Spooky Interaction and its Discontents: Compilers for Succinct Two-Message Argument Systems

Cynthia Dwork* Moni Naor† Guy N. Rothblum‡

Abstract

We are interested in constructing short two-message arguments for various languages, where the complexity of the verifier is small (e.g. linear in the input size, or even sublinear if the input is coded appropriately).

In 2000 Aiello et al. suggested the tantalizing possibility of obtaining such arguments for all of NP. These have proved elusive, despite extensive efforts. Our work builds on the compiler of Kalai and Raz, which takes as input an interactive proof system consisting of several rounds and produces a two-message argument system. The proof of soundness of their compiler relies on superpolynomial hardness assumptions.

In this work we obtain a succinct two-message argument system for any language in NC, where the verifier's work is linear (or even polylogarithmic). This is the first non trivial two-message succinct argument system that is based on a standard polynomial-time hardness assumption. We obtain this result by showing that the compiler is sound if the verifier in the original protocol runs in logarithmic space and public coins. On the other hand, we prove that under standard assumptions there is a sound interactive proof protocol that, when run through the compiler, results in a protocol that is not sound.

*Microsoft Research Silicon Valley.

†Dept of Computer Science and Applied Math, Weizmann Institute of Science. Incumbent of the Judith Kleeman Professorial Chair. Research supported in part by grants from the Israel Science Foundation, BSF and Israeli Ministry of Science and Technology and from the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation (grant No. 4/11). Part of this work was done while visiting Microsoft Research.

‡Samsung Research America.

1 Introduction

Imagine going on vacation and upon your return you find that not only has your home computer ordered the garden robot to mow the lawn but it has also commissioned from some vendor a lengthy computation that you have been postponing for a while. While verifying that the lawn has been properly mowed is simple, you are suspicious about the computation and would like to receive a confirmation that it was performed correctly. Ideally such a proof would be a short certificate attached to the result of the program. Hence we are interested in proofs or arguments[1] that are either non-interactive or two-message.

The problem of constructing short two-message arguments for various languages where the verifier is very efficient (e.g. linear in the input size, or even sublinear if it is coded properly) has received quite a lot of attention over the last twenty years (see Section 1.1). Suppose that we have a low communication public coins *interactive* (multi-round) protocol for proving (or arguing) membership in the language. A possible approach for obtaining short arguments is using a "compiler" that takes any protocol consisting of several rounds and removes the need for interaction, producing a two-message argument system.

One approach to constructing such a compiler is having the verifier encrypt and send all of its (public coin) challenges, where encryption is performed using a very malleable[2] scheme, such as Private Information Retrieval (PIR) or Fully Homomorphic Encryption (FHE)[3]. The prover uses the malleability to simulate his actions had the queries been given in plaintext, generating and sending back the appropriate ciphertexts (which should correspond to encryption of the answers the prover would give in the plain protocol). This compiler was studied by Kalai and Raz [KR09], as well as Kalai, Raz, and Rothblum [KRR14].

We investigate whether this compiler can be proved secure under *standard* cryptographic assumptions. That is, we do not want to base security on assumptions such as "having access to random oracles" or on so-called "knowledge assumptions", i.e. that one can extract from any machine that computes a certain function a value that is *seemingly* essential to that computation. Furthermore, we prefer not to rely on "super-polynomial hardness assumptions", i.e. that a certain cryptographic primitive is so hard it remains secure even if given enough time to break another primitive. In particular, such assumptions are not *falsifiable* in the sense of Naor [Nao03],[4] and they assume a strict hierarchy beyond $P \neq NP$. Also we want the underlying assumptions to be simple to state such as "*Learning With Errors is hard*". We prove positive and negative results about the above compiler:

- Assume FHE or PIR exist. Then there exists a sound interactive proof protocol, and there exists an FHE or PIR (respectively) scheme E, such that when the compiler is applied to the proof system using E, *the resulting two-message argument is insecure*. In fact, the compiler (when applied to this protocol) is insecure using all known FHE schemes. See Theorem 4.2.

[1] An argument is a "proof" that is sound (under cryptographic assumptions) so long as its creator is computationally bounded.

[2] Malleable in the cryptographic sense means that it is possible to manipulate a given ciphertext to generate related ciphertexts.

[3] As we shall see, the latter is needed if the prover's messages in the multi-round protocol depend on super-logarithmically-many bits sent by the verifier.

[4] A "falsifiable" cryptographic assumption is one that can be refuted efficiently. Falsifiability is a basic "litmus test" for cryptographic assumptions.

- For any FHE or PIR, if the verifier in the original protocol is log-space and uses only public coins, then the compiled argument is sound (See Theorem 5.3). Combining this with the work of Goldwasser et al. [GKR15], we obtain a succinct two-message argument system for any language in NC, where the verifier's work is linear (or even polylogarithmic if the input is coded appropriately). See Theorem 5.5. This is the first succinct two-message argument based on standard polynomial-time hardness assumptions.

1.1 Background

Obtaining succinct (e.g. sub-linear) two-message proof (or argument) systems has been a long-standing goal in the literature. Two primary approaches have been explored, both using cryptography to transform information-theoretic proof systems (interactive proofs, PCPs, or multi-prover interactive proofs) into non-interactive or two-message computationally-sound arguments.

Two-message arguments from PIR or FHE. The compiler studied in this work is rooted in a tantalizing suggestion of Aiello et al. [ABOR00a] in 2000, who proposed combining two powerful tools: The PCP (probabilistically checkable proofs) Theorem[5] and Computational PIR schemes[6] in order to obtain a succinct two-message argument system. In particular, leveraging the full strength of the PCP theorem, one could hope to obtain such arguments for all of NP. However, shortly thereafter Dwork et al. [DLN$^+$] pointed out problems in the proof and showed various counter examples for techniques of proving such a statement (see [ABOR00b]). No direct refutation was shown.[7]

Kalai and Raz [KR09] modified the Aiello et al. method, and suggested using it as a general compiler for turning public-coin *interactive proofs* (rather than PCPs) into two argument systems (without increasing the communication significantly, see below). They showed that, for any interactive proof system, one can tailor the compiler to that proof system by taking a large enough security parameter (polynomial in the communication of the proof system), and obtain a secure two-message argument. This requires subexponential hardness assumptions about the PIR or FHE. Applying the compiler to the interactive proofs of Goldwasser, Kalai and Rothblum [GKR08, GKR15] (see below), they obtain two-message arguments for bounded-depth computations.

Kalai, Raz and Rothblum [KRR14] study *no-signalling* proof systems, a restricted type of multi-prover interactive proof. They showed that, fixing any no-signalling proof, the compiler can also be tailored to that proof system, again giving a secure two-message argument (and also using sub-exponential hardness assumptions). Since no-signalling proof systems are more powerful than interactive proofs, they obtain two-message arguments for a larger class of computations (going from bounded-depth in [KR09] to P in [KRR14]).

Kilian, Micali, et sequelae. In 1992 Kilian [Kil92] suggested a short argument system for any language in NP. The protocol required 4 messages and the total amount of bits sent was polylog(n),

[5] That states that for every language $L \in \mathsf{NP}$ there exist a polynomial size witness/proof that may be verified, with constant error probability, by probing only a constant number of locations of the proof.

[6] Enabling a two-party protocol where one party holds a long string S and the other party is interested the value of the string at a particular location i; the second party does not want to reveal i and the goal is to have a low communication (much shorter than S) protocol; See Section 2.

[7] The original Aiello et al. [ABOR00a] protocol had an additional oversight, having to do with verifying the consistency of query answers. As Dwork et al. [DLN$^+$] showed, this can be corrected using a probabilistic consistency check.

where n is the input length, times a security parameter. The cryptographic assumption needed was fairly conservative, namely the existence of collision-resistant hash functions. Provided the prover has a witness, the work done by the prover is polynomial in the instance plus the witness sizes. In this protocol, the prover first committed to a PCP proof using a hash function provided by the verifier via a Merkle Tree (first two messages). The verifier then issued queries to the PCP and the prover needed to open its commitment in the specified locations in a way that would make the PCP verifier accept[8] (requiring two additional messages).

Some time later, Micali [Mic00] suggested using the Fiat-Shamir methodology [FS86] of removing interaction from public-coin protocols using an idealized hash function (random oracle) to obtain a two-message (or even non-interactive) succinct argument for any language in NP. Micali's work raised the issue of whether it is possible to obtain such argument systems in the "real world" (rather than in an idealized model). Barak and Goldreich [BG08] showed that security for the 4-message protocol could be based on standard (polynomial-time) hardness assumptions, but no secure instantiation of non-interactive arguments for NP is known under standard cryptographic assumptions.

Negative results and perspective. On the negative side, Gentry and Wichs [GW11] have shown that constructing two-message adaptively sound arguments for NP is going to be tricky: take any short two-message (even designated-verifier) proof system for NP, and assume that there are exponentially hard one-way functions. Then, paradoxically, any black-box reduction from a cheating prover to a falsifiable assumption can actually be used to break the assumption. One can interpret this result in several ways: (*i*) We need to find non black-box techniques in this realm. (*ii*) We should explore the boundaries of the Gentry-Wichs proof, i.e. when can we obtain black-box reductions and in particular what happens to computation in P (as opposed to NP). (*iii*) Use a non-falsifiable assumption. We prefer the first two interpretations, but there are quite a few works taking approach (*iii*). Thus, Kalai and Raz [KR09] and Kalai, Raz and Rothblum [KRR14] used super-polynomial hardness assumptions and obtained two-message succinct protocols for all languages computable in bounded depth and in P (respectively). Several works, Mie [Mie08], Groth [Gro10], Bitansky et al. [BCCT12] and Goldwasser et al. [GLR11] used a knowledge assumption (where one assumes that in order to perform a certain computation another piece of information is necessary and extractable).

Proofs for Muggles. Goldwasser, Kalai and Rothblum [GKR08, GKR15] were able to obtain a succinct interactive *proof* system (with many rounds) for any language that can be computed using small-depth circuits of polynomial size (NC, or, more generally, bounded-depth circuits). The prover in their system runs in polynomial time. The verifier runs in nearly-linear time and logarithmic space, and uses only public-coin. The communication and round complexities are related to the circuit depth (using bounded fan-in circuits).

Other related works. Paneth and Rothblum [PR14] construct non-interactive arguments in a common reference string model for any computation in P. Their constructions are based on efficiently falsifiable assumptions over multilinear maps. Candidates for multilinear maps have been suggested recently, starting with the work of Garg, Gentry and Halevi [GGH13a], but the

[8] This is not a precise representation of Kilian's work, for instance the PCP Theorem did not exist in its 'final' form when he proved his result.

security of these objects is not yet well understood, and is an active area of research. Looking ahead, we note that our construction of two-message arguments for bounded-depth computations is currently the only other construction based on efficiently falsifiable assumptions; we assume only PIR or FHE, rather than assumptions over multilinear maps. In a different vein, Bitansky et al. [BGL$^+$15] construct non-interactive arguments using Indistinguishability Obfuscation (IO). This can be instantiated using the candidate of Garg et al. [GGH$^+$13b] (which itself builds on multilinear maps).

Gennaro, Gentry and Parno [GGP10] have suggested a combination of garbled circuits and FHE in order to obtain non-interactive verification of outsourced work. In their setting a long setup message is sent by the verifier (whose length is proportional to the *total* amount of work) and for each subsequent input the verifier only needs to send a message proportional in length to the input size. The prover sends a short message and verification is quick.

1.2 Our Results

We investigate the compiler for converting public-coin interactive protocols into two-message protocols and show positive and negative results. On the positive side, we show that if the verifier uses only public coins and logarithmic space (and in particular, it has no secret memory), then *the compiler is secure*. This result can then be used to show that *any language in* NC *has a succinct two-message protocol based on any FHE*. More generally, if the computation involves a circuit of depth $D(n)$, then it can be proven by sending a message whose length is polynomial in $D(n)$ times the length of FHE ciphertexts. This is because not only does NC have log-space public-coin interactive proofs [FL93], but these can be made succinct, and moreover, such interactive proofs exist for any bounded-depth computation [GKR15]. These results are described in Section 5.

An application of the positive results could be for cases where exhaustive search is involved, and the entity performing the search wishes to show that it was unsuccessful or that the given result is the best possible. (A recent instance of such cases occurs in pools for mining Bitcoins: the goal is to search for a "nonce" that when added to the current block and hashed yields a certain number of ending 0's). Such an entity (in the Bitcoin case, the participant in the pool) can provide a two-message argument that the computation was properly performed but alas, the search was not successful; the length of the argument is poly-logarithmic in the space searched (in case of Bitcoin the argument length would be polynomial in the length of the nonce). See details in Section 5.3.

On the negative side, we show that if FHE schemes exist, then there exists a simple three-message interactive proof (i.e. with unconditional soundness) that, when compiled, yields an unsound argument. In particular, this example means that to instantiate the compiler one must consider the protocol compiled and take into account the communication and runtimes of the parties. This is described in Section 4.

The Compiler is described in detail in Section 3 and general definitions are given in Section 2.

2 Definitions and basic properties

A function $\mu\colon \mathbb{N} \to [0,1]$ is *negligible*, denoted by $\mu = negl(n)$, if for every polynomial p, there exists $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$, $\mu(n) \leq \frac{1}{p(n)}$.

2.1 Interactive Protocols

In this work, an interactive protocol consists of a pair $(\mathcal{P}, \mathcal{V})$ of interactive Turing machines that are run on a common input x, whose length we denote by $n = |x|$. The first machine is called *the prover* and is denoted by $\mathcal{P}$, and the second machine, which is probabilistic, is called *the verifier* and is denoted by $\mathcal{V}$. At the end of the protocol, the verifier accepts or rejects (this is the protocol's output).

Public-coin Protocols. An interactive protocol is *public coins* if each bit sent from the verifier to the prover is uniformly random and independent of the rest of the communication transcript.

Definition 2.1 (Interactive Proof [GMR89])**.** *An interactive protocol $(\mathcal{P}, \mathcal{V})$ (as above) is an* Interactive Proof *for a language L if it satisfies the following two properties:*

- **completeness:** *For every $x \in L$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on common input x, then $\mathcal{V}$ accepts with probability* 1.[9]
- **s_{IP}-soundness:** *For every $x \notin L$ and every (computationally unbounded) cheating prover strategy $\mathcal{P}^*$, the probability that the verifier $\mathcal{V}$ accepts when interacting with $\mathcal{P}^*$ is at most $s_{IP} = s_{IP}(n)$, where s_{IP} is called the*soundness error *of the proof-system. The probability is over the verifier's coin tosses.*

A verifier is log-space and public-coin *if it is public coin (as above), and uses only a $O(\log n)$-size memory tape (on top of one-way access to the communication and randomness tapes).*

Definition 2.2 (λ-History-Aware Interactive Proof)**.** *An interactive proof is $\lambda = \lambda(n)$-history-aware if on top of the requirements of Definition 2.1, it is also the case that each message sent by the (honest) prover P is only a function of the last λ bits sent by the verifier.*

Note that in the above definition we make no assumptions on the strategies that can be employed by cheating provers. Note also that we do not use the related "history ignorant" terminology of [KR09], as we prefer the convenience of the "history-aware" definition.

We add explicit timing and probability parameters to the usual definition of argument systems.

Definition 2.3 (Argument System)**.** *An interactive protocol $(\mathcal{P}, \mathcal{V})$ (as above) is an* Argument System *for L if it is complete, as per Definition 2.1, and satisfies computational soundness:*

- **s_{arg}-soundness against T_{arg}-time cheating provers:** *For every $x \notin L$ and every cheating prover $\mathcal{P}^*$ whose strategy can be implemented by a $T_{arg} = T_{arg}(n)$-time Turing Machine, the probability that the verifier $\mathcal{V}$ accepts when interacting with $\mathcal{P}^*$ is at most $s_{arg} = s_{arg}(n)$. The probability is over the verifier's coin tosses.*

2.2 FHE

Both FHE and PIR schemes allow one party to send to another party a relatively short string that is an encryption of a query. The second party then computes a ciphertext of a message that is supposed to be a function of the original message and information that the second party possesses. In the case of FHE, the query is a vector y in (say) $\{0,1\}^m$, the second party possesses a function $f\colon \{0,1\}^m \to \{0,1\}$, and the answer-ciphertext is an encryption of $f(y)$.

[9] More generally, there could be a small completeness error.

Definition 2.4 (Fully Homomorphic Encryption)**.** *An FHE scheme is defined by algorithms:* $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$*, who all get as part of their input the security parameter* 1^κ *and an input length parameter* m *(we omit these two inputs when they are clear from the context). The* KeyGen *algorithm outputs a pair of public and secret keys* (pk, sk)*. The encryption algorithm* Enc *takes the public key and a vector* $y \in \{0,1\}^m$*, and outputs an encryption of* y*. The* Eval *algorithm takes as input the public key* pk*, an encryption of* y *(under* pk*) and a function* $f\colon \{0,1\}^m \to \{0,1\}$*, and outputs an encryption of* $f(y)$*. Finally, the decryption algorithm takes as input the secret key* sk *and the encryption of* $f(y)$ *produced by* Eval *and outputs the plaintext* $f(y)$*. We require:*

- **Completeness:** $\forall \kappa, m \in \mathcal{N}, y \in \{0,1\}^m$*, for any function* f *of circuit-size* $\mathrm{poly}(m)$ *and* (pk, sk) *generated by* KeyGen*, we have:*

$$\mathsf{Dec}(sk, \mathsf{Eval}(pk, f, \mathsf{Enc}(pk, y))) = f(y).$$

- **Semantic Security:** $\forall \kappa, m \in \mathcal{N}, y, y' \in \{0,1\}^m$*, the distributions* $\mathsf{Enc}(pk, y)$ *and* $\mathsf{Enc}(pk, y')$ *(where* pk *is generated by* KeyGen*) are* $negl(\kappa)$*-indistinguishable.*

- **Complexity:** *The algorithm* KeyGen *runs in time* $\mathrm{poly}(\kappa)$*. The algorithms* $\mathsf{Enc}, \mathsf{Dec}$ *run in time* $\mathrm{poly}(\kappa, m)$*. The algorithm* Eval *runs in time* $\mathrm{poly}(\kappa, m, |f|)$*. The outputs of* Enc *and* Eval *are of length* $\mathrm{poly}(\kappa, m)$*.*

The possible existence of FHE scheme was an open question for many years until Gentry's work [Gen09] and we know now that FHE schemes can be constructed under standard lattice assumptions such as LWE [BV14].

2.3 PIR

Here the query is an index $y \in [\lambda]$, the second party possesses a database $Z \in \{0,1\}^\lambda$, and the answer-ciphertext is an encryption of Z_y.

Definition 2.5 (Private Information Retrieval (PIR) Scheme)**.** *A PIR scheme is defined by three algorithms:* $\mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$*, who all get as part of their input the security parameter* 1^κ *and database length* λ *(we omit these two inputs when they are clear from the context).* Enc *also takes as input an index* $y \in [\lambda]$*, and outputs an "encryption"* c *of* y*, and a "secret key"* sk *for decryption. The* Eval *algorithm takes as input an encryption of* y *and a database* $Z \in \{0,1\}^\lambda$*, and outputs an "encryption" of* Z_y *(the* y*-th bit of* Z*). Finally,* Dec *takes as input the secret key* sk *and a ciphertext generated by* Eval *and outputs* Z_y*. We make the following requirements:*

- **Completeness:** $\forall \kappa, \lambda \in \mathcal{N}, y \in [\lambda], Z \in \{0,1\}^\lambda$*, and* $(c, sk) \leftarrow \mathsf{Enc}(y)$ *we have that:*

$$\mathsf{Dec}(sk, \mathsf{Eval}(Z, c)) = Z_y.$$

- **Semantic Security:** $\forall \kappa, \lambda \in \mathcal{N}, y, y' \in [\lambda]$*, taking* $(c, sk) \leftarrow \mathsf{Enc}(y)$ *and* $(c', sk') \leftarrow \mathsf{Enc}(y')$*, the distributions of* c *and of* c' *are* $negl(\kappa)$*-indistinguishable.*

- **Complexity:** *The algorithms* $\mathsf{Enc}, \mathsf{Dec}$ *run in time* $\mathrm{poly}(\kappa, \log \lambda)$*. The algorithm* Eval *runs in time* $\mathrm{poly}(\kappa, \lambda)$*. In particular, the outputs of* Enc *and* Eval *are of length* $\mathrm{poly}(\kappa, \log \lambda)$*.*

PIR Schemes exist under a variety of assumptions such as quadratic residuosity [KO97] , Φ-hiding [CMS99] and Learning with Errors [BV14].

3 Detailed Description of The Compiler

3.1 The Compiler: FHE Variant

We now describe the compiler in detail, focusing first on the FHE variant and in Section 3.2 the PIR variant. The compiler starts with a many-round public-coin interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$, and produces a two-message argument system $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$. It is based on that of Kalai and Raz [KR09]. However, we leave the security parameter free, rather than tailoring it to the Interactive Proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ to be compiled.[10]

We denote by $\mathsf{Enc}_{pk}(y)$ the encryption of $y \in \{0,1\}^m$ under public key pk. Note that this is really a *distribution* on ciphertexts. Assume that $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ consists of k rounds (and $2k$ messages). For each round i, in which the verifier should send a random value α_i, the (compiled) verifier chooses an independent key pk_i from the underlying encryption system. In a single message, it sends the public keys $\{pk_i\}_{i=1}^k$ and the ciphertexts $\{a_i = \mathsf{Enc}_{pk_i}(\alpha_1, \alpha_2, \ldots \alpha_i)\}_{i=1}^k$. That is, the ciphertext $a_i = \mathsf{Enc}_{pk_i}(\alpha_1, \alpha_2, \ldots \alpha_i)$ is the encryption of the messages sent in rounds $1, \ldots, i$ of the simulated protocol. The prover uses the ciphertext a_i to homomorphically compute an encryption of the answer that it would have sent given the queries $\alpha_1, \alpha_2, \ldots \alpha_i$, i.e., what it "would have done" at round i.

Let the (efficiently computable) function that computes the i-th prover message in the interactive protocol be $\mathcal{P}_i(\alpha_1, \alpha_2, \ldots \alpha_i)$. So the prover computes and sends $b_i = \mathsf{Enc}_{pk_i}(\mathcal{P}_i(\alpha_1, \alpha_2, \ldots \alpha_i))$. This is done simultaneously for all the rounds. The verifier then decrypts the messages it receives, where β_i is the decryption of the ciphertext b_i, and accepts if and only if the simulated verifier accepts the transcript $(\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k)$.

The resulting protocol is given in Figure 1. By construction, it is a two-message protocol. Completeness rests on the completeness of the FHE scheme, i.e. if the scheme is complete, then so is the resulting protocol. The communication complexity of the new protocol increases: the verifier sends $k^2/2$ bit-encryptions and k public keys. The prover responds with k ciphertexts. Letting γ bound the length of ciphertexts and public keys, the total communication complexity is $O(k^2 \cdot \gamma)$. *The soundness of the resulting protocol is the main issue addressed in this work.*

Historical Note: Multi-Prover Proof Systems A related compiler starts with a multi-prover two-message scheme instead of a single-prover protocol (this is closer to the original idea of [ABOR00a]). As in the above compiler, the idea is for the verifier to encrypt the queries using independent keys and then ask that the prover perform the computation it would have done to answer the queries in the original protocol.

3.2 The Compiler: PIR Variant

The PIR-based variant of the compiler is given in Figure 2. We assume that the interactive proof to be compiled $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is only λ-history-aware, so the prover's i-th message only depends on the last λ bits sent by the verifier.

The verifier $\mathcal{V}_{arg}$ simulates the interactive verifier $\mathcal{V}_{IP}$ to generate its k messages $\alpha_1, \ldots, \alpha_k \in \{0,1\}$. For each $i \in [k]$, $\mathcal{V}_{arg}$ uses the PIR scheme to compute: $(c_i, sk_i) \leftarrow \mathsf{Enc}(\alpha_{i-\lambda+1}, \ldots, \alpha_i)$, where we interpret α_j as α_1 if $j < 1$ (this will occur when $i < \lambda$). For each $i \in [k]$, the prover $\mathcal{P}_{arg}$ interprets c_i, which encrypts a string of $\lambda' \leq \lambda$ bits, as a PIR query into a database of size at most

[10] The compiler can be based on FHE or PIR, see Section 3.2 for the PIR-based variant.

Protocol $(\mathcal{P}_{arg}, \mathcal{V}_{arg})(x, 1^\kappa)$ for Language L

The compiler uses an FHE scheme, with security parameter κ. Without loss of generality, we assume that each message in $\Pi = (\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is only a single bit.

$\mathcal{V}_{arg} \to \mathcal{P}_{arg}$: The verifier $\mathcal{V}_{arg}$ simulates the interactive verifier $\mathcal{V}_{IP}$ to generate its k challenges $\alpha_1, \ldots, \alpha_k \in \{0,1\}$. For each $i \in [k]$, $\mathcal{V}_{arg}$ chooses keys (pk_i, sk_i) for the PIR or FHE.

$\mathcal{V}_{arg}$ then sends the keys and ciphertexts $\{pk_i, \mathsf{Enc}_{pk_i}(\alpha_1, \ldots, \alpha_i)\}_{i\in[k]}$ (as a single message).

$\mathcal{P}_{arg} \to \mathcal{V}_{arg}$: The prover $\mathcal{P}_{arg}$ simulates the interactive prover $\mathcal{P}_{IP}$, where the function $\mathcal{P}_i$ that computes $\mathcal{P}_{IP}$'s i-th message is applied to the encrypted challenges sent under pk_i. I.e., $\mathcal{P}_{arg}$ homomorphically computes $b_i = \mathsf{Enc}_{pk_i}(\mathcal{P}_i(\alpha_1, \ldots, \alpha_i))$.

$\mathcal{P}_{arg}$ then sends th ciphertexts $\{b_i\}_{i\in[k]}$ to $\mathcal{V}_{arg}$ (as a single message)

Verification: The verifier $\mathcal{V}_{arg}$ decrypts each ciphertext b_i to retrieve the message β_i. It accepts if and only if the interactive verifier $\mathcal{V}_{IP}$ accepts the transcript $(\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k)$.

Figure 1: Compiler (FHE variant): Compiling k-round Interactive Proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ to 2-msg argument $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$. Based on [KR09].

$2^{\lambda'}$. For each i, $\mathcal{P}_{arg}$ computes a database $Z^{(i)}$ containing all $2^{\lambda'}$ answers, one for each possible λ'-bit history that $\mathcal{P}_{IP}$ might have encountered in its i-th round in the underlying interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$. $\mathcal{P}_{arg}$ responds with $b_i \leftarrow \mathsf{Eval}(Z^{(i)}, c_i)$, which contains the answer to the λ'-bit history encrypted in the i-th verifier query.

In the compiled protocol, the *honest* prover $\mathcal{P}_{arg}$ runs in time 2^λ. When λ is at most logarithmic in the input length, the running time of $\mathcal{P}_{arg}$ remains polynomial. Indeed, in our positive result we apply the compiler to the interactive proof of [GKR15], which has a logarithmic λ (see Section 5). The communication complexity of the new protocol is as follows: the verifier sends k PIR queries into a database of size 2^λ, and the prover responds with k answers to the PIR queries. Letting γ be the communication complexity of the PIR scheme (the combined length of the PIR query and response for databases of size 2^λ), the total communication complexity is $k\gamma$. Note that (for logarithmic λ) this is an improvement over the $O(k^2\gamma)$ obtained using FHE.

4 The Negative Result: A Protocol that does not Compile Well

We now present a protocol $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ that does not compile well under the Compiler of Section 3. We work with the FHE variant of the compiler. The results hold *mutatis mutandis* for the PIR variant (see Remark 4.1 below).

This is what we would like to be able to say: "For any possible instantiation of the compiler with an FHE, there exists an (unconditionally sound) interactive protocol that, when compiled, yields an unsound two-message argument system." What we can actually say is: "For any possible instantiation of the compiler with an FHE, there exist another instantiation of the compiler with a (different) FHE and an interactive protocol that, when compiled, yields an unsound two-message

Protocol $(\mathcal{P}_{arg}, \mathcal{V}_{arg})(x, 1^\kappa)$ for Language L

The compiler uses a PIR scheme with security parameter κ. Without loss of generality, we assume that each message in $\Pi = (\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is only a single bit.

$\mathcal{V}_{arg} \to \mathcal{P}_{arg}$: The verifier $\mathcal{V}_{arg}$ simulates the interactive verifier $\mathcal{V}_{IP}$ to generate its k messages $\alpha_1, \ldots, \alpha_k \in \{0,1\}$. For each $i \in [k]$, $\mathcal{V}_{arg}$ uses the PIR scheme to compute:

$$(c_i, sk_i) \leftarrow \mathsf{Enc}(\alpha_{i-\lambda+1}, \ldots, \alpha_i).$$

$\mathcal{V}_{arg}$ then sends the PIR queries $\{c_i\}_{i\in[k]}$ (as a single message).

$\mathcal{P}_{arg} \to \mathcal{V}_{arg}$: For each $i \in [k]$, the prover $\mathcal{P}_{arg}$ interprets c_i, which encrypts a λ-bit string, as a PIR query into a database of size 2^λ. For each i, $\mathcal{P}_{arg}$ computes a database $Z^{(i)}$ containing all 2^λ answers, one for each possible λ-bit history that $\mathcal{P}_{IP}$ might have encountered in its i-th round in the underlying interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$.

For each $i \in [k]$, $\mathcal{P}_{arg}$ computes $b_i \leftarrow \mathsf{Eval}(Z^{(i)}, c_i)$, and sends the strings $\{b_i\}_{i\in[k]}$ to $\mathcal{V}_{arg}$ (as a single message).

Verification: The verifier $\mathcal{V}_{arg}$ decrypts each value b_i using the key sk_i and retrieves a message β_i. It accepts if and only if the interactive verifier $\mathcal{V}_{IP}$ accepts the transcript $(\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k)$.

Figure 2: Compiler (PIR variant). Compiling k-round λ-history-aware interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ to 2-message argument $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$

argument". That is, given the FHE we will need to modify it a bit (still getting an FHE), so that the compiler will fail. We suggest two alternate modifications to the underlying FHE to undermine the compiler. We stress that the compiler fails under all known implementations of FHE (without any modification).

The rough idea of the interactive protocol $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is for the prover to commit to a string $x_p \in \{0,1\}^n$ in the first round. The verifier then sends a "guess" for this commitment string in the form of $x_v \in \{0,1\}^n$ chosen uniformly at random. Finally the prover opens his commitment to the string x_p. The prover wins if the opening of x_p is legitimate (i.e. accepted by the receiver in the commitment protocol) and $x_v = x_p$. Obviously, since x_v is chosen after x_p, if the commitment is perfect (there is only one way to open any commitment), then the probability that the prover succeeds is $1/2^n$. This as an interactive proof protocol for the empty language.

Perfect and weak commitments. The protocol $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ uses a public-key encryption scheme to commit to the string x_p. To commit, the prover sends a public key pk and an encryption of x_p. To open the commitment, he sends the randomness used in the encryption. The resulting protocol is sound so long as the encryption scheme is *committing*: for every public key, and every ciphertext, there is only one message-randomness pair that yields that ciphertext.

We cannot base the above protocol on "non-committing" encryption, where some ciphertexts can be opened to *all* possible values, as is the case in deniable encryption [CDNO97, SW14]. If we

use such an encryption scheme in the above protocol, then the resulting protocol will not be sound (the prover can win). Simply put, the prover can open the encryption as the string that the verifier sent.

Nevertheless, we can relax the commitment property a bit. Instead of a perfect commitment, we can use a *weak commitment* scheme, where we modify the requirement that there is a unique opening, to one where there are few openings. If there are at most w different values that can be opened, then the probability of the prover winning the above game (guessing at most w strings that include the one chosen by the verifier) is at most $w/2^n$. We can then use any semantically secure public-key encryption scheme (even a non-committing one) to get a weak commitment as follows. The commitment is as above (i.e. consists of a public key and a ciphertext). To open the commitment, the committer sends a decryption key sk corresponding to pk. Assuming the decryption algorithm is deterministic (which is w.l.o.g, since we can fix the coins), there is a unique plaintext corresponding to the ciphertext given a candidate for sk.

For this to make sense we need to make sure that the length of the decryption key is much shorter than $n = |x|$, and we get a weak commitment as above (the number of possible openings is bounded from above by the number of decryption keys, $w = 2^{|sk|}$, much smaller than $2^{|x|}$).

4.1 The Protocol $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$

$(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is an interactive proof for the empty language, i.e. the verifier should reject any input w.h.p. The proposed protocol consists of 4 messages and uses public coins, where the first message, sent by the verifier, is empty. We can base it on any committing encryption scheme as above (and in parenthesis describe how to deal with non committing encryption). The notation $\mathsf{Enc}_{pk}(x, r)$ indicates that the message x is encrypted under public key pk using randomness r.
The protocol $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ is:

1. $\mathcal{V} \mapsto \mathcal{P}$: empty message.
2. $\mathcal{P} \mapsto \mathcal{V}$: The prover uses a committing encryption scheme. It picks public key pk_p, string $x_p \in \{0,1\}^n$, randomness r_p and sends $(pk_p, c_p = \mathsf{Enc}_{pk_p}(x_p, r_p))$.

   (same is done in the non-committing case.)
3. $\mathcal{V} \mapsto \mathcal{P}$: the verifier picks random $x_v \in \{0,1\}^n$ and sends it.
4. $\mathcal{P} \mapsto \mathcal{V}$: the prover sends $m = (x_p, r_p)$.

Verification. The verifier checks whether $x_p = x_v$ and $c_p = \mathsf{Enc}_{pk_p}(x_p, r_p)$ and accepts if they are both satisfied. Note that to perform this check, the verifier needs to "remember" pk_p, c_p and x_v (we refer to this fact in Section 5).

In the non-committing encryption variant of the protocol, in Step 4 the prover sends the decryption key sk_p corresponding to pk_p. In the verification step, the verifier decrypts c_p using sk_p and accepts if the answer equals x_v.

Theorem 4.1. *For any perfectly committing encryption scheme (respectively, non-committing scheme), the above protocol is complete and sound, with soundness error at most* $1/2^n$ *(respectively,* $2^{|sk|}/2^n$ *for the non-commiting variant).*

4.2 The compiled protocol

Let $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$, described next, be the argument system obtained by applying the compiler of Figure 1 to the interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ for the empty language, described in Section 4.1.

$\mathcal{V}_{arg} \mapsto \mathcal{P}_{arg}$: Verifier picks and sends $pk_{v,1}$ (for the first round's empty message), and $pk_{v,2}, c_v = \mathsf{Enc}_{pk_{v,2}}(x_v, r_v)$ (for the verifier's second message in $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$).

$\mathcal{P}_{arg} \mapsto \mathcal{V}_{arg}$: The (honest) prover $\mathcal{P}_{arg}$ sends $\mathsf{Enc}_{pk_{v,1}}((pk_p, c_p = \mathsf{Enc}_{pk_p}(x_p, r_p)), r')$ (for the first prover message), and $\mathsf{Enc}_{pk_{v,2}}(m = (x_p, r_p), r'')$ (for the second message).

The verifier decrypts the first prover message using $sk_{v,1}$ to retrieve (pk_p, c_p), decrypts the second message using $sk_{v,2}$ to retrieve $m = (x_p, r_p)$, and accepts if the original protocol's verifier accepts.

Speaking intuitively, the compiler will fail because a cheating prover can use the encryption c_v of the message x_v that the compiled verifier sends him to come up with a commitment to $x_p = x_v$. The challenge to the cheating prover then is how to obtain an encryption of the randomness r_p sent in Step 3 of $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$. This seems like quite an obstacle for an arbitrary FHE scheme.

Suppose, however, that FHE scheme E "makes the cheating prover's life easy". That is, every encryption also includes an encryption of the randomness r (using freshly chosen randomness r_{add}) (the decryption algorithm simply ignores the second part of the ciphertext). The cheating prover for $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$ can use this encryption of the randomness r_p to break soundness. Any FHE can be tweaked in this way, without harming its homomorphic or security properties. The case of a non-committing encryption is handled similarly.

Breaking the compiled protocol. Given $pk_{v,1}, pk_{v,2}$ and $c_v = \mathsf{Enc}_{pk_{v,2}}(x_v, r_v)$, the cheating prover P^* sends $\mathsf{Enc}_{pk_{v,1}}(pk_{v,2}, c_v)$ as its first message, and c_v as its second message. Recall that $c_v = \mathsf{Enc}_{pk_{v,2}}(x_v, r_v)$, and this includes both an encryption of x_v and of r_v, since we assumed that the cryptosystem Enc is such that it also gives an encryption of the random string. Thus, by following this strategy, P^* makes $\mathcal{V}_{arg}$ accept with probability 1 (based on perfect completeness of the encryption scheme).

Alternatively, for non-committing encryption, we assume that the public key includes an encryption of the secret key. Thus, the public key $pk_{v,2}$ includes $\mathsf{Enc}_{pk_{v,2}}(sk_{v,2})$, which can be sent by the cheating prover P^* as its second message to "de-commit" and break security. Thus, we can use a "circular-secure" FHE scheme, where semantic security holds even when the public key includes an encryption of the secret key, to show that the compiler fails. We note that it is not known in general whether "circular security" holds, namely whether including an encryption of the secret key in the public key *always* preserves semantic security (see e.g. Rothblum [Rot13]). However, for known FHE schemes, an encryption of the secret key is already included in the public key to enable "bootstrapping" (see e.g. Gentry [Gen09]). Thus, *the compiler is insecure when instantiated with all known concrete FHE candidates.* Moreover, even if a new (and non-committing) FHE candidate is discovered, we can modify its public key to include an encryption of the secret key. If the modified scheme remains semantically secure, then (as above) the compiler fails on $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$. Thus, proving that the compiler is secure with the modified scheme would require proving that the original FHE scheme is *not* circular secure.

We can see that in both cases the cheating prover P^* succeeds and the verifier accepts. We therefor have:

Theorem 4.2. *If an FHE scheme* Enc *exists, then there exists an instantiation of the compiler of Figure 1 with a (possibly) modified FHE scheme and a sound protocol in the public coins model such that the resulting compiled protocol is not sound.*

Remark 4.1. *The same protocol "misbehaves" under the PIR-based compiler in Figure 2. The compiled protocol is unsound when the compiler uses a PIR scheme that is directly based on the same FHE: where ciphertexts are modified to include encryptions of the randomness (in the perfectly committing case), or of the secret key (in the weakly committing case). The PIR scheme operates by sending the (modified) encryption of the index being queried. The* Eval *algorithm responds with an answer-ciphertext, and the* Dec *algorithm simply decrypts this ciphertext using the FHE decryption.*

5 Positive Results

We show that the Compiler of Figure 1 is secure when applied to interactive proofs with a public-coin log-space verifier. More generally, the compiler is secure for interactive proofs where (for any fixed partial transcript) the *optimal* continuation strategy, i.e. the strategy that maximizes the verifier's success probability, can be computed in polynomial time. We only assume the existence of standard (polynomially hard) PIR or FHE. This result is in Theorem 5.3 below, which we prove using a careful analysis of the Kalai-Raz compiler [KR09]. Recall that the negative example of Section 4 shows that the compiler is *insecure* for general interactive proofs. In particular, recall that (as noted above) the verifier needs enough space to "remember" pk_p, c_p, x_v. Thus, we need to leverage additional structure in order to prove security, and we do so via the space complexity of the verifier (or the optimal-continuation strategy). Kalai and Raz, in contrast, showed that security could be obtained by making super-polynomial hardness assumptions and simultaneously tailoring the compiler's security parameter to a given interactive proof system (that is, they choose the security parameter after seeing the interactive proof to be compiled).

Recall that for any language computable by depth $D(n)$ (log-space uniform) circuits, there exists an interactive proof where the verifier uses logarithmic space and public coins [GKR15]. By applying the compiler to these interactive proofs, we obtain a succinct two-message argument using any (polynomially-hard) PIR scheme. The communication is $\tilde{O}(D(n) \cdot \gamma)$, where γ is the communication required by the PIR scheme (for a database of length $\text{poly}(n)$). Similarly to [KR09], we use the fact that every prover message in the succinct interactive proof only depends on a logarithmic number of bits sent by the verifier. This result is in Theorem 5.5. In Section 5.3 we discuss applications to proving the results of an exhaustive search.

5.1 Security of the Compiler

As described above, our main insight is that if there is a polynomial time algorithm that computes an optimal prover-strategy for any interactive proof, then we can compile the protocol with no significant soundness error. For a log-space public-coin verifier this is possible and arguments of this type were used by Condon [Con91] and Fortnow and Sipser (see [For89]) to show that these proof systems can only compute languages in P. In fact, for any fixed partial transcript between the prover and verifier, we show how to *efficiently* compute an optimal prover strategy for continuing the interaction. The main cause for the super-polynomial security gap in the Kalai-Raz security reduction was the running time required to compute an optimal prover strategy (for a

fixed partial transcript). For general interactive proofs, this requires time that is exponential in the communication. We leverage the *polynomial-time* transcript-completion algorithm to obtain a tighter security reduction and our main result.

We proceed by first defining the optimal transcript completion task for an interactive proof. We then show that (*i*) for log-space public-coin interactive proofs, there is an *efficient* transcript completion algorithm (Theorem 5.2), and (*ii*) we can strengthen the Kalai-Raz security reduction from the security of the argument to the security of the PIR scheme to reduce the loss in security: rather than *exponential* in the communication complexity of the interactive proof, as in [KR09], it is *polynomial* in the time required for optimal transcript completion and in the security parameter (Theorem 5.3).

Definition 5.1 (Optimal Transcript Completion)**.** *We say that a public-coin interactive proof* $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ *with communication complexity* ℓ_{IP} *supports* optimal transcript completion *in time* $T_{IP}(n)$*, if there exists an algorithm* $\mathcal{P}'$*, running in time* $T_{IP}(n)$*, that, on input* any *partial transcript of communication with the verifier* $\mathcal{V}_{IP}$ *and ending with a message sent by* $\mathcal{V}_{IP}$*, produces a next message from the prover to the verifier, satisfying the following guarantee: For any algorithm* $\mathcal{P}^*$ *(with unbounded running time), the probability that* $\mathcal{V}_{IP}$ *accepts when the transcript is completed using* $\mathcal{P}^*$ *is no greater than the probability that* $\mathcal{V}_{IP}$ *accepts when the transcript is completed using* $\mathcal{P}'$*.*

Remark 5.1. *We note that even if the interactive proof has an efficient (i.e.* poly(n)*-time) honest prover, there may not be an efficient optimal transcript completion algorithm* $\mathcal{P}'$*. Indeed, we can build explicit examples under (mild) cryptographic assumptions—e.g. using the Interactive Proof described in Section 4.*

We now show that any interactive proof with a log-space public-coin verifier supports (polynomial time) optimal transcript completion:

Theorem 5.2 (See also [Con91, For89])**.** *Let* $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ *be a log-space public-coin interactive proof. Then* $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ *supports optimal transcript completion in* poly(n) *time.*

Proof Sketch. The transcript completion algorithm $\mathcal{P}'$ constructs the verifier's directed layered state graph, where every node is a possible memory configuration for the verifier. We assume w.l.o.g. that the verifier keeps track of the current round, and that each message in the protocol is a single bit. Each round consists of a pair of messages: a single random bit sent from the verifier to the prover, and a single bit sent in response from the prover to the verifier. For round i and memory configuration u, and round $(i+1)$ and memory configuration v, there is an edge from (i, u) to $(i+1, v)$ iff starting with configuration u just after round i, the verifier can reach configuration v just after round $(i+1)$. Since each round consists of two single-bit messages, each node in the graph has at most 4 successors. This completes the description of the graph, and we note that it is of poly(n) size (because the verifier runs in $O(\log n)$ space).

Now, for any verifier state (i, u) we compute the optimal prover strategy as follows. We start from terminal nodes and work our way "backwards" in the state graph to nodes corresponding to states in earlier rounds. For each node/state, we compute the probability that the verifier accepts once it reaches that state, and the optimal prover response to the next verifier challenge. For node (i, u) we denote this by $a(i, u)$. For terminal nodes, the acceptance probability is either 0 or 1 (depending on whether this is a rejecting or accepting node). Once the accepting probabilities have been computed for all round-$(i+1)$-states we can compute the accepting probabilities (and best

prover responses) for round-i-states. For a non-terminal node (i, u), there are 4 possible transcripts for round i: $00, 01, 10, 11$, where transcript (α, β) leads to state $(i+1, w_{\alpha,\beta})$ (here $\alpha, \beta \in \{0,1\}$). For each verifier challenge α, the "best response" is the message leading to the state that maximizes the acceptance probability:

$$b(\alpha) = argmax_{\beta \in \{0,1\}}(a(i+1, w_{\alpha,\beta}))$$

and the (maximal) probability of acceptance is

$$a(i,u) = \frac{a(i+1, w_{0,b(0)}) + a(i+1, w_{1,b(1)})}{2}.$$

By construction, this procedure computes an optimal prover strategy for any verifier state.

Given a partial transcript, $\mathcal{P}'$ can compute the current verifier state and use this procedure to complete a best-response strategy. □

Next, we give a strengthened security analysis for the Kalai-Raz compiler, where the reduction is parameterized in terms of the time required for optimal transcript completion (see in comparison Lemma 4.2 of [KR09] where "brute-force" completion is used):

Theorem 5.3. *Let $\Pi = (\mathcal{P}_{IP}, \mathcal{V}_{IP})$ be a public-coin interactive proof for a language L that is $\lambda(n)$-history-aware (as in Definition 2.2), with completeness $c_{IP}(n)$, soundness $s_{IP}(n)$ and communication complexity $\ell_{IP}(n)$. Let PIR be a PIR scheme with communication $\ell_{PIR}(m, \kappa)$.*

Then $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$, the argument system of Figure 2 instantiated with $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ and PIR, is a 2-message argument system for L, with completeness $c_{arg} = c_{IP}$, communication complexity $\ell_{arg}(n, \kappa) = \ell_{IP}(n) \cdot \ell_{PIR}(2^{\lambda(n)}, \kappa)$, and the following properties:

1. **Computational Soundness:** *If the interactive proof supports optimal transcript completion, as in Definition 5.1, in time $T_{IP}(n)$, and the PIR system is secure against adversaries running in time $T_{PIR}(\kappa)$, then the argument system has soundness $s_{arg}(n) = (\ell_{IP}(n) \cdot (s_{IP}(n) + negl(\kappa)))$ against adversaries running in time $T_{arg} = T_{PIR}(\kappa)/\text{poly}(n, \ell_{IP}(n), T_{IP}(n))$.*
2. **Honest Prover Complexity.** *$\mathcal{P}_{arg}$ runs in time $\text{poly}(T_{IP}(n), \kappa, 2^{\lambda(n)})$.*

Security of the compiler using FHE. An analogous theorem statement holds for the FHE version of the compiler. The advantage over the PIR version is that there is no need to assume that the interactive proof is λ-history aware. The disadvantage (other than the stronger assumption) is the quadratic blowup in the communication complexity (see Section 3).

Proof of Theorem 5.3. We assume w.l.o.g. that the each message sent by the verifier is 1 bit long, and take k to be the number of rounds. Suppose that there exists an input $x^* \notin L$ and a cheating prover $\mathcal{P}^*_{arg}$ that manages to convince $\mathcal{V}_{arg}$ with probability ε.

Definition of p_i. For $i \in \{0, 1, \ldots, k\}$, define p_i to be the success probability of the following process called $\mathsf{Experiment}_i$: run the argument system with the cheating prover $\mathcal{P}^*_{arg}$. Let $\{(pk_i, sk_i)\}_{i \in [k]}$ be the keys and α_i be the bits encrypted in the challenges sent by $\mathcal{V}_{arg}$. Let $\{b_i\}_{i \in [k]}$ be the ciphertext answers returned by $\mathcal{P}^*_{arg}$, and let β_i be the plaintext value in b_i. Fixing the partial transcript $(\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)$ for the first i rounds, run the optimal-completion strategy $\mathcal{P}'_{IP}$ with

the verifier $\mathcal{V}_{IP}$ (who simply generates random messages) to complete the transcript (i.e. for the last $(k-i)$ rounds), generating messages $(\alpha'_{i+1}, \beta'_{i+1}, \ldots, \alpha'_k, \beta'_k)$. $\mathsf{Experiment}_i$ succeeds if and only if $\mathcal{V}_{IP}$ accepts the resulting transcript $(\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i, \alpha'_{i+1}, \beta'_{i+1}, \ldots, \alpha'_k, \beta'_k)$.

Claim 5.4. *There exists $i^* \in [k]$ s.t.:*

$$p_{i^*} - p_{i^*-1} \geq \frac{\varepsilon - s_{IP}}{k}$$

Proof. The proof is by a hybrid argument. p_0 is bounded by the success probability of a cheating prover in the (sound) interactive proof, and thus it is at most s_{IP} (since $x^* \notin L$). p_k is exactly the success probability of the cheating prover $\mathcal{P}^*_{arg}$ in the two-message argument system, and thus by assumption it is at least ε. □

Breaking the Encryption. We show that any $\mathcal{P}^*_{arg}$ that succeeds with probability at least ε, can be used to break semantic security of the encryption scheme with advantage at least $(\varepsilon - s_{IP})/k$. We do this by constructing a distinguisher for the following two distributions. In both distributions, generate keys $(pk_i, sk_i)_{i\in[k]}$, random bits $(\alpha_1, \ldots, \alpha_k)$, and the challenge ciphertexts $\{a_i\}_{i\in[k]}$. The distribution outputs all of the public keys and ciphertexts, the first $(i^* - 1)$ secret keys, and all plaintext values except the i^*-th, i.e. $\{\alpha_i\}_{i\neq i^*}$. So far the distributions are identical, the only difference is in a final output value α (see below). In particular, a sample from D_1 or D_2 is of the form:

$$\left(\{pk_i, a_i\}_{i\in[k]}, \{sk_i\}_{i<i^*}, (\alpha_1, \ldots, \alpha_{i^*-1}, \alpha, \alpha_{i^*+1}, \ldots, \alpha_k)\right),$$

where in D_1 we set $\alpha = \alpha_{i^*}$, and in D_2 we draw a uniformly random and independent bit α'_{i^*}, and set $\alpha = \alpha'_{i^*}$. By construction, if the distinguisher distinguishes D_1 and D_2 with non-negligible advantage, then semantic security is broken.

We now use the cheating prover $\mathcal{P}^*_{arg}$ to construct such a distinguisher. The distinguisher runs $\mathcal{P}^*_{arg}$ on the public keys and ciphertexts (these are distributed identically in D_1 and D_2). $\mathcal{P}^*_{arg}$ outputs its response ciphertexts $\{b_1, \ldots, b_k\}$, and the distinguisher uses the secret keys $\{sk_i\}_{i<i^*}$ to retrieve the plaintexts $(\beta_1, \ldots, \beta_{i^*-1})$. Starting with the partial transcript $(\alpha_1, \beta_1, \ldots, \alpha_{i^*-1}, \beta_{i^*-1}, \alpha)$, the distinguisher completes the transcript by simulating the interaction between the "optimal-completion prover" $\mathcal{P}'_{IP}$ and the verifier $\mathcal{V}_{IP}$. It outputs 1 if the verifier accepts and 0 otherwise. Observe that:

- On distribution D_2, the probability that the distinguisher outputs 1 is exactly p_{i^*-1}: the distribution of the partial transcript $(\alpha_1, \beta_1, \ldots, \alpha_{i^*-1}, \beta_{i^*-1})$ is identical to $\mathsf{Experiment}_{i^*-1}$. When drawing according to D_2, the i^*-th verifier challenge is α'_{i^*}, which is uniformly random and independent of the preceding partial transcript, as it is in $\mathsf{Experiment}_{i^*-1}$. The remainder of the transcript is also generated using the optimal-completion strategy, exactly as in $\mathsf{Experiment}_{i^*-1}$. The verifier accepts with probability p_{i^*-1}.

- On distribution D_1, the probability that the distinguisher outputs 1 is *at least* p_{i^*}: the distribution of the the partial transcript $(\alpha_1, \beta_1, \ldots, \alpha_{i^*-1}, \beta_{i^*-1})$ is identical to $\mathsf{Experiment}_{i^*}$. When drawing by D_1, the i^*-th challenge α_{i^*} equals the plaintext encrypted in the ciphertext a_{i^*}, exactly as in $\mathsf{Experiment}_{i^*}$. Here, however, the i^*-th prover message β'_{i^*} is drawn according to the optimal completion strategy, whereas in $\mathsf{Experiment}_{i^*}$ we use the plaintext β_{i^*} generated by $\mathcal{P}^*_{arg}$. Still, since the remainder of the transcript will be computed using random verifier queries, replacing the i^*-th prover message with the optimal-completion strategy

cannot decrease the probability that the verifier accepts. The verifier accepts with probability p_{i^*}.

The above distinguisher runs in time $|\mathcal{P}^*_{arg}| + k \cdot T_{IP}(n)$, and has advantage at least $(\varepsilon - s_{IP})/k$ in distinguishing the distributions D_1 and D_2. The theorem follows. □

5.2 Succinct Two-Message Arguments

Instantiating the secure compiler of Theorem 5.3 with the Interactive Proof of Theorem 5.6 below, we obtain succinct two-message arguments for bounded-depth computations:

Theorem 5.5. *Assume the existence of a PIR scheme with communication $\ell_{PIR}(m, \kappa)$ that is secure against time $T_{PIR}(\kappa)$, as per Definition 2.5. Then any language L that can be computed by logspace-uniform circuits of size $\mathrm{poly}(n)$ and depth $D(n) \geq \log n$ has a two-message argument system $(\mathcal{P}_{arg}, \mathcal{V}_{arg})$ with perfect completeness and negligible soundness error against adversaries that run in time $T_{PIR}(\kappa)/\mathrm{poly}(n)$. The communication complexity is $\ell_{PIR}(\mathrm{poly}(n), \kappa) \cdot \mathrm{poly}(D(n))$. $\mathcal{P}_{arg}$ runs in time $\mathrm{poly}(\kappa, n)$ and $\mathcal{V}_{arg}$ runs in time $n \cdot \mathrm{poly}(\kappa, D(n))$.*

This represents an exponential improvement in the security of the resulting argument system. Previously, Kalai and Raz [KR09] showed a similar result, but proved security of the argument system against adversaries running in time $T_{PIR}(\kappa)/2^{\mathrm{poly}(D(n))}$.[11] Interpreting their result, for a language L in NC, to obtain any argument system with $\mathrm{poly}(n)$ communication complexity and security against polynomial-time adversaries (i.e., a non-trivial argument system), quasi-polynomial hardness assumptions are needed (as one needs to have a PIR scheme that is secure against adversaries running in time $T_{PIR}(\kappa) \gg 2^{\mathrm{poly}(D(n))}$). In comparison, our results show that a PIR scheme secure against *polynomial*-time adversaries is sufficient for obtaining $\mathrm{poly}(n)$ communication.

Before proving Theorem 5.5, we review the main result of Goldwasser et al. [GKR08, GKR15]:

Theorem 5.6 (GKR Interactive Proof [GKR15])**.** *Any language L that can be computed by logspace-uniform circuits of size $\mathrm{poly}(n)$ and depth $D(n) \geq \log n$ has a multi-round interactive proof $(\mathcal{P}_{IP}, \mathcal{V}_{IP})$ with perfect completeness, negligible soundness error, and communication complexity $D(n) \cdot \mathrm{polylog}(n)$. Moreover, $\mathcal{P}_{IP}$ runs in time $\mathrm{poly}(n)$ and is $O(\log n)$-history-aware; $\mathcal{V}_{IP}$ is a public-coin logspace verifier, runs in time $n \cdot \mathrm{poly}(D(n))$, and sends messages of length $O(\log(n))$.*

Proof of Theorem 5.5. By Theorem 5.6, the GKR Interactive Proof [GKR15] for L is λ-history-aware, has a log-space public-coin verifier, and verifier messages of length $O(\log n)$. By Theorem 5.2, we conclude that it supports optimal transcript completion in time $\mathrm{poly}(n)$. Plugging this interactive proof into the transformation of Figure 2, and using Theorem 5.3, we obtain a two-message argument with negligible soundness error against $\mathrm{poly}(n, \kappa)$-time adversaries. The communications complexity and the prover and verifier running times follow by the parameters of the interactive proof, the PIR scheme, and Theorem 5.3. □

5.3 Application to Exhaustive Search

The methods of this section are appropriate for the verification of a type of computation that is often distributed among not completely trusted servers, that of exhaustive search. In this setting

[11] the denominator in their result was super-polynomial in n; In particular, it was at least $n^{D(n)}$.

there is some space of solutions S that is partitioned into subspaces $\{S_i\}_i$ and processor i is assigned to search all possible solutions in S_i.

Usually it is easy to identify a successful search, say it satisfies some set of constraints. Therefore it is easy to verify the work of a processor that was successful. But how about verifying the work of an unsuccessful search? How can such an unlucky processor convince, say a central authority, that it performed the computation properly?

A good illustrating example to consider is the case of *Bitcoin mining* (but see caveat below), used to maintain the so called *block chain of transactions*. Here the processors (miners) are looking for a value called a 'nonce', such that when the current block content is (cryptographically) hashed together with the nonce, the result ends with a certain number of leading 0's (i.e. is numerically smaller than the current difficulty target)

A successful search is worth a certain number of Bitcoins (25 as of 2015). Now suppose there is a pool of miners who cooperate in order to reduce the variance in the reward. How can the pool manager verify that searches over the nonce space that were not successful were properly executed?

Given that exhaustive search is "embarrassingly parallel" (one for which no effort is required to separate the problem into a number of parallel tasks), it follows that we can apply the framework of Theorem 5.5.

We can express the search that a processor i should perform of the set S_i as a set of constraints over the set S_i, and for each element in S_i check whether it satisfies the constraints. The circuit will be of depth proportional to $\log |S_i|$ plus the depth of checking whether the set of constraints is satisfied. By Theorem 5.5, assuming the appropriate PIR scheme exists, we will have an argument system whose length is proportional to polynomial in $\log |S_i|$ plus the complexity of checking an instance. In the context of Bitcoin this means that any participant can provide a proof of search whose length is proportional to the nonce length plus the complexity of the hash functions.

Bitcoin pools reward members who come up with nearly-satisfying solutions, i.e. partition the prize according to the closeness. This makes perfect sense in the random oracle world. Our solution may be viewed as more equitable, and does not rely on random oracles for fairness: everybody gets rewarded for the work they perform.

Caveat: given that many things in the Bitcoin world are based on heuristics and on modeling functions as random oracles, the above ideas can probably be thought of as casting pearls before swine. So we prefer to think of it as an illustrating example rather than an actual application.

Acknowledgments

We thank Pavel Hubáček and Ilan Komargodski for helpful comments on the paper.

References

[ABOR00a] William Aiello, Sandeep N. Bhatt, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for np. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 463–474. Springer, 2000.

[ABOR00b] William Aiello, Sandeep N. Bhatt, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for np. *IACR Cryptology ePrint Archive*, 2000:018, 2000.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS*, pages 326–349. ACM, 2012.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

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

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Burton S. Kaliski, editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 1997.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 402–414, 1999.

[Con91] Anne Condon. Space-bounded probabilistic game automata. *J. ACM*, 38:472–494, April 1991.

[DLN+] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for NP ander spooky interactions. manuscript, `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/spooky.pdf`.

[FL93] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theor. Comput. Sci.*, 113(1):55–73, 1993.

[For89] Lance Fortnow. Complexity-theoretic aspects of interactive proof systems. Technical report, Ph.D. Thesis, Laboratory for Computer Science, MIT, 1989.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27, 2015.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 99–108. ACM, 2011.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 723–732. ACM, 1992.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373, 1997.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, pages 143–159, 2009.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC 2014*, pages 485–494, 2014.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[Mie08] Thilo Mie. Polylogarithmic two-round argument systems. *J. Mathematical Cryptology*, 2(4):343–363, 2008.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[PR14] Omer Paneth and Guy N. Rothblum. Publicly verifiable non-interactive arguments for delegating computation. *IACR Cryptology ePrint Archive*, 2014:981, 2014.

[Rot13] Ron Rothblum. On the circular security of bit-encryption. In *TCC*, pages 579–598, 2013.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 475–484, New York, NY, USA, 2014. ACM.