}} \\ 0 & \text{if } q_T = q_{\mathsf{rej}}. \end{cases}$$

When no transcript exists, we say that $M(x; r) = \perp$, and that the running time of $M(x; r)$ is infinite.

Definition A.15. *If*

$$C : Q \times W \times R \to Q \times \mathsf{Ops}_{A,W},$$

is a probabilistic CPU circuit with $R = \{0,1\}^{\ell_R}$, if $T = 2^t$ is some implicit time bound, and if $f : \{0,1\}^t \to \{0,1\}^{\ell_R}$ is any function, then we write C^f to denote a (deterministic) CPU circuit

$$C^f : Q' \times W \to Q' \times \mathsf{Ops}_{A,W},$$

where:

- $Q' = \Big(\{0,1\}^t \times (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\})\Big) \sqcup \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$
- $$C^f((t,q),w) \stackrel{\mathsf{def}}{=} \begin{cases} ((t+1,q'),\mathsf{op}) & \text{if } q' \notin \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\} \\ (q',\mathsf{op}) & \text{otherwise,} \end{cases}$$

  where $(q',\mathsf{op}) := C(q,w,f(t))$.

Definition A.16. *If M is a bounded probabilistic RAM machine with CPU circuit representation $(C, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ for a probabilistic CPU circuit*

$$C : Q \times W \times R \to Q \times \mathsf{Ops}_{A,W},$$

if $T = 2^t$ is an associated bound on the running time of M, and if $f : \{0,1\}^t \to \{0,1\}^{\ell_R}$ is any function, then we write M^f to denote the bounded (deterministic) RAM machine with CPU circuit representation $(C^f, (0,q_0), q_{\mathsf{acc}}, q_{\mathsf{rej}})$.

Appendix B

The Base No-Signaling PCP

Our PCP is the same as in [KRR14], which in turn is based on the PCP of [BFLS91].

As in the remark following Definition 4.13, we can assume we are given N which bounds the number of variables of φ. First pad φ so that it has exactly N variables. Let $x = (x_1, \ldots, x_N) \in \{0,1\}^N$ denote a satisfying assignment to the variables of φ. Let

$$X : \mathbb{F}^m \to \mathbb{F}$$

be the low-degree extension of x (as defined in Section 4.2.1), where $\mathbb{F}$ is defined so that

$$O(\log^2 N) \leq |\mathbb{F}| \leq \text{polylog}(N).$$

Namely, let $H = \{0, 1, \ldots, \log N - 1\}$ and let $m = \frac{\log N}{\log \log N}$, so that $N = |H|^m$. (For simplicity and without loss of generality we assume that $\log N$ and $\frac{\log N}{\log \log N}$ are integers). Since $N = |H|^m$, we can identify $[N]$ and H^m by the lexicographic order on H^m. Thus, we can view $x_1, \ldots, x_N$ as indexed by $i \in H^m$ (rather than $i \in [N]$). We can hence view $x = (x_1, \ldots, x_N)$ as a function $x : H^m \to \{0,1\}$ (given by $x(i) = x_i$, where we identify $[N]$ and H^m). The low-degree extension of x is the (unique) multivariate polynomial

$$X : \mathbb{F}^m \to \mathbb{F}$$

of degree $|H| - 1$ in each variable, such that $X|_{H^m} \equiv x$

Let $\phi : (H^m)^3 \times \{0,1\}^3 \to \{0,1\}$ be the clause indicator function of φ. That is, $\phi(i_1, i_2, i_3, b_1, b_2, b_3) = 1$ if and only if the clause $(w_{i_1} = b_1) \vee (w_{i_2} = b_2) \vee (w_{i_3} = b_3)$ appears in φ.

In what follows, we think of ϕ as a function $\phi : (H^m)^3 \times H^3 \to \{0,1\}$ where for every $(b_1, b_2, b_3) \in H^3 \setminus \{0,1\}^3$ and for every $(i_1, i_2, i_3) \in (H^m)^3$,

$$\phi(i_1, i_2, i_3, b_1, b_2, b_3) = 0.$$

We denote by $\ell = 3m + 3$, and thus $\phi : H^\ell \to \{0,1\}$.

Let

$$\hat{\phi} : \mathbb{F}^\ell \to \mathbb{F}$$

be the low-degree extension of ϕ (of degree at most $|H| - 1$ in each variable).

The fact that $X : \mathbb{F}^m \to \mathbb{F}$ encodes a satisfying assignment for φ implies that for every $z = (i_1, i_2, i_3, b_1, b_2, b_3) \in (H^m)^3 \times H^3 = H^\ell$,

$$\hat{\phi}(z) \cdot (X(i_1) - b_1) \cdot (X(i_2) - b_2) \cdot (X(i_3) - b_3) = 0 \tag{B.1}$$

Let $P_0 : \mathbb{F}^\ell \to \mathbb{F}$ be the ℓ-variate polynomial defined as follows:
For $z = (i_1, i_2, i_3, b_1, b_2, b_3) \in (\mathbb{F}^m)^3 \times \mathbb{F}^3 = \mathbb{F}^\ell$,

$$P_0(z) \triangleq \hat{\phi}(z) \cdot (X(i_1) - b_1) \cdot (X(i_2) - b_2) \cdot (X(i_3) - b_3)$$

Equation (B.1) implies that $P_0|_{H^\ell} \equiv 0$ (assuming that indeed X encodes a satisfying assignment). Moreover, the fact that X has degree $< |H|$ in each variable, and $\hat{\phi}$ has degree $< |H|$ in each variable, implies that P_0 has degree $< 2|H|$ in each variable, and hence total degree $< 2|H|\ell$.

Next we define $P_1 : \mathbb{F}^\ell \to \mathbb{F}$. For every $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$, let

$$P_1(z) = \sum_{h \in H} P_0(h, z_2, \ldots, z_\ell) {z_1}^h$$

Note that $P_1|_{\mathbb{F} \times H^{\ell-1}} \equiv 0$. More generally, we define by induction $P_1, \ldots, P_\ell : \mathbb{F}^\ell \to \mathbb{F}$ where for every $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$,

$$P_i(z) = \sum_{h \in H} P_{i-1}(z_1, \ldots, z_{i-1}, h, z_{i+1}, \ldots, z_\ell) {z_i}^h$$

Note that $P_1, \ldots, P_{\ell-1}$ have degree $< 2|H|$ in each variable, and hence total degree $< 2|H|\ell$. Note also that $P_i|_{\mathbb{F}^i \times H^{\ell-i}} \equiv 0$, and in particular $P_\ell \equiv 0$.

The PCP proof for the fact that φ is satisfiable consists of the polynomial $X : \mathbb{F}^m \to \mathbb{F}$ (which is the low-degree extension of a satisfiable assignment), and the ℓ polynomials $P_i : \mathbb{F}^\ell \to \mathbb{F}$, for $i = 0, \ldots, \ell - 1$. To these polynomials we add the polynomial $P_\ell \equiv 0$. The polynomial P_ℓ can be removed from the PCP proof (as it is the 0 polynomial) and is added just for simplicity of the notation. When the verifier queries $P_\ell(z)$ she gets 0 automatically.

Let $D_X = \mathbb{F}^m$ be the domain of X, and let $D_0, \ldots, D_\ell$ be $\ell + 1$ copies of $\mathbb{F}^\ell$, the domain of $P_0, \ldots, P_\ell$. We view $D_X, D_0, \ldots, D_\ell$ as the domains of $X, P_0, \ldots, P_\ell$, respectively. Denote

$$D \stackrel{\text{def}}{=} D_X \sqcup D_0 \sqcup \ldots \sqcup D_\ell$$

The set D is the alphabet of queries in the PCP. We will refer to D as the domain of the PCP.

Remark B.1. The entire PCP proof can be generated in time poly(N).

B.1 The PCP Verifier, $V = (V_0, V_1)$

As mentioned above, we assume that the verifier V is not given φ explicitly, since we require the runtime of V to be significantly smaller than $|\varphi|$. Rather, we assume that

the verifier is given oracle access to $\hat{\phi}$ which is the low-degree extension of ϕ, where ϕ is the clause indicator function of φ and is viewed as a function $\phi : H^{3m+3} \to \{0,1\}$.[1] We also assume that the verifier is given N, the number of variables of φ.

Thus, we assume that the verifier has oracle access to honestly generated $\hat{\phi}$. That is, we assume that V can get the correct value of $\hat{\phi}(z)$ for free, for as many points $z \in \mathbb{F}^\ell$ as she wants.

Notation. We denote by λ the security parameter, we denote by a_i the i^{th} coordinate of a vector a. In particular, for a line $L : \mathbb{F} \to \mathbb{F}^\ell$, a field element $t \in \mathbb{F}$ and a coordinate $i \in \{1, \ldots, \ell\}$, we denote by $L(t)_i$ the i^{th} coordinate of the point $L(t) \in \mathbb{F}^\ell$. We say that a line $L : \mathbb{F} \to \mathbb{F}^\ell$ is orthogonal to the i^{th} coordinate if for every $t_1, t_2 \in \mathbb{F}$, we have $L(t_1)_i = L(t_2)_i$.

The verifier V makes the following tests on the PCP proof (the exact tests are described below):

- **Low Degree Test for X.**
- **Low Degree Tests for P_i.**
- **Sum Check for P_i.**
- **Consistency Test of X and P_0.**

We note that we will have four types of low degree tests for each P_i, rather than just one. This is only for the simplicity of the analysis. It would be sufficient to do only one test, similar to the low degree test for X (but repeated on $O(|\mathbb{F}|^2)$ random lines, rather than a single random line), since all four types of tests that we actually do (and are formally described below) can be embedded in such a test.

Formally, the verifier V makes the following tests, and accepts if and only if the PCP proof passes all of them:

1. **Low Degree Test for X:** Choose a random line $L : \mathbb{F} \to \mathbb{F}^m$, and query X on all the points $\{L(t)\}_{t \in \mathbb{F}}$. Check that the univariate polynomial $X \circ L : \mathbb{F} \to \mathbb{F}$ is of degree $< m|H|$.

2. **Low Degree Test for P_i: Type 1 (fixed $L(0)_{i+1}$):** For every $i \in \{0, \ldots, \ell-1\}$ and every $u \in \mathbb{F}$, choose a random line $L : \mathbb{F} \to \mathbb{F}^\ell$, such that $L(0)_{i+1} = u$. Query P_i on all the points $\{L(t)\}_{t \in \mathbb{F}}$, and check that the univariate polynomial $P_i \circ L : \mathbb{F} \to \mathbb{F}$ is of degree $< 2|H|\ell$.

3. **Low Degree Test for P_i: Type 2 (orthogonal to the $(i+1)^{th}$ coordinate):** For every $i \in \{0, \ldots, \ell-1\}$, choose a random line $L : \mathbb{F} \to \mathbb{F}^\ell$ that is orthogonal to the $(i+1)^{th}$ coordinate.

   Query P_i on all the points $\{L(t)\}_{t \in \mathbb{F}}$, and check that the univariate polynomial $P_i \circ L : \mathbb{F} \to \mathbb{F}$ is of degree $< 2|H|\ell$.

[1] Jumping ahead, we note that when we use this PCP for delegation, we will use it with specific formulas φ for which the verifier can compute $\hat{\phi}$ on his own efficiently (in time polylog(N)).

4. **Low Degree Test for P_i: Type 3 (fixed $L(0)_{i+1}$; orthogonal to the i^{th} coordinate):** For every $i \in \{1, \ldots, \ell - 1\}$, and every $u \in \mathbb{F}$, choose a random line $L : \mathbb{F} \to \mathbb{F}^\ell$ that is orthogonal to the i^{th} coordinate, and satisfies $L(0)_{i+1} = u$. Query P_i on all the points $\{L(t)\}_{t \in \mathbb{F}}$, and check that the univariate polynomial $P_i \circ L : \mathbb{F} \to \mathbb{F}$ is of degree $< 2|H|\ell$.

5. **Low Degree Test for P_i: Type 4 (fixed $L(0)_i$; orthogonal to the $(i+1)^{th}$ coordinate):** For every $i \in \{1, \ldots, \ell - 1\}$, and every $u \in \mathbb{F}$, choose a random line $L : \mathbb{F} \to \mathbb{F}^\ell$ orthogonal to the $(i+1)^{th}$ coordinate, and satisfies $L(0)_i = u$. Query P_i on all the points $\{L(t)\}_{t \in \mathbb{F}}$, and check that the univariate polynomial $P_i \circ L : \mathbb{F} \to \mathbb{F}$ is of degree $< 2|H|\ell$.

6. **Sum Check for P_i:** For every $i \in \{1, \ldots, \ell\}$, choose a random point $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$. Query P_i, P_{i-1} on all the points $\{(z_1, \ldots, z_{i-1}, t, z_{i+1}, \ldots, z_\ell)\}_{t \in \mathbb{F}}$, and check that for every $t \in \mathbb{F}$,

$$P_i(z_1, \ldots, z_{i-1}, t, z_{i+1}, \ldots, z_\ell) = \sum_{h \in H} P_{i-1}(z_1, \ldots, z_{i-1}, h, z_{i+1}, \ldots, z_\ell) t^h$$

7. **Consistency of X and P_0:** Choose a random point $z = (i_1, i_2, i_3, b_1, b_2, b_3) \in (\mathbb{F}^m)^3 \times \mathbb{F}^3 = \mathbb{F}^\ell$. Query P_0 on the point z and X on the points i_1, i_2, i_3, and check that

$$P_0(z) = \hat{\phi}(z) \cdot (X(i_1) - b_1) \cdot (X(i_2) - b_2) \cdot (X(i_3) - b_3)$$

Complexity of the Verifier

Note that the total number of queries made by V to the PCP proof, as well as the total number of queries made by V to the function $\hat{\phi}$, are both at most $k = 6\ell|\mathbb{F}|^2$. The time complexity of V is polylog(N).

B.2 From Amplified Cheating Prover to Assignment Generator

Lemma B.2. *Let (V_0, V_1, P) be the PCP from Appendix B. There exists a probabilistic polynomial-time oracle machine* Assign *and a polynomial ℓ_0 such that for every negligible $\epsilon = \epsilon(\lambda)$, every polynomial ℓ, every security parameter $\lambda \in \mathbb{N}$, and every adaptive $k_{max} = \ell \cdot \ell_0$-computational no-signaling cheating prover* Prover**, if*

$$\Pr\left[\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda); \\ (\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(Q); \\ 1 \leftarrow \left(V_1^{\geq \lambda - r}\right)^{\hat{\phi}_1, \hat{\phi}_2}(\mathsf{st}, A_1, A_2)\end{array}\right] \geq 1 - \epsilon,$$

then Assign$^{\mathsf{Prover}^*}$ *is an adaptive (ℓ, ϵ')-partial assignment generator with*

$$\epsilon' = \epsilon \cdot \text{poly}(\lambda) + \text{negl}(\lambda) = \text{negl}(\lambda).$$

Moreover, the distribution (φ_0', φ_1')*, generated by sampling*

$$(\varphi_0', \varphi_1', A_0, A_1) \leftarrow \mathsf{Assign}_c^{\mathsf{Prover}^*}(1^\lambda, W)$$

is computationally indistinguishable from the distribution of (φ_0, φ_1) *generated as above.*

Fix $\ell_0 = \lambda \cdot |\mathbb{F}|^2 \leq \text{poly}(\lambda)$. The probabilistic polynomial-time oracle machine Assign is defined as follows: For any $\ell \leq \text{poly}(\lambda)$, any $\epsilon = \epsilon(\lambda) > 0$, and any adaptive k_{max}-computational no-signaling cheating prover Prover^*, where $k_{max} = \ell \cdot \ell_0$, such that

$$\Pr\left[\begin{array}{l}(Q, \mathsf{st}) \leftarrow V_0^{\otimes\lambda}(1^\lambda);\\ (\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(Q);\\ 1 \leftarrow \left(V_1^{\geq\lambda-r}\right)^{\hat{\phi}_1, \hat{\phi}_2}(\mathsf{st}, A_1, A_2)\end{array}\right] \geq 1 - \epsilon,$$

$\mathsf{Assign}^{\mathsf{Prover}^*}(\lambda, \mathsf{q}_1, \ldots, \mathsf{q}_\ell)$ does the following:

1. For each $i \in [\ell]$, choose λ random lines $L_{i,1}, \ldots, L_{i,\lambda} : \mathbb{F} \to D_X$ such that for every $j \in [\lambda]$ it holds that $L_{i,j}(0) = q_i$. (Note that $q_i \in D_X$.)

2. Run
$$(\varphi_1', \varphi_2', \{\mathsf{a}_{i,j}^{(1)}(t)\}, \{\mathsf{a}_{i,j}^{(2)}(t)\}) \leftarrow \mathsf{Prover}^*\left(\{L_{i,j}(t)\}_{i\in[\ell], j\in[\lambda], t\in\mathbb{F}}\right).$$

3. For every $i \in [\ell]$ and every $j \in [\lambda]$ do the following for every $v \in \mathbb{F}$:

   (a) Define $f^v : \mathbb{F} \to \mathbb{F}$ by setting $f^v(t) = \mathsf{a}_{i,j}^{(1)}(t)$ for every $t \neq 0$ and setting $f^v(0) = v$. Similarly, define $g^v : \mathbb{F} \to \mathbb{F}$ by setting $g^v(t) = \mathsf{a}_{i,j}^{(2)}(t)$ for every $t \neq 0$ and setting $g^v(0) = v$.

   (b) If f^v is a polynomial of degree $\leq m \cdot |H|$ then set $v_{i,j}^{(1)} \triangleq v$. If no such v exists then set $v_{i,j}^{(1)} = \perp$. Similarly, if g^v is a polynomial of degree $\leq m \cdot |H|$ then set $v_{i,j}^{(2)} \triangleq v$. If no such v exists then set $v_{i,j}^{(2)} = \perp$.

4. For every $i \in [\ell]$, let $v_i^{(1)} = \mathsf{maj}\{v_{i,1}^{(1)}, \ldots, v_{i,\lambda}^{(1)}\}$ and let $v_i^{(2)} = \mathsf{maj}\{v_{i,1}^{(2)}, \ldots, v_{i,\lambda}^{(2)}\}$.

5. Output $\{\varphi_1', \varphi_2', (v_1^{(1)}, \ldots, v_\ell^{(1)}), (v_1^{(2)}, \ldots, v_\ell^{(2)})\}$.

We need to prove that $\mathsf{Assign}^{\mathsf{Prover}^*}$ is an adaptive (ℓ, ϵ')-partial assignment generator with

$$\epsilon' = \epsilon \cdot \text{poly}(\lambda) + \text{negl}(\lambda).$$

Moreover, we need to prove that for any $k' \leq k_{\max}$, any $(\mathsf{q}_1, \ldots, \mathsf{q}_{k'}) \in D^{k'}$, and any $Q \subset D_X$ such that $|Q| \leq \ell$,

$$(\varphi_1, \varphi_2) \approx (\varphi_1', \varphi_2'),$$

where (φ_1, φ_2) denotes the instances that $\mathsf{Prover}^*(Q)$ outputs, and (φ_1', φ_2') denotes the instances that the assigner $\mathsf{Assign}^{\mathsf{Prover}^*}(Q)$ outputs. The latter follows immediately by the fact that Prover^* is computationally no-signaling, together with the fact that

$\mathsf{Assign}^{\mathsf{Prover}^*}(Q)$ always runs the prover Prover^* with some set of queries $Q' \subseteq D_X$ and outputs the instances (φ_1', φ_2') that $\mathsf{Prover}^*(Q')$ outputs. A similar argument shows that Assign is computational no-signaling.

It thus remains to prove that Assign satisfies the local soundness condition. The proof of the latter is quite involved, and is very similar to the proof from [KRR14].[2]

Local Soundness. Fix $\ell = \ell(\lambda)$ and $\epsilon = \epsilon(\lambda)$, and fix an adaptive k_{max}-wise CNS cheating prover Prover^*, where $k_{max} = \ell \cdot \ell_0$, such that

$$\Pr\left[1 \leftarrow \left(V_1^{\geq \lambda - r}\right)^{\hat{\phi}_1, \hat{\phi}_2}(\mathsf{st}, A_1, A_2) \right] \geq 1 - \epsilon.$$

in the probability space defined by sampling

$$\begin{array}{l} (Q, \mathsf{st}) \leftarrow V_0^{\otimes \lambda}(1^\lambda); \\ (\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(Q); \end{array}$$

We assume, without loss of generality that $|\varphi_1|, |\varphi_2| \geq \lambda$. This is without loss of generality, since otherwise a proof can consist of an entire satisfying assignment. Roughly speaking, the proof of local soundness can be partitioned into three parts.

We prove local soundness of φ_1. The proof of local soundness of φ_2 is identical. Denote the computationally no-signaling strategy of Prover^* by

$$\{\mathcal{A}_S\}_{S \subset D, |S| \leq k_{max}},$$

where $\mathcal{A}_S$ is a distribution that generates (φ, A) as follows: Run $(\varphi_1, \varphi_2, A_1, A_2) \leftarrow \mathsf{Prover}^*(S)$, and set $\varphi = \varphi_1$ and $A = A_1$.

Part 1. We start with the following definition that will be useful in the proof. Intuitively, a point z satisfies property $\mathcal{Z}(\epsilon', r')$ if when taking λ lines through it, and sending all these points to Prover^*, then with high probability, for most of these lines, the answers of Prover^* correspond to low degree polynomials that "evaluate" the point z to 0.

Definition B.3. Property $\mathcal{Z}(\epsilon', r')$:
Let $\epsilon' \geq 0$ and $r' \geq 0$. Let $i \in \{0, \ldots, \ell\}$. Let $z \in D_i$.

Let $L_1, \ldots, L_\lambda : \mathbb{F} \to D_i$ be λ random lines, such that for every $L \in \{L_1, \ldots, L_\lambda\}$, we have $L(0) = z$. Let $S = \{L_j(t)\}_{j \in [\lambda], t \in \mathbb{F}} \subset D_i$. Let $(\varphi, A) \in_R \mathcal{A}_S$.

Define $A^0 : S \to \mathbb{F}$ by $A^0(z') = A(z')$ for $z' \neq z$ and $A^0(z) = 0$.

We say that the point z satisfies property $\mathcal{Z}(\epsilon', r')$ (also denoted $z \in \mathcal{Z}(\epsilon', r')$) if with probability $\geq 1 - \epsilon'$, for at least $\lambda - r'$ of the lines $L \in \{L_1, \ldots, L_\lambda\}$, we have that $A^0 \circ L : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$ (where the probability is over $L_1, \ldots, L_\lambda, A$).

[2] Much of the text below is indeed taken from [KRR14].

Our main lemma about property $\mathcal{Z}(\epsilon', r')$ is that the property is satisfied, with small ϵ' and r', for *any* point $z = (z_1, \ldots, z_\ell) \in D_0$ such that $z_1, \ldots, z_\ell \in H$. (Intuitively, this is analogous to the formula $P_0|_{H^\ell} \equiv 0$, that is satisfied for $x \in \mathcal{L}$).

Lemma B.4. *There exists a negligible function $\mu = \mu(\lambda)$ such that for any $z = (z_1, \ldots, z_\ell) \in D_0$ such that $z_1, \ldots, z_\ell \in H$, we have $z \in \mathcal{Z}(\epsilon', r')$, where $\epsilon' = 6\ell|\mathbb{F}|\epsilon + \mu$ and $r' = 8\ell|\mathbb{F}|r$.*

Part 2. In the second part of the proof we show that when taking a large number of lines through a point $z \in D_X$, with high probability, there exists a value $v \in \mathbb{F}$, such that for most of these lines, the answers of Prover* correspond to low degree polynomials that "evaluate" the point z to v.

Lemma B.5. *There exists a negligible function $\mu = \mu(\lambda)$ such that for any $z \in D_X$ the following holds. Let $L_1, \ldots, L_\lambda : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_1, \ldots, L_\lambda\}$, we have $L(0) = z$. Let $S = \{L_j(t)\}_{j \in [\lambda], t \in \mathbb{F}} \subset D_X$. Let $(\varphi, A) \in_R \mathcal{A}_S$.*

For any $v \in \mathbb{F}$, define $A^v : S \to \mathbb{F}$ by $A^v(z') = A(z')$ for $z' \neq z$ and $A^v(z) = v$. Let $r' = 30|\mathbb{F}|r$ and let $\epsilon' = 2|\mathbb{F}|\epsilon + \mu$. Then, with probability $\geq 1 - \epsilon'$, there exists $v \in \mathbb{F}$, such that, for at least $\lambda - r'$ of the indices $j \in [\lambda]$, $A^v \circ L_j : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree less than $m|H|$ (where the probability is over $L_1, \ldots, L_\lambda, A$).

Part 3. In the third part of the proof we use Lemma B.4 and Lemma B.5 above to prove the local consistency guarantee.

Lemma B.6. *There exists a negligible function $\mu = \mu(\lambda)$ such that the following holds. Let $r' = 9\ell|\mathbb{F}|r$ and let $\epsilon' = 7\ell|\mathbb{F}|\epsilon + \mu$. Let $i_1, i_2, i_3 \in H^m$ and view i_1, i_2, i_3 as points in D_X.*

Let $L_{1,1}, \ldots, L_{1,\lambda} : \mathbb{F} \to D_X$ be λ uniformly random lines conditioned on the constraint that for every $L \in \{L_{1,1}, \ldots, L_{1,\lambda}\}$, we have $L(0) = i_1$. Let $L_{2,1}, \ldots, L_{2,\lambda} : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_{2,1}, \ldots, L_{2,\lambda}\}$, we have $L(0) = i_2$. Let $L_{3,1}, \ldots, L_{3,\lambda} : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_{3,1}, \ldots, L_{3,\lambda}\}$, we have $L(0) = i_3$.

Let $S = \{L_{1,j}(t), L_{2,j}(t), L_{3,j}(t)\}_{j \in [\lambda], t \in \mathbb{F}} \subset D_X$. Let $(\varphi, A) \in_R \mathcal{A}_S$. For any $i \in D_X$ and $v \in \mathbb{F}$, define $A^{i \to v} : S \to \mathbb{F}$ by $A^{i \to v}(i') = A(i')$ for $i' \neq i$ and $A^{i \to v}(i) = v$.

With probability $\geq 1 - \epsilon'$, there exist $v_1, v_2, v_3 \in \mathbb{F}$, such that, for at least $\lambda - r'$ of the indices $j \in [\lambda]$, the following is satisfied (where the probability is over $L_{1,1}, \ldots, L_{1,\lambda}, L_{2,1}, \ldots, L_{2,\lambda}, L_{3,1}, \ldots, L_{3,\lambda}, \varphi, A$):

1. *$A^{i_1 \to v_1} \circ L_{1,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
2. *$A^{i_2 \to v_2} \circ L_{2,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
3. *$A^{i_3 \to v_3} \circ L_{3,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
4. *If φ contains a clause of the form $(w_{i_1} = b_1) \vee (w_{i_2} = b_2) \vee (w_{i_3} = b_3)$, then $(v_1 - b_1) \cdot (v_2 - b_2) \cdot (v_3 - b_3) = 0$.*

Note that Lemma B.6, together with the computational no-signaling property, implies local soundness. As mentioned above, we prove this lemma using Lemma B.4 and Lemma B.5 above.

Claim B.6.1. *There exists a negligible function $\mu = \mu(\lambda) = \mathrm{negl}(\lambda)$ such that for every $\epsilon_1 \geq 0$, every $r_1 \geq 0$, every $i \in \{1, \ldots, \ell - 1\}$, and every $z \in D_i$, if $z \in \mathcal{Z}^{i+1}(\epsilon_1, r_1)$ then $z \in \mathcal{Z}^{i}(\epsilon_2, r_2)$, where $\epsilon_2 = \epsilon_1 + 2|\mathbb{F}|\epsilon + \mu(\lambda)$, and $r_2 = r_1 + 3|\mathbb{F}|r$.*

Claim B.6.2. *There exists a negligible function $\mu = \mu(\lambda) = \mathrm{negl}(\lambda)$ such that for every $\epsilon_1 \geq 0$, every $r_1 \geq 0$, every $i \in \{0, \ldots, \ell - 1\}$, and every $z \in D_i$, if $z \in \mathcal{Z}^{i+1}(\epsilon_1, r_1)$ then $z \in \mathcal{Z}(\epsilon_2, r_2)$, where $\epsilon_2 = \epsilon_1 + 2|\mathbb{F}|\epsilon + \mu(\lambda)$, and $r_2 = r_1 + 3|\mathbb{F}|r$.*

Claim B.6.3. *There exists a negligible function $\mu = \mu(\lambda) = \mathrm{negl}(\lambda)$ such that for every $\epsilon_1 \geq 0$, every $r_1 \geq 0$, and every $i \in \{1, \ldots, \ell\}$, the following holds: Let $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$ be a point such that $z_i \in H$. For every $t \in \mathbb{F}$, let $z(t) = (z_1, \ldots, z_{i-1}, t, z_{i+1}, \ldots, z_\ell) \in \mathbb{F}^\ell$. Assume that for every $t \in \mathbb{F}$, the point $z(t)$, viewed as a point in D_i, satisfies property $\mathcal{Z}^i(\epsilon_1, r_1)$. Then the point z, viewed as a point in D_{i-1}, satisfies property $\mathcal{Z}^i(\epsilon_2, r_2)$, where $\epsilon_2 = \frac{\epsilon_1}{1-\gamma} + |\mathbb{F}|\epsilon + \mu$, and $r_2 = \frac{r_1}{1-\gamma} + 2|\mathbb{F}|r$, and $\gamma = \sqrt{\frac{|H|}{|\mathbb{F}|}}$.*

Before we prove these claims, we prove Lemma B.4 based on these claims.

Proof of Lemma B.4. Recall that we assume that for every distribution $\mathcal{A}_S$ in the family $\{\mathcal{A}_S\}$, every query in $S \cap D_\ell$ is answered by 0 with probability 1 (since the polynomial P_ℓ was just the 0 polynomial and was added to the PCP proof for simplicity of notations). Therefore, any point $z \in D_\ell$ satisfies property $\mathcal{Z}^\ell(\epsilon_\ell, r_\ell)$, where $\epsilon_\ell = 0$ and $r_\ell = 0$.

Combining Claims B.6.1 and B.6.3 we obtain the following claim.

Claim B.6.4. *There exists a negligible function $\mu = \mu(\lambda)$ for which the following holds: Let $\epsilon_1 \geq 0$. Let $r_1 \geq 0$. Let $i \in \{2, \ldots, \ell\}$. Let $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$ be a point such that $z_i \in H$. For every $t \in \mathbb{F}$, let $z(t) = (z_1, \ldots, z_{i-1}, t, z_{i+1}, \ldots, z_\ell) \in \mathbb{F}^\ell$. Assume that for every $t \in \mathbb{F}$, the point $z(t)$, viewed as a point in D_i, satisfies property $\mathcal{Z}^i(\epsilon_1, r_1)$. Then the point z, viewed as a point in D_{i-1}, satisfies property $\mathcal{Z}^{i-1}(\epsilon_2, r_2)$, where $\epsilon_2 = \frac{\epsilon_1}{1-\gamma} + 3|\mathbb{F}|\epsilon + \mu$, and $r_2 = \frac{r_1}{1-\gamma} + 5|\mathbb{F}|r$, and $\gamma = \sqrt{\frac{|H|}{|\mathbb{F}|}}$.*

By inductive application of Claim B.6.4, for any $i \in \{1, \ldots, \ell - 1\}$, any point $z = (z_1, \ldots, z_\ell) \in D_i$, such that $z_{i+1}, \ldots, z_\ell \in H$, satisfies property $\mathcal{Z}^i(\epsilon_i, r_i)$, where $\epsilon_i = \frac{\epsilon_{i+1}}{1-\gamma} + 3|\mathbb{F}|\epsilon + \mu$ and $r_i = \frac{r_{i+1}}{1-\gamma} + 5|\mathbb{F}|r$, and $\gamma = \sqrt{\frac{|H|}{|\mathbb{F}|}}$.

In particular, any point $z = (z_1, \ldots, z_\ell) \in D_1$, such that $z_2, \ldots, z_\ell \in H$, satisfies property $\mathcal{Z}^1(\epsilon_1, r_1)$, where $\epsilon_1 \leq \frac{3\ell|\mathbb{F}|\epsilon + \ell \cdot \mu}{(1-\gamma)^\ell} < 4\ell|\mathbb{F}|\epsilon + 2\ell \cdot \mu$ and $r_1 \leq \frac{5\ell|\mathbb{F}|r}{(1-\gamma)^\ell} < 6\ell|\mathbb{F}|r$.

Hence, by Claim B.6.3, any point $z = (z_1, \ldots, z_\ell) \in D_0$, such that $z_1, \ldots, z_\ell \in H$, satisfies property $\mathcal{Z}^1(\epsilon_0, r_0)$, where $\epsilon_0 = \frac{\epsilon_1}{1-\gamma} + |\mathbb{F}|\epsilon + \mu < 5\ell|\mathbb{F}|\epsilon + 3\ell \cdot \mu$ and $r_0 = \frac{r_1}{1-\gamma} + 2|\mathbb{F}|r < 7\ell|\mathbb{F}|r$.

Finally, by Claim B.6.2, any point $z = (z_1, \ldots, z_\ell) \in D_0$, such that $z_1, \ldots, z_\ell \in H$, satisfies property $\mathcal{Z}(\epsilon', r')$, where $\epsilon' < 6\ell|\mathbb{F}|\epsilon + 4\ell \cdot \mu$ and $r' < 8\ell|\mathbb{F}|r$.

□

In what follows we prove Claims B.6.1 to B.6.3.

Proof of Claim B.6.1. Assume that $z \in \mathcal{Z}^{i+1}(\epsilon_1, r_1)$.

Let $L_1, \ldots, L_\lambda : \mathbb{F} \to D_i$ be λ random lines, such that for every $L \in \{L_1, \ldots, L_\lambda\}$ we have $L(0) = z$, and L is orthogonal to the $(i+1)^{st}$ coordinate. Let $L'_1, \ldots, L'_\lambda : \mathbb{F} \to D_i$ be λ random lines, such that for every $L' \in \{L'_1, \ldots, L'_\lambda\}$ we have $L'(0) = z$, and L' is orthogonal to the i^{th} coordinate.

Let $M_1, \ldots, M_\lambda : \mathbb{F}^2 \to D_i$ be λ planes, where $M_j(t_1, t_2) = L_j(t_1) + L'_j(t_2) - z$ (where the addition/substraction are over the vector space $D_i = \mathbb{F}^\ell$).

Let $S = \{M_j(t_1, t_2)\}_{j \in [k], t_1, t_2 \in \mathbb{F}} \subset D_i$. Let $(\varphi, A) \in_R \mathcal{A}_S$. Define $A^0 : S \to \mathbb{F}$ by $A^0(z') = A(z')$ for $z' \neq z$ and $A^0(z) = 0$.

We say that M_j is *good* if the following is satisfied:

1. For every $t_1 \in \mathbb{F} \setminus \{0\}$, the function $A^0 \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

2. For every $t_2 \in \mathbb{F}$, the function $A^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

By Claim B.6.5 below (applied with $f = A^0 \circ M_j$ and $d = 2\ell|H|$), if M_j is good then $A^0 \circ L'_j = A^0 \circ M_j(0, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Claim B.6.5. *Let $f : \mathbb{F}^2 \to \mathbb{F}$ be a function. Assume that for every $t_1 \in \mathbb{F} \setminus \{0\}$, the function $f_{(t_1,*)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$, and for every $t_2 \in \mathbb{F}$, the function $f_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$, where $d < |\mathbb{F}|$. Then, $f_{(0,*)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$.*

Proof. For every $t_2 \in \mathbb{F}$, the function $f_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$. Therefore, there exist $a_1, \ldots, a_d \in \mathbb{F}$, (where $a_1, \ldots, a_d$ are the Lagrange interpolation coefficients), such that for every $t_2 \in \mathbb{F}$, we have $f(0, t_2) = \sum_{t=1}^{d} a_t \cdot f(t, t_2)$. That is, $f_{(0,*)} = \sum_{t=1}^{d} a_t \cdot f_{(t,*)}$. Since $f_{(1,*)}, \ldots, f_{(d,*)}$ are univariate polynomials of degree $< d$, their linear combination $f_{(0,*)}$ is also a univariate polynomial of degree $< d$. □

We will show that with high probability, at least $\lambda - r_2$ of the planes $M \in \{M_1, \ldots, M_\lambda\}$ are good (where the probability is over $L_1, \ldots, L_\lambda, L'_1, \ldots, L'_\lambda, A$). By Claim B.6.5, this implies that with high probability, at least $\lambda - r_2$ of the lines $L' \in \{L'_1, \ldots, L'_\lambda\}$ satisfy that $A^0 \circ L' : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$ (where the probability is over $L_1, \ldots, L_\lambda, L'_1, \ldots, L'_\lambda, A$).

Claim B.6.6. *There exists a negligible function $\mu = \mu(\lambda)$ (independent of $i \in [\ell]$) such that with probability $\geq 1 - \epsilon_1 - 2|\mathbb{F}|\epsilon - \mu$, for at least $\lambda - r_1 - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that M_j is good.*

Proof. For every $t_1 \in \mathbb{F} \setminus \{0\}$, consider the set of lines $\{M_j(t_1, *)\}_{j\in[\lambda]}$ and note that this is a set of λ random lines in D_i, such that every line $L \in \{M_j(t_1, *)\}_{j\in[\lambda]}$ is orthogonal to the i^{th} coordinate, and satisfies $L(0)_{i+1} = z_{i+1}$. Hence, by the computational no-signaling condition, there exists a negligible function $\mu_1 = \mu_1(\lambda)$ (independent of $i \in [\ell]$) such that with probability $> 1 - \epsilon - \mu_1$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, we have that $A \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Denote by E_j the event that the lines L_j, L'_j are in general position, that is, the vectors $L_j(1) - L_j(0), L'_j(1) - L'_j(0)$ span a linear subspace of dimension 2 (as vectors in $D_i = \mathbb{F}^\ell$). Note that event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$. Moreover, note that if event E_j occurs then $z \notin M_j(t_1, *)$, and hence $A^0 \circ M_j(t_1, *) = A \circ M_j(t_1, *)$. Therefore, if $A \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$ and event E_j holds, then $A^0 \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is also a univariate polynomial of degree $< 2\ell|H|$.

For every $t_2 \in \mathbb{F} \setminus \{0\}$, consider the set of lines $\{M_j(*, t_2)\}_{j\in[\lambda]}$ and note that this is a set of λ random lines in D_i such that every line $L \in \{M_j(*, t_2)\}_{j\in[\lambda]}$ is orthogonal to the $(i + 1)^{th}$ coordinate, and satisfies $L(0)_i = z_i$. Hence, by the computational no-signaling condition, there exists a negligible function $\mu_2 = \mu_2(\lambda)$ (independent of $i \in [\ell]$) such that with probability $> 1 - \epsilon - \mu_2$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, we have that $A \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$. As above, if event E_j occurs then $z \notin M_j(*, t_2)$, and hence $A^0 \circ M_j(*, t_2) = A \circ M_j(*, t_2)$. Therefore, if $A \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$ and event E_j holds then $A^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is also a univariate polynomial of degree $< 2\ell|H|$.

Consider the set of lines $\{M_j(*, 0)\}_{j\in[\lambda]}$ and note that $M_j(*, 0) = L_j$. The fact that $z \in \mathcal{Z}^{i+1}(\epsilon_1, r_1)$, together with the computational no-signaling condition, implies that there exists a negligible function $\mu_3 = \mu_3(\lambda)$ (independent of $i \in [\ell]$) such that with probability $\geq 1 - \epsilon_1 - \mu_3$, for at least $\lambda - r_1$ of the indices $j \in [\lambda]$, we have that $A^0 \circ M_j(*, 0) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Recall that each event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$, and these events are independent. This, together with our assumption that

$$|\mathbb{F}|^{\ell-1} \geq |H|^{3\cdot(\ell-1)} \geq |H|^{3\cdot 3m} \geq \lambda^9$$

(where the latter inequality follows from our assumption that $|H|^m \geq \lambda$), implies that

$$\Pr\left[|\{j : \neg E_j\}| \geq |\mathbb{F}|\right] \leq \binom{\lambda}{|\mathbb{F}|} \cdot \left(\frac{1}{|\mathbb{F}|^{\ell-1}}\right)^{|\mathbb{F}|} \leq \left(\frac{e\lambda}{|\mathbb{F}|^\ell}\right)^{|\mathbb{F}|} \leq \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}.$$

Let $\mu_0 = \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}$. Note that $\mu_0 = \mathrm{negl}(\lambda)$.

Adding up all this, by the union bound, we obtain that with probability $\geq 1 - \epsilon_1 - 2|\mathbb{F}|\epsilon - \mu$, where $\mu = \mu_0 + \mu_1 + \mu_2 + \mu_3$, for at least $\lambda - r_1 - 2|\mathbb{F}|r - |\mathbb{F}| \geq \lambda - r_1 - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that:

1. For every $t_1 \in \mathbb{F}\setminus\{0\}$, $A^0 \circ M_j(t_1, *)$ is a univariate polynomial of degree $< 2\ell|H|$.

2. For every $t_2 \in \mathbb{F}\backslash\{0\}$, $A^0 \circ M_j(*, t_2)$ is a univariate polynomial of degree $< 2\ell|H|$.

3. $A^0 \circ M_j(*, 0)$ is a univariate polynomial of degree $< 2\ell|H|$.

That is, with probability at least $1 - \epsilon_1 - 2|\mathbb{F}|\epsilon - \mu$, for at least $\lambda - r_1 - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that M_j is good. □

Proof of Claim B.6.2. The proof of this claim is identical to the proof of Claim B.6.1, except that we let $L'_1, \ldots, L'_\lambda : \mathbb{F} \to D_i$ be λ random lines such that for every $L' \in \{L_1, \ldots, L_\lambda\}$ we have that $L'(0) = z$ (without the requirement that L' is orthogonal to the i^{th} coordinate). □

Proof of Claim B.6.3. Fix $\epsilon_1 \geq 0$, $r_1 \geq 0$, and $i \in \{1, \ldots, \ell\}$. Fix $z = (z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$ such that $z_i \in H$. Assume that for every $t \in \mathbb{F}$, the point $z(t) = (z_1, \ldots, z_{i-1}, t, z_{i+1}, \ldots, z_\ell)$, viewed as a point in D_i, satisfies property $\mathcal{Z}^i(\epsilon_1, r_1)$.

Let $L_1, \ldots, L_\lambda : \mathbb{F} \to \mathbb{F}^\ell$ be λ random lines, such that for every $L \in \{L_1, \ldots, L_\lambda\}$, we have $L(0) = 0$, and L is orthogonal to the i^{th} coordinate.

Let $M_1, \ldots, M_\lambda : \mathbb{F}^2 \to \mathbb{F}^\ell$ be λ planes, where $M_j(t_1, t_2) = L_j(t_1) + z(t_2)$ (where the addition is over the vector space $\mathbb{F}^\ell$).

Let S^i and S^{i-1} be two copies of the set of points $\{M_j(t_1, t_2)\}_{j \in [\lambda], t_1, t_2 \in \mathbb{F}} \subset \mathbb{F}^\ell$, and view S^i as a subset of D_i and S^{i-1} as a subset of D_{i-1}. Let $S = S^i \cup S^{i-1} \subset D$. Let $(\varphi, A) \in_R \mathcal{A}_S$. Recall that we view A as a function $A : S \to \mathbb{F}$, and we denote by A_i, A_{i-1} the restriction of that function to S^i, S^{i-1}, respectively.

Define $A^0_i : S^i \to \mathbb{F}$ by $A^0_i(z') = A_i(z')$ for $z' \notin \{z(t)\}_{t \in \mathbb{F}}$, and $A^0_i(z') = 0$ for $z' \in \{z(t)\}_{t \in \mathbb{F}}$. Define $A^0_{i-1} : S^{i-1} \to \mathbb{F}$ by $A^0_{i-1}(z') = A_{i-1}(z')$ for $z' \notin \{z(t)\}_{t \in \mathbb{F}}$ and $A^0_{i-1}(z') = 0$ for $z' \in \{z(t)\}_{t \in \mathbb{F}}$.

We say that M_j is *good* if the following is satisfied:

1. For every $t_1 \in \mathbb{F}$, and every $t \in \mathbb{F}$,

$$A^0_i(M_j(t_1, t)) = \sum_{h \in H} A^0_{i-1}(M_j(t_1, h)) t^h$$

2. For at least $|H|$ values $t_2 \in \mathbb{F}$, the function $A^0_i \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

By Claim B.6.7 below (applied with $f = A^0_i \circ M_j$, $f' = A^0_{i-1} \circ M_j$ and $d = 2\ell|H|$), if M_j is good then for every $t_2 \in H$, the function $A^0_{i-1} \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Claim B.6.7. *Let $f : \mathbb{F}^2 \to \mathbb{F}$ and $f' : \mathbb{F}^2 \to \mathbb{F}$ be two functions. Assume that:*

1. *For every $t_1 \in \mathbb{F}$ and every $t \in \mathbb{F}$,*

$$f(t_1, t) = \sum_{h \in H} f'(t_1, h) t^h$$

2. *For at least $|H|$ values $t_2 \in \mathbb{F}$, the function $f_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$.*

Then, for every $t_2 \in H$, the function $f'_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$.

Proof. For every $h \in H$, present the function $f'_{(*,h)} : \mathbb{F} \to \mathbb{F}$ as a univariate polynomial (in the free variable y),

$$f'(y,h) = f'_{(*,h)}(y) = \sum_{s=0}^{|\mathbb{F}|-1} a_{h,s} \cdot y^s$$

where $a_{h,0}, \ldots, a_{h,|\mathbb{F}|-1} \in \mathbb{F}$. Thus, for every $y \in \mathbb{F}$ and every $t \in \mathbb{F}$,

$$f_{(*,t)}(y) = f(y,t) = \sum_{h \in H} f'(y,h)t^h = \sum_{h \in H} \sum_{s=0}^{|\mathbb{F}|-1} a_{h,s} \cdot y^s \cdot t^h = \sum_{s=0}^{|\mathbb{F}|-1} \left(\sum_{h \in H} a_{h,s} \cdot t^h \right) \cdot y^s$$

Assume for a contradiction that for some $s \geq d$, the polynomial $\sum_{h \in H} a_{h,s} \cdot t^h$ is not the identically 0 polynomial, and let s be the largest such index. Since $\sum_{h \in H} a_{h,s} \cdot t^h$ is not identically 0, and its degree is $\leq |H| - 1$, it gives 0 on at most $|H| - 1$ values of $t \in \mathbb{F}$. Hence, the polynomial $f_{(*,t)}(y)$ is of degree $< s$ for at most $|H| - 1$ values of $t \in \mathbb{F}$, which is a contradiction to the assumption that for at least $|H|$ values $t \in \mathbb{F}$, the function $f_{(*,t)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$.

Thus, for every $s \geq d$, the polynomial $\sum_{h \in H} a_{h,s} \cdot t^h$ is the identically 0 polynomial. That is, for every $s \geq d$ and every $h \in H$ we have $a_{h,s} = 0$. Hence, for every $h \in H$, the function $f'_{(*,h)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$. □

We will show that with high probability, at least $\lambda - r_2$ of the planes $M \in \{M_1, \ldots, M_\lambda\}$ are good (where the probability is over $L_1, \ldots, L_\lambda, A$).

Claim B.6.8. *There exists a negligible function $\mu = \mu(\lambda)$ (independent of $i \in [\ell]$) such that with probability $\geq 1 - |\mathbb{F}|\epsilon - \frac{\epsilon_1}{1-\gamma} - \mu$, for at least $\lambda - 2|\mathbb{F}|r - \frac{r_1}{1-\gamma}$ of the indices $j \in [\lambda]$, we have that M_j is good, where $\gamma = \sqrt{\frac{|H|}{|\mathbb{F}|}}$.*

Proof. First note that for $t_1 = 0$,

$$A_i^0(M_j(t_1,t)) = \sum_{h \in H} A_{i-1}^0(M_j(t_1,h))t^h$$

is satisfied trivially (for every $j \in [\lambda]$, and every $t \in \mathbb{F}$), since $A_i^0 \circ M_j(0,*)$ and $A_{i-1}^0 \circ M_j(0,*)$ are the identically 0 function (by the definitions).

For every $t_1 \in \mathbb{F} \setminus \{0\}$, consider the set of points $\{M_j(t_1,0)\}_{j \in [\lambda]}$ and note that this is a set of λ random points in $\mathbb{F}^\ell$, such that the i^{th} coordinate of each of these points is 0 (that is, all other coordinates of all these points are uniformly distributed and independent random variables). Hence, by the computational no-signaling property, there exists a negligible function $\mu_1 = \mu_1(\lambda)$ (independent of $i \in [\ell]$) such that with

probability $> 1 - \epsilon - \mu_1$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, the following is satisfied for every $t \in \mathbb{F}$:

$$A_i(M_j(t_1, t)) = \sum_{h \in H} A_{i-1}(M_j(t_1, h)) t^h$$

For every $j \in [\lambda]$, denote by E_j the event that the line L_j is in a general position (as a line in $\mathbb{F}^\ell$), that is, its image is not just a single point. Note that event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$. Moreover, note that if event E_j occurs, then for every $t \in \mathbb{F}$ we have that $z(t) \notin M_j(t_1, t)$, and hence $A_i^0(M_j(t_1, t)) = A_i(M_j(t_1, t))$ and $A_{i-1}^0(M_j(t_1, t)) = A_{i-1}(M_j(t_1, t))$. Therefore, if event E_j occurs and $A_i(M_j(t_1, t)) = \sum_{h \in H} A_{i-1}(M_j(t_1, h)) t^h$ then

$$A_i^0(M_j(t_1, t)) = \sum_{h \in H} A_{i-1}^0(M_j(t_1, h)) t^h$$

(and recall that for $t_1 = 0$ this is satisfied trivially).

Recall that each event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$, and these events are independent. This, together with our assumption that

$$|\mathbb{F}|^{\ell-1} \geq |H|^{3 \cdot (\ell-1)} \geq |H|^{3 \cdot 3m} \geq \lambda^9$$

(where the latter inequality follows from our assumption that $|H|^m \geq \lambda$), implies that

$$\Pr\left[|\{j : \neg E_j\}| \geq |\mathbb{F}|\right] \leq \binom{\lambda}{|\mathbb{F}|} \cdot \left(\frac{1}{|\mathbb{F}|^{\ell-1}}\right)^{|\mathbb{F}|} \leq \left(\frac{e\lambda}{|\mathbb{F}|^\ell}\right)^{|\mathbb{F}|} \leq \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}.$$

Denote by $\mu_0 = \mu_0(\lambda) = \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}$, and note that $\mu_0 = \mathrm{negl}(\lambda)$. Adding up all this, by the union bound, with probability $> 1 - |\mathbb{F}|\epsilon - |\mathbb{F}|\mu_1 - \mu_0$, for at least $\lambda - |\mathbb{F}|r - |\mathbb{F}| \geq \lambda - 2|\mathbb{F}|r$ of the indices $j \in [\lambda]$, the following is satisfied for every $t_1 \in \mathbb{F}$ and every $t \in \mathbb{F}$:

$$A_i^0(M_j(t_1, t)) = \sum_{h \in H} A_{i-1}^0(M_j(t_1, h)) t^h \tag{B.2}$$

For every $t_2 \in \mathbb{F}$, consider the set of lines $\{M_j(*, t_2)\}_{j \in [\lambda]}$ and note that this is a set of λ random lines, such that for every $L \in \{M_j(*, t_2)\}_{j \in [\lambda]}$, we have $L(0) = z(t_2)$, and L is orthogonal to the i^{th} coordinate. The fact that $z(t_2)$, viewed as a point in D_i, satisfies property $\mathcal{Z}^i(\epsilon_1, r_1)$, together with the computational no-signaling property, implies that there exists a negligible function $\mu_2 = \mu_2(\lambda)$ (independent of $i \in [\lambda]$) such that with probability $\geq 1 - \epsilon_1 - \mu_2$, for at least $\lambda - r_1$ of the indices $j \in [\lambda]$, we have that $A_i^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Since this is true for every $t_2 \in \mathbb{F}$, by Claim B.6.9 below, applied with $\alpha = \epsilon_1 + \mu_2$, we obtain the following for any $\gamma < 1$:

With probability $\geq 1 - \frac{\epsilon_1 + \mu_2}{1 - \gamma}$, for at least $\gamma|\mathbb{F}|$ values $t_2 \in \mathbb{F}$ we have that for at least $\lambda - r_1$ of the indices $j \in [\lambda]$, the function $A_i^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Claim B.6.9. *Let $\{V_t\}_{t\in\mathbb{F}}$ be a set of events, such that, for every $t \in \mathbb{F}$, $\Pr[V_t] \geq 1-\alpha$. Then, for any $\gamma < 1$, with probability of at least $1-\frac{\alpha}{1-\gamma}$, at least $\gamma|\mathbb{F}|$ events in $\{V_t\}_{t\in\mathbb{F}}$ occur.*

Proof. Let I_t be the characteristic function of the event $\neg V_t$. Let $I = \sum_{t\in\mathbb{F}} I_t$. Thus, $\mathbb{E}[I] \leq \alpha|\mathbb{F}|$. By Markov's inequality, $\Pr[I > (1-\gamma)|\mathbb{F}|] < \alpha/(1-\gamma)$. Thus, with probability of at least $1-\alpha/(1-\gamma)$, at least $\gamma|\mathbb{F}|$ events in $\{V_t\}_{t\in\mathbb{F}}$ occur. □

Recall that with probability $\geq 1-\frac{\epsilon_1+\mu_2}{1-\gamma}$, for at least $\gamma|\mathbb{F}|$ values $t_2 \in \mathbb{F}$ we have that for at most r_1 of the indices $j \in [\lambda]$, the function $A_i^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is not a univariate polynomial of degree $< 2\ell|H|$.

Since in a $\{0,1\}$-matrix with $\gamma|\mathbb{F}|$ rows and $[\lambda]$ columns, with at most r_1 ones in each row, there are at most $\frac{\gamma|\mathbb{F}|r_1}{\gamma|\mathbb{F}|-|H|}$ columns with more than $\gamma|\mathbb{F}|-|H|$ ones (otherwise, the total number of ones is $> \gamma|\mathbb{F}|r_1$), this implies that with probability $\geq 1-\frac{\epsilon_1+\mu_2}{1-\gamma}$, for all but at most $\frac{\gamma|\mathbb{F}|r_1}{\gamma|\mathbb{F}|-|H|}$ indices $j \in [\lambda]$ we have that for at least $|H|$ of the values $t_2 \in \mathbb{F}$, the function $A_i^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

Combined with Equation (B.2), by the union bound, with probability $> 1-|\mathbb{F}|\epsilon-|\mathbb{F}|\mu_1-\mu_0-\frac{\epsilon_1+\mu_2}{1-\gamma}$, for at least $\lambda-2|\mathbb{F}|r-\frac{\gamma|\mathbb{F}|r_1}{\gamma|\mathbb{F}|-|H|}$ of the indices $j \in [\lambda]$, we have that:

1. For every $t_1 \in \mathbb{F}$ and every $t \in \mathbb{F}$:

$$A_i^0(M_j(t_1,t)) = \sum_{h\in H} A_{i-1}^0(M_j(t_1,h))t^h$$

2. For at least $|H|$ of the values $t_2 \in \mathbb{F}$ the function $A_i^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$.

That is, with probability $\geq 1-|\mathbb{F}|\epsilon-|\mathbb{F}|\mu_1-\mu_0-\frac{\epsilon_1+\mu_2}{1-\gamma}$, for at least $\lambda-2|\mathbb{F}|r-\frac{\gamma|\mathbb{F}|r_1}{\gamma|\mathbb{F}|-|H|}$ of the indices $j \in [\lambda]$, we have that M_j is good. In particular, for $\gamma = \sqrt{\frac{|H|}{|\mathbb{F}|}}$, we have that with probability $\geq 1-|\mathbb{F}|\epsilon-|\mathbb{F}|\mu_1-\mu_0-\frac{\epsilon_1+\mu_2}{1-\gamma}$, for at least $\lambda-2|\mathbb{F}|r-\frac{r_1}{1-\gamma}$ of the indices $j \in [\lambda]$, we have that M_j is good.

Setting $\mu = |\mathbb{F}|\mu_1+\mu_0+\frac{\mu_2}{1-\gamma}$, we conclude that with probability $\geq 1-|\mathbb{F}|\epsilon-\frac{\epsilon_1}{1-\gamma}-\mu$, for at least $\lambda-2|\mathbb{F}|r-\frac{r_1}{1-\gamma}$ of the indices $j \in [\lambda]$, we have that M_j is good. □

By Claim B.6.7, Claim B.6.8 implies that with probability $\geq 1-|\mathbb{F}|\epsilon-\frac{\epsilon_1}{1-\gamma}-\mu$, at least $\lambda-r_2$ of the indices $j \in [\lambda]$ satisfy that for every $t_2 \in H$, the function $A_{i-1}^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$, (where the probability is over $L_1, \ldots, L_\lambda, A$).

Fix $t_2 = z_i$. Consider the set of lines $\{M_j(*, t_2)\}_{j\in[\lambda]}$ and note that this is a set of λ random lines, such that for every $L \in \{M_j(*, t_2)\}_{j\in[\lambda]}$, we have $L(0) = z(t_2) = z$, and L is orthogonal to the i^{th} coordinate.

Thus, by the above, with probability $\geq 1-|\mathbb{F}|\epsilon-\frac{\epsilon_1}{1-\gamma}-\mu$, at least $\lambda-r_2$ of the indices $j \in [\lambda]$ satisfy that the function $A_{i-1}^0 \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 2\ell|H|$. Thus, by the computational no-signaling property,

there exists a negligible function $\mu_3 = \mu_3(\lambda)$ (independent of $i \in [\lambda]$) such that the point z, viewed as a point in D_{i-1}, satisfies property $\mathcal{Z}^i(\epsilon_2, r_2)$, with $\epsilon_2 = |\mathbb{F}|\epsilon - \frac{\epsilon_1}{1-\gamma} - \mu^*$, for $\mu^* = \mu + \mu_3$.

This concludes the proof of Claim B.6.3. □

□

B.3 Proof of Lemma B.5

Proof. Fix $z \in D_X$. Let $L_1, \ldots, L_\lambda : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_1, \ldots, L_\lambda\}$, we have $L(0) = z$.

Let $L'_1, \ldots, L'_\lambda : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L'_1, \ldots, L'_\lambda\}$, we have $L(0) = z$. Let $M_1, \ldots, M_\lambda : \mathbb{F}^2 \to D_X$ be λ planes, where $M_j(t_1, t_2) = L_j(t_1) + L'_j(t_2) - z$ (where the addition/substraction are over the vector space $D_X = \mathbb{F}^m$).

Let $S = \{M_j(t_1, t_2)\}_{j \in [\lambda], t_1, t_2 \in \mathbb{F}} \subset D_X$. Let $(\varphi, A) \in_R \mathcal{A}_S$. For any $v \in \mathbb{F}$, define $A^v : S \to \mathbb{F}$ by $A^v(z') = A(z')$ for $z' \neq z$ and $A^v(z) = v$.

We say that M_j is *good* if the following is satisfied:

1. For every $t_1 \in \mathbb{F} \setminus \{0\}$, the function $A \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

2. For every $t_2 \in \mathbb{F} \setminus \{0\}$, the function $A \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

For every $j \in [\lambda]$, let E_j denote the event that the lines L_j and L'_j are in general position; i.e., the vectors $L_j(1) - L_j(0)$ and $L'_j(1) - L'_j(0)$ span a linear subspace of dimension two. Note that each event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$

Claim B.6.10 below (applied with $f = A \circ M_j$ and $d = m|H|$) implies that if M_j is good and event E_j holds, then there exists $v \in \mathbb{F}$, such that, $A^v \circ L_j = A^v \circ M_j(*, 0) : \mathbb{F} \to \mathbb{F}$ and $A^v \circ L'_j = A^v \circ M_j(0, *) : \mathbb{F} \to \mathbb{F}$ are both univariate polynomials of degree $< m|H|$.

Claim B.6.10. *Let $f : \mathbb{F}^2 \to \mathbb{F}$ be a function. Assume that for every $t_1 \in \mathbb{F} \setminus \{0\}$, the function $f_{(t_1,*)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$, and for every $t_2 \in \mathbb{F} \setminus \{0\}$, the function $f_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$, where $d < |\mathbb{F}|$. For any $v \in \mathbb{F}$, define $f^v : \mathbb{F}^2 \to \mathbb{F}$ by $f^v(t_1, t_2) = f(t_1, t_2)$ for $(t_1, t_2) \neq (0, 0)$ and $f^v(0, 0) = v$. Then, there exists $v \in \mathbb{F}$, such that, $f^v_{(0,*)} : \mathbb{F} \to \mathbb{F}$ and $f^v_{(*,0)} : \mathbb{F} \to \mathbb{F}$ are both univariate polynomials of degree $< d$.*

Proof. For every $t_2 \in \mathbb{F} \setminus \{0\}$, the function $f_{(*,t_2)} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< d$. Therefore, there exist $a_1, \ldots, a_d \in \mathbb{F}$, (where $a_1, \ldots, a_d$ are the Lagrange interpolation coefficients), such that for every $t_2 \in \mathbb{F} \setminus \{0\}$, we have $f(0, t_2) = \sum_{t=1}^d a_t \cdot f(t, t_2)$. Since $f^v(t_1, t_2) = f(t_1, t_2)$ for $(t_1, t_2) \neq (0, 0)$, this implies that for every $t_2 \in \mathbb{F} \setminus \{0\}$ and every $v \in \mathbb{F}$, we have $f^v(0, t_2) = \sum_{t=1}^d a_t \cdot f^v(t, t_2)$.

Let $v = \sum_{t=1}^d a_t \cdot f(t, 0)$. Since $f^v(0, 0) = v$, we now have for every $t_2 \in \mathbb{F}$ (including $t_2 = 0$), $f^v(0, t_2) = \sum_{t=1}^d a_t \cdot f^v(t, t_2)$. That is, $f^v_{(0,*)} = \sum_{t=1}^d a_t \cdot f^v_{(t,*)}$. Since

$f^v_{(1,*)}, \ldots, f^v_{(d,*)}$ are identical to $f_{(1,*)}, \ldots, f_{(d,*)}$ and are hence univariate polynomials of degree $< d$, their linear combination $f^v_{(0,*)}$ is also a univariate polynomial of degree $< d$.

The proof now follows from Claim B.6.5, applied on the function f^v (with variables t_1, t_2 switched).

□

We show (in Claim B.6.11 below) that with high probability, for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, the event E_j holds and the plane M_j is good (where the probability is over $L_1, \ldots, L_k, L'_1, \ldots, L'_k, A$). By Claim B.6.10, this implies that with high probability, for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, there exists $v \in \mathbb{F}$ (which may depend on j) such that $A^v \circ L_j = A^v \circ M_j(*, 0) : \mathbb{F} \to \mathbb{F}$ and $A^v \circ L'_j = A^v \circ M_j(0, *) : \mathbb{F} \to \mathbb{F}$ are both univariate polynomials of degree $< m|H|$ (where the probability is over $L_1, \ldots, L_k, L'_1, \ldots, L'_k, A$).

Claim B.6.11. *There exists a negligible function $\mu = \mathrm{negl}(\lambda)$ such that with probability $\geq 1 - 2|\mathbb{F}|\epsilon - \mu$, for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that event E_j holds and M_j is good.*

Proof. Recall that each event E_j occurs with probability $1 - \frac{1}{|\mathbb{F}|^{\ell-1}}$, and these events are independent.

This, together with our assumption that

$$|\mathbb{F}|^{\ell-1} \geq |H|^{3\cdot(\ell-1)} \geq |H|^{3\cdot 3m} \geq \lambda^9$$

(where the latter inequality follows from our assumption that $|H|^m \geq \lambda$), implies that

$$\Pr\left[|\{j : \neg E_j\}| \geq |\mathbb{F}|\right] \leq \binom{\lambda}{|\mathbb{F}|} \cdot \left(\frac{1}{|\mathbb{F}|^{\ell-1}}\right)^{|\mathbb{F}|} \leq \left(\frac{e\lambda}{|\mathbb{F}|^{\ell}}\right)^{|\mathbb{F}|} \leq \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}.$$

Denote by $\mu_0 = \mu_0(\lambda) = \left(\frac{1}{\lambda^8}\right)^{|\mathbb{F}|}$, and note that $\mu_0 = \mathrm{negl}(\lambda)$.

For every $t_1 \in \mathbb{F} \setminus \{0\}$, consider the set of lines $\{M_j(t_1, *)\}_{j\in[\lambda]}$ and note that this is a set of λ random lines in D_X. Hence, by the computational no signaling, there exists a negligible function $\mu_1 = \mu_1(\lambda)$ such that with probability $> 1 - \epsilon - \mu_1$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, we have that $A \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

For every $t_2 \in \mathbb{F} \setminus \{0\}$, consider the set of lines $\{M_j(*, t_2)\}_{j\in[\lambda]}$ and note that this is a set of λ random lines in D_X. Hence, by the computational no-signaling property, with probability $> 1 - \epsilon - \mu_1$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, we have that $A \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

Adding up these facts, by the union bound, we obtain that with probability $\geq 1 - 2|\mathbb{F}|\epsilon - 2|\mathbb{F}|\mu_1$, for at least $\lambda - 2|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that:

1. For every $t_1 \in \mathbb{F} \setminus \{0\}$, $A \circ M_j(t_1, *) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

2. For every $t_2 \in \mathbb{F} \setminus \{0\}$, $A \circ M_j(*, t_2) : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

That is, with probability at least $1 - 2|\mathbb{F}|\epsilon - 2|\mathbb{F}|\mu_1$, for at least $\lambda - 2|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that M_j is good.

By setting $\mu = 2|\mathbb{F}|\mu_1 + \mu_0$ we conclude that with probability $\geq 1 - 2|\mathbb{F}|\epsilon - \mu$ for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that event E_j holds and M_j is good, as desired.

□

So far we proved that with probability at least $1 - 2|\mathbb{F}|\epsilon - \mu$ it holds that for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$ there exists a value $v \in \mathbb{F}$ (which may depend on j), such that $A^v \circ L_j : \mathbb{F} \to \mathbb{F}$ and $A^v \circ L'_j : \mathbb{F} \to \mathbb{F}$ are univariate polynomials of degree $< m|H|$.

To conclude the proof, we need to prove that there exists a negligible function μ' such that with probability at least $1 - 2|\mathbb{F}|\epsilon - \mu'$ there exists a single value $v \in \mathbb{F}$ such that for at least $\lambda - r'$ of the indices $j \in [\lambda]$ it holds that $A^v \circ L_j$ is univariate polynomials of degree less than $m \cdot |H|$, where $r' = 30|\mathbb{F}|r$.

To this end, denote by E the event that indeed there exists $v \in \mathbb{F}$, such that for at least $\lambda - r'$ of the indices $j \in [\lambda]$, $A^v \circ L_j : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$. We need to prove that there exists a negligible function μ' such that $\Pr[E] \geq 1 - 2|\mathbb{F}|\epsilon - \mu'$ (where the probability is over $L_1, \ldots, L_{2\lambda}, A$).

Denote by E' the event that for at least $\lambda - 3|\mathbb{F}|r$ of the indices $j \in [\lambda]$, there exists $v \in \mathbb{F}$ (that may depend on j), such that both $A^v \circ L_j : \mathbb{F} \to \mathbb{F}$ and $A^v \circ L'_j : \mathbb{F} \to \mathbb{F}$ are univariate polynomials of degree $< m|H|$. By Claim B.6.11,

$$\Pr[E'] \geq 1 - 2|\mathbb{F}|\epsilon - \mu.$$

Claim B.6.12. $\Pr[E' \mid \neg E] = \text{negl}(\lambda)$

Proof. In what follows, to simplify notation, we denote the random lines $L'_1, \ldots, L'_\lambda$ by $L_{\lambda+1}, \ldots, L_{2\lambda}$, and consider the 2λ lines $L_1, \ldots, L_{2\lambda}$. Note that these are 2λ random lines such that for every $j \in [2\lambda]$, it holds that $L_j : \mathbb{F} \to D_X$ and $L_j(0) = z$.

For every $v \in \mathbb{F}$, let J_v be the set of indices $j \in [2\lambda]$ such that $A^v \circ L_j : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$. Note that for every $v \neq v' \in \mathbb{F}$,

$$J_v \cap J_{v'} = \emptyset.$$

If event $\neg E$ occurs then for every $v \in \mathbb{F}$,

$$|J_v| < 2\lambda - r'.$$

Denote by J the largest set J_v and by $\bar{J}$ the complement of J in $[2\lambda]$. Thus, if event $\neg E$ occurs then $|\bar{J}| > r'$.

Note that given the sets $\{J_v\}_{v\in\mathbb{F}}$, the probability that E' occurs is the probability that when partitioning $[2\lambda]$ randomly into λ pairs, for at least $\lambda - 3|\mathbb{F}|r$ pairs the two indices in the pair are in the same set J_v. Assuming that $|\bar{J}| > r'$, this probability can be bounded by $2^{-|\mathbb{F}|r}$ by the following argument:

Choose the partition as follows: First choose randomly $\lambda' = r'/2 = 15|\mathbb{F}|r$ different indices $j_1, \ldots, j_{\lambda'}$ in $\bar{J}$. Match the indices $j_1, \ldots, j_{\lambda'}$ one by one, each to a random index in $[2\lambda]$ that was still not chosen. Say that $j_t \in \{j_1, \ldots, j_{\lambda'}\}$ is good if it was matched to an index in a set J_v such that $j_t \in J_v$. Finally, extend the partial partition randomly into a partition of $[2\lambda]$ into λ pairs. Note that the probability for an index j_t to be good is at most $\frac{\lambda}{2\lambda - r'} < 0.51$, independently of all previous choices of indices. Thus, the probability that at least $\lambda' - 3|\mathbb{F}|r$ indices $j_t \in \{j_1, \ldots, j_{\lambda'}\}$ are good is at most

$$\binom{\lambda'}{3|\mathbb{F}|r} \cdot 0.51^{\lambda' - 3|\mathbb{F}|r} \leq \left(\frac{\lambda' \cdot e}{3|\mathbb{F}|r}\right)^{3|\mathbb{F}|r} \cdot 0.51^{12|\mathbb{F}|r} = \left((5e)^3 \cdot 0.51^{12}\right)^{|\mathbb{F}|r} \leq 0.8^{|\mathbb{F}|r} = \mathrm{negl}(\lambda),$$

where the first inequality follows from the standard inequality that $\binom{n}{k} \leq \left(\frac{n \cdot e}{k}\right)^k$, and the second to last inequality follows from basic calculations.

Thus, $\Pr[E' \mid \neg E] = \mathrm{negl}(\lambda)$. □

We can now bound

$$1 - 2|\mathbb{F}|\epsilon - \mu \leq \Pr[E'] \leq \Pr[E' \mid \neg E] + \Pr[E]$$

Thus,

$$\Pr[E] > 1 - 2|\mathbb{F}|\epsilon - \mu - \Pr[E' \mid \neg E].$$

Let $\mu' = \mu + \Pr[E' \mid \neg E]$. Thus,

$$\Pr[E] > 1 - 2|\mathbb{F}|\epsilon - \mu',$$

and by Claim B.6.12, μ' is indeed a negligible function, as desired. □

B.4 Proof of Lemma B.6

Proof. Since φ is polynomially-sized, it suffices to fix any $i_1, i_2, i_3 \in H^m$ and $b_1, b_2, b_3 \in \{0,1\}$ and view i_1, i_2, i_3 as points in D_X. Let $L_{1,1}, \ldots, L_{1,\lambda} : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_{1,1}, \ldots, L_{1,\lambda}\}$ we have $L(0) = i_1$. Let $L_{2,1}, \ldots, L_{2,\lambda} : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_{2,1}, \ldots, L_{2,\lambda}\}$ we have $L(0) = i_2$. Let $L_{3,1}, \ldots, L_{3,\lambda} : \mathbb{F} \to D_X$ be λ random lines, such that for every $L \in \{L_{3,1}, \ldots, L_{3,\lambda}\}$ we have $L(0) = i_3$. Let

$$S^X = \{L_{1,j}(t), L_{2,j}(t), L_{3,j}(t)\}_{j\in[\lambda], t\in\mathbb{F}}.$$

Let $z = (i_1, i_2, i_3, b_1, b_2, b_3) \in H^\ell$. Let $L^1_{b_1,b_2,b_3}, \ldots, L^\lambda_{b_1,b_2,b_3} : \mathbb{F} \to D_0$ be λ random

lines such that for every $j \in [\lambda]$ the following holds:

1. $L^j_{b_1,b_2,b_3}(0) = z$.
2. $L_{1,j} : \mathbb{F} \to D_X$ is the restriction of $L^j_{b_1,b_2,b_3}$ to coordinates $\{1,\ldots,m\}$.
3. $L_{2,j} : \mathbb{F} \to D_X$ is the restriction of $L^j_{b_1,b_2,b_3}$ to coordinates $\{m+1,\ldots,2m\}$.
4. $L_{3,j} : \mathbb{F} \to D_X$ is the restriction of $L^j_{b_1,b_2,b_3}$ to coordinates $\{2m+1,\ldots,3m\}$.

Let

$$S^0_{b_1,b_2,b_3} = \left\{L^j_{b_1,b_2,b_3}(t)\right\}_{j\in[\lambda],t\in\mathbb{F}}.$$

Let $S^0 = \{S^0_{b_1,b_2,b_3}\}_{b_1,b_2,b_3\in\{0,1\}}$ and let $S = S^0 \cup S^X \subset D$. Let $(\varphi, A) \in_R \mathcal{A}_S$. We denote $A = A_X \cup A_0$, where A_X corresponds to the answers corresponding to the queries in S^X and A_0 are the answers corresponding to the queries in S^0.

In what follows, for any $i \in D_X$ and $v \in \mathbb{F}$, we define $A_X^{i\to v} : S^X \to \mathbb{F}$ by $A_X^{i\to v}(i') = A_X(i')$ for $i' \neq i$ and $A_X^{i\to v}(i) = v$.

Claim B.6.13. *There exists a negligible function μ such that with probability $\geq 1 - 7\ell|\mathbb{F}|\epsilon - \mu$, there exist $v_1, v_2, v_3 \in \mathbb{F}$ such that for at least $\lambda - 9\ell|\mathbb{F}|r$ of the indices $j \in [\lambda]$, the following is satisfied:*

1. *$A_X^{i_1\to v_1} \circ L_{1,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
2. *$A_X^{i_2\to v_2} \circ L_{2,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
3. *$A_X^{i_3\to v_3} \circ L_{3,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.*
4. *$\phi(i_1,i_2,i_3,b_1,b_2,b_3)\cdot(v_1-b_1)\cdot(v_2-b_2)\cdot(v_3-b_3) = 0$.*

By computational no-signaling requirement, Claim B.6.13 immediately implies Lemma B.6. Thus, in the remaining of the proof we focus on proving Claim B.6.13.

Let $A_0^0 \circ L^j_{b_1,b_2,b_3} : \mathbb{F} \to \mathbb{F}$ be the function defined by $A_0^0 \circ L^j_{b_1,b_2,b_3}(0) = 0$ and $A_0^0 \circ L^j_{b_1,b_2,b_3}(t) = A_0 \circ L^j_{b_1,b_2,b_3}(t)$ for every $t \neq 0$. By Lemma B.4, together with the computational no-signaling property, there exists a negligible function $\mu_1 = \mu_1(\lambda)$, such that with probability $\geq 1 - 6\ell|\mathbb{F}|\epsilon - \mu_1$, for at least $\lambda - 8\ell|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that

$$A_0^0 \circ L^j_{b_1,b_2,b_3} : \mathbb{F} \to \mathbb{F}$$

is a univariate polynomial of degree $< 2\ell|H|$.

By Lemma B.5, together with the computational no-signaling property, there exists a negligible function $\mu_2 = \mu_2(\lambda)$, such that the following holds:

1. With probability $\geq 1 - 2|\mathbb{F}|\epsilon - \mu_2$, there exists $v_1 \in \mathbb{F}$, such that, for at least $\lambda - 30|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that $A_X^{i_1\to v_1} \circ L^1_j : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

2. With probability $\geq 1 - 2|\mathbb{F}|\epsilon - \mu_2$, there exists $v_2 \in \mathbb{F}$, such that, for at least $\lambda - 30|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that $A_X^{i_2 \to v_2} \circ L_j^2 : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

3. With probability $\geq 1 - 2|\mathbb{F}|\epsilon - \mu_2$, there exists $v_3 \in \mathbb{F}$, such that, for at least $\lambda - 30|\mathbb{F}|r$ of the indices $j \in [\lambda]$, we have that $A_X^{i_3 \to v_3} \circ L_j^3 : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.

For every $t \in \mathbb{F} \setminus \{0\}$, consider the set of points $\left\{L^j_{b_1,b_2,b_3}(t)\right\}_{j\in[\lambda]}$ and note that this is a set of λ random points in D_0. Each point $L^j_{b_1,b_2,b_3}(t) \in \mathbb{F}^\ell$ can be written as

$$L^j_{b_1,b_2,b_3}(t) = \left(L_{1,j}(t), L_{2,j}(t), L_{3,j}(t), \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-2}, \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-1}, \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell}\right)$$

in $(\mathbb{F}^m)^3 \times \mathbb{F}^3 = \mathbb{F}^\ell$, where $\left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-2}, \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-1}, \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell}$ are the last 3 coordinates of $L^j_{b_1,b_2,b_3}(t)$. By the computational no-signaling condition, there exists a negligible function $\mu_3 = \mu_3(\lambda)$ such that for every $t \in \mathbb{F} \setminus \{0\}$, with probability $> 1 - \epsilon - \mu_3$, for at least $\lambda - r$ of the indices $j \in [\lambda]$, we have

$$\begin{aligned} A_0\left(L^j_{b_1,b_2,b_3}(t)\right) = \hat{\phi}\left(L^j_{b_1,b_2,b_3}(t)\right) \\ \cdot \left(A_X(L_{1,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-2}\right) \\ \cdot \left(A_X(L_{2,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-1}\right) \\ \cdot \left(A_X(L_{3,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell}\right) \end{aligned}$$

For every $j \in [\lambda]$ denote by E_j the event that for every $w \in \{1,2,3\}$ the line $L_{w,j}$ is in a general position (as a line in $\mathbb{F}^m$); i.e., it's image is not a single point. Note that event E_j occurs with probability at least $1 - \frac{3}{|\mathbb{F}|^{m-1}}$.

Moreover, note that if event E_j holds then for every $t \in \mathbb{F} \setminus \{0\}$ it holds that $L_{1,j}(t) \neq i_1$ and $L_{2,j}(t) \neq i_2$ and $L_{3,j}(t) \neq i_3$. In particular, $L^j_{b_1,b_2,b_3}(t) \neq z$. Thus, if the equation above holds and event E_j holds, then for every $v_1, v_2, v_3 \in \mathbb{F}$,

$$\begin{aligned} A_0^0\left(L^j_{b_1,b_2,b_3}(t)\right) = \hat{\phi}\left(L^j_{b_1,b_2,b_3}(t)\right) \\ \cdot \left(A_X^{i_1 \to v_1}(L_{1,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-2}\right) \\ \cdot \left(A_X^{i_2 \to v_2}(L_{2,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-1}\right) \\ \cdot \left(A_X^{i_3 \to v_3}(L_{3,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell}\right) \end{aligned}$$

Our assumption that $|H|^m \geq \lambda$, together with the assumption that $|\mathbb{F}| \geq |H|^3$, implies that

$$|\mathbb{F}|^{m-1} \geq \lambda^2,$$

which implies that

$$\Pr\left[|\{j : \neg E_j\}| \geq |\mathbb{F}|\right] \leq \binom{\lambda}{|\mathbb{F}|} \cdot \left(\frac{3}{|\mathbb{F}|^{m-1}}\right)^{|\mathbb{F}|} \leq \left(\frac{3e\lambda}{|\mathbb{F}|^m}\right)^{|\mathbb{F}|} \leq \left(\frac{3}{\lambda}\right)^{|\mathbb{F}|}.$$

Let $\mu_4 = \left(\frac{3}{\lambda}\right)^{|\mathbb{F}|}$. Adding up all this, by the union bound, we obtain that with probability $\geq 1 - \epsilon'$, where

$$\epsilon' = 6\ell|\mathbb{F}|\epsilon + \mu_1 + 3(2|\mathbb{F}|\epsilon + \mu_2) + |\mathbb{F}|(\epsilon + \mu_3) + \mu_4 \leq 7\ell|\mathbb{F}|\epsilon + \mu$$

(and where $\mu = \mu_1 + 3\mu_2 + |\mathbb{F}|\mu_3 + \mu_4$) there exist $v_1, v_2, v_3 \in \mathbb{F}$, such that, for at least $\lambda - r'$ of the indices $j \in [\lambda]$, where

$$r' = 8\ell|\mathbb{F}|r + 30|\mathbb{F}|r + r + |\mathbb{F}| \leq 9\ell|\mathbb{F}|r,$$

the following is satisfied (where the probability is over $L_1, \ldots, L_\lambda, A$):

1. $A_0^0 \circ L^j_{b_1,b_2,b_3} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< 3\ell|H|$.
2. $A_X^{i_1 \to v_1} \circ L_{1,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.
3. $A_X^{i_2 \to v_2} \circ L_{2,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.
4. $A_X^{i_3 \to v_3} \circ L_{3,j} : \mathbb{F} \to \mathbb{F}$ is a univariate polynomial of degree $< m|H|$.
5. For every $t \in \mathbb{F} \setminus \{0\}$,

$$\begin{aligned} A_0^0\left(L^j_{b_1,b_2,b_3}(t)\right) = \hat{\phi}\left(L^j_{b_1,b_2,b_3}(t)\right) \\ \cdot \left(A_X^{i_1 \to v_1}(L_{1,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-2}\right) \\ \cdot \left(A_X^{i_2 \to v_2}(L_{2,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell-1}\right) \\ \cdot \left(A_X^{i_3 \to v_3}(L_{3,j}(t)) - \left(L^j_{b_1,b_2,b_3}(t)\right)_{\ell}\right) \end{aligned}$$

Note that since both sides of the equation are polynomials of degree $< |\mathbb{F}|$ in the variable t, the equation must be satisfied for $t = 0$ as well. Substituting $t = 0$, since $L^j_{b_1,b_2,b_3}(0) = z = (i_1, i_2, i_3, b_1, b_2, b_3)$, we have

$$0 = \phi(z) \cdot (v_1 - b_1) \cdot (v_2 - b_2) \cdot (v_3 - b_3),$$

as desired. □

Bibliography

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[ACC+16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM computations with adaptive soundness and privacy. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 3–30, 2016.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, pages 166–175. IEEE Computer Society, 2004.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 162–172. Springer, 2014.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In *TCC (A1)*, volume 9562 of *Lecture Notes in Computer Science*, pages 125–153. Springer, 2016.

[BBK+16] Nir Bitansky, Zvika Brakerski, Yael Tauman Kalai, Omer Paneth, and Vinod Vaikuntanathan. 3-message zero knowledge against human ignorance. In *TCC (B1)*, volume 9985 of *Lecture Notes in Computer Science*, pages 57–83, 2016.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptology*, 30(4):989–1066, 2017.

[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. ACM, 2013.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73. Springer, 2014.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31. ACM, 1991.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *STOC*, pages 439–448. ACM, 2015.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, pages 474–482. ACM, 2017.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153. Springer, 2012.

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the integrity of agent-based computations by short proofs. In *Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 1998.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300, 2013.

[CCC+16] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel RAM from indistinguishability obfuscation. In *ITCS*, pages 179–190. ACM, 2016.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled RAM or: How to delegate your database. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 61–90, 2016.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, pages 209–218. ACM, 1998.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In *ITCS*, pages 169–178. ACM, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *STOC*, pages 429–437. ACM, 2015.

[CHSH69] John F Clauser, Michael A Horne, Abner Shimony, and Richard A Holt. Proposed experiment to test local hidden-variable theories. *Physical review letters*, 23(15):880, 1969.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[CP13] Kai-Min Chung and Rafael Pass. A simple ORAM. *IACR Cryptology ePrint Archive*, 2013:243, 2013.

[Dam89] Ivan Damgård. A design principle for hash functions. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 54–74. Springer, 2012.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *CRYPTO (3)*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122. Springer, 2016.

[DLN+01] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for np and spooky interactions. Unpublished manuscript, 2001.

[DNR16] Cynthia Dwork, Moni Naor, and Guy N. Rothblum. Spooky interaction and its discontents: Compilers for succinct two-message argument systems. In *CRYPTO (3)*, volume 9816 of *Lecture Notes in Computer Science*, pages 123–145. Springer, 2016.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *EUROCRYPT*, pages 405–422, 2014.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113. IEEE Computer Society, 2003.

[GK16] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. In *TCC (A1)*, volume 9562 of *Lecture Notes in Computer Science*, pages 505–522. Springer, 2016.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2013.

[GKR08a] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122. ACM, 2008.

[GKR08b] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In Ronitt Rubinfeld, editor, *Symposium on the Theory of Computing (STOC)*, 2015.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In *CRYPTO (3)*, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178. Springer, 2016.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HW15] Pavel Hubáček and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, pages 163–172. ACM, 2015.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304. IEEE Computer Society, 2000.

[IKO05] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2005.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732. ACM, 1992.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, pages 419–428. ACM, 2015.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 91–118, 2016.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, pages 669–684, 2013.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2009.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, pages 565–574, 2013.

[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.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs? In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology âĂŞ EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 719–734. Springer Berlin Heidelberg, 2013.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT (1)*, volume 9452 of *Lecture Notes in Computer Science*, pages 121–145. Springer, 2015.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *TCC (2)*, volume 10678 of *Lecture Notes in Computer Science*, pages 283–315. Springer, 2017.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, pages 49–62. ACM, 2016.

[SW] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.