On the (In)security of Kilian-Based SNARGs

James Bartusek* Liron Bronfman† Justin Holmgren‡ Fermi Ma§
Ron D. Rothblum†

Abstract

The Fiat-Shamir transform is an incredibly powerful technique that uses a suitable hash function to reduce the interaction of general public-coin protocols. Unfortunately, there are known counterexamples showing that this methodology may not be sound (no matter what concrete hash function is used). Still, these counterexamples are somewhat unsatisfying, as the underlying protocols were specifically tailored to make Fiat-Shamir fail. This raises the question of whether this transform is sound when applied to natural protocols.

One of the most important protocol for which we would like to reduce interaction is Kilian's four-message argument system for all of NP, based on collision resistant hash functions (CRHF) and probabilistically checkable proofs (PCPs). Indeed, an application of the Fiat-Shamir transform to Kilian's protocol is at the heart of both theoretical results (e.g., Micali's CS proofs) as well as leading practical approaches of highly efficient non-interactive proof-systems (e.g., SNARKs and STARKs).

In this work, we show significant obstacles to establishing soundness of (what we refer to as) the "Fiat-Shamir-Kilian-Micali" (FSKM) protocol. More specifically:

- We construct a (contrived) CRHF for which FSKM is unsound for a very large class of PCPs and for any Fiat-Shamir hash function. The collision-resistance of our CRHF relies on very strong but plausible cryptographic assumptions. The statement is "tight" in the following sense: any PCP outside the scope of our result trivially implies a SNARK, eliminating the need for FSKM in the first place.
- Second, we consider a known extension of Kilian's protocol to an interactive variant of PCPs called *probabilistically checkable interactive proofs* (PCIP) (also known as *interactive oracle proofs* or IOPs). We construct a particular (contrived) PCIP for NP for which the FSKM protocol is unsound no matter what CRHF and Fiat-Shamir hash function is used. This result is unconditional (i.e., does not rely on any cryptographic assumptions).

Put together, our results show that the soundness of FSKM must rely on some special structure of *both* the CRHF and PCP that underlie Kilian's protocol. We believe these negative results may cast light on how to securely instantiate the FSKM protocol by a synergistic choice of the PCP, CRHF, and Fiat-Shamir hash function.

*UC Berkeley. Email: bartusek.james@gmail.com. Research conducted at Princeton University.

†Technion. Email: {br,rothblum}@cs.technion.ac.il. Partially supported by the Israeli Science Foundation (Grant No. 1262/18), a Milgrom family grant, the Technion Hiroshi Fujiwara cyber security research center and the Israel cyber directorate.

‡Simons Institute. Email: holmgren@alum.mit.edu. Research conducted at Princeton University, supported in part by the Simons Collaboration on Algorithms and Geometry and NSF grant No. CCF-1714779.

§Princeton University. Email: fermima@alum.mit.edu. Supported by the NSF and DARPA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or DARPA.

Contents

1 Introduction

The Fiat-Shamir heuristic [FS87] is an extremely influential approach for eliminating or reducing interaction in a wide variety of cryptographic protocols. First proposed as a practical method for constructing digital signature schemes from identification protocols, it was later generalized to reduce interaction of *arbitrary* (public-coin) protocols. In a nutshell, the idea is to replace the messages from the public-coin verifier (which are uniformly random strings) with a suitable hash of all preceding prover messages.

Identifying whether and when the Fiat-Shamir heuristic is sound has been a focus of cryptographic research for decades. It has been known for over 25 years that security holds in an idealized model where the hash function is modeled as a random oracle [PS96]. While the random oracle is often a useful methodology for designing heuristically secure protocols [BR94], it does not provide any guarantees when the random oracle is replaced with any explicit hash function family. As a matter of fact, results of Barak [Bar01] and Goldwasser and Kalai [GK03] give a strong negative indication. Specifically, these works exhibit sound protocols which become totally insecure after the application of the Fiat-Shamir transform *using any hash function*.

Still, the protocols designed by [Bar01, GK03] were, in a sense, specifically tailored to make the Fiat-Shamir fail. Thus, it is important to understand whether this methodology can be soundly applied to protocols with additional natural structure that we care about. A prominent example for such a protocol is Kilian's [Kil92] beautiful 4-message argument-system for any NP language. Remarkably, this protocol (which relies on a relatively mild cryptographic assumption) can be used to prove the correctness of any NP language with an extremely short proof and with a super efficient verification procedure. A main drawback of Kilian's protocol is that it requires back and forth interaction between the prover and verifier (which is undesirable for some applications) and so this protocol is (arguably) the prime example of a protocol for which we would like to apply Fiat-Shamir.

Indeed, this very idea was advocated by Micali [Mic00] in his construction of CS proofs, which are now more commonly referred to as SNARGs (an abbreviation for Succinct Non-interactive ARGuments). SNARGs are an incredibly powerful and versatile tool that are currently being implemented and adopted in practice (especially in the domain of blockchain technology and cryptocurrencies [BGG17, BGM17, BBB+18]). Some leading SNARG implementation efforts are closely following the basic approach of applying Fiat-Shamir to (suitable extensions of) Kilian's protocol [BCS16, BBC+17, BBHR18a, BBHR18b, BCR+19]. For convenience, throughout this work we refer to the candidate SNARG obtained by applying Fiat-Shamir to Kilian's protocol as the FSKM protocol.

Thus, a basic question that we would like to understand (and was posed explicitly by [GK03]) is the following:

Do there exist hash functions with which the FSKM *protocol is sound?*

Jumping ahead, we show that the FSKM protocol can potentially be *insecure* when instantiated with *any* Fiat-Shamir hash function family. However, to explain our results more precisely, we first recall some details of Kilian's original protocol and the resulting FSKM protocol.

1.1 Kilian's Protocol and FSKM

First and foremost, Kilian's protocol relies on *probabilistically checkable proofs* (PCPs). Recall that PCPs can be thought of as a way to encode a witness w so that the encoded witness π can be verified by only reading a few of its bits. The celebrated PCP Theorem [ALM+98] shows that such PCPs exist for all NP languages.

Consider some language $\mathcal{L} \in \mathsf{NP}$ and let $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$ be a PCP proof-system for $\mathcal{L}$. To establish that $x \in \mathcal{L}$, and given a witness w, the prover $\mathcal{P}_{\mathsf{Kilian}}$ engages in the following protocol with the verifier $\mathcal{V}_{\mathsf{Kilian}}$:

1. $\mathcal{V}_{\mathsf{Kilian}}$ samples a collision-resistant hash function $h_{\mathsf{CRHF}} \leftarrow \mathcal{H}_{\mathsf{CRHF}}$, and sends h_{CRHF} to $\mathcal{P}_{\mathsf{Kilian}}$.
2. $\mathcal{P}_{\mathsf{Kilian}}$ constructs a PCP π for x's membership in $\mathcal{L}$, and sends a Merkle hash (using h_{CRHF}) of π to $\mathcal{V}_{\mathsf{Kilian}}$.

3. $\mathcal{V}_{\mathsf{Kilian}}$ chooses random coins r for the PCP verifier $\mathcal{V}_{\mathsf{PCP}}$ and sends them to $\mathcal{P}_{\mathsf{Kilian}}$.

4. $\mathcal{P}_{\mathsf{Kilian}}$ computes the locations $i_1, \ldots, i_q$ that $\mathcal{V}_{\mathsf{PCP}}$ would query on randomness r and input x, and "decommits" to the values of $(\pi_{i_1}, \ldots, \pi_{i_q})$.[1]

5. $\mathcal{V}_{\mathsf{Kilian}}$ checks that the decommitments are valid and that the values that were opened make the PCP verifier accept using the random string r.

We denote by $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ the protocol that results from applying Fiat-Shamir with hash family $\mathcal{H}_{\mathsf{FS}}$ to the above 4 message argument. Observe that $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ can be implemented using only 2 messages: in the first message the verifier specifies the collision resistant hash function and the Fiat-Shamir hash function $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}$, and in the second message the prover reveals the root rt of the Merkle tree together with the relevant decommitments with respect to $r = h_{\mathsf{FS}}(\mathsf{rt})$.

We will also consider a variant of the FSKM protocol that uses a generalization of PCPs called probabilistically checkable *interactive* proofs or PCIPs [BCS16, RRR16].[2] A PCIP is a type of proof system that combines the locally checkable aspect of PCPs with the multi-round aspect of interactive proofs, thereby generalizing both. More precisely, in a PCIP the prover first sends a PCP proof to the verifier, which can make some queries to this proof string (as in standard PCPs). The difference however is that now the verifier is allowed to respond with a random challenge and the prover sends an *additional* PCP proof string - this process can continue for several rounds.

One of the key benefits of PCIPs (advocated by [BCS16]) is that they can allow for practical efficiency benefits over standard PCPs. As observed by [BCS16], Kilian's protocol can be readily extended to handle PCIPs, by having the prover send a Merkle hash of its entire message in every round, and eventually decommiting to the desired bits as in Kilian's original protocol. It is natural therefore to apply Fiat-Shamir to the resulting protocol and this was shown to be sound by [BCS16] in the *random oracle model*. We extend our notation of FSKM to the more general setting of PCIPs in the natural way (see Section 2.3.2 for details).

As briefly mentioned above, the FSKM protocol, when combined with highly efficient PCIPs is at the heart of current successful implementations of SNARGs [BCS16, BBC+17, BBHR18a, BBHR18b, BCR+19].

1.2 Our Results

Loosely speaking, we show that the FSKM protocol can be insecure when instantiated with *any* Fiat-Shamir hash function unless security relies on specific properties of both (1) the collision resistant hash function, *and* (2) the underlying PCP (or more precisely PCIP). This is established by our two main results which are described next.

Our first main result shows that there exists a collision-resistant hash family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that for any "reasonable" PCP Π_{PCP} and *all* candidate Fiat-Shamir hash families $\mathcal{H}_{\mathsf{FS}}$, the protocol $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound. We refer to such a CRHF as being FSKM-incompatible. The existence of such an FSKM-incompatible CRHF shows that soundness of the FSKM protocol cannot be based on a generic CRHF. Loosely speaking, by "a reasonable PCP," we mean one where it is possible *given the verifier's randomness* to compute a proof string that the verifier would accept.

Unreasonable PCPs *may* exist. For instance, if SNARGs exist, then any PCP for an NP language $\mathcal{L}$ can be modified (in a contrived way) to be an unreasonable PCP for $\mathcal{L}$: Honest proof strings for $x \in \mathcal{L}$ are modified by appending a SNARG π_{SNARG} attesting that $x \in \mathcal{L}$; the verifier is modified so that in addition to performing the original PCP verifier's checks, it also verifies that π_{SNARG} is a valid SNARG. However there is a sense in which such PCPs (already having an embedded SNARG) are the only unreasonable PCPs. We formalize this in Theorem 3.5.

The collision-resistance of our FSKM-Incompatible CRHF relies on a strong cryptographic assumption: the existence of Succinct Non-Interactive ARguments of Knowledge (SNARKs) with "computationally unique"

[1] A succinct decommitment to the value π_{i_j} can be accomplished by having $\mathcal{P}_{\mathsf{Kilian}}$ reveal the hash values of all vertices in the tree that are either on, or adjacent to, the path from π_{i_j} to the root in the Merkle tree.

[2] PCIPs are also called *interactive oracle proofs* IOPs.

proofs. By computationally unique we mean that it should be infeasible to find two different proofs corresponding to the same NP witness.

Our result implies that under this assumption, $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ cannot be proven to be sound when $\mathcal{H}_{\mathsf{CRHF}}$ is a generic collision-resistant hash family — even if the PCP Π_{PCP} and Fiat-Shamir hash function $\mathcal{H}_{\mathsf{FS}}$ are carefully engineered.

Theorem 1 (Informally Stated, see Theorem 3.5)**.** *Assume the existence of collision resistant hash functions and (publicly verifiable)* SNARK*s with computationally unique proofs. Then, there exists a collision-resistant hash family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that for every "reasonable"* PCP Π_{PCP} *and every hash family $\mathcal{H}_{\mathsf{FS}}$, the protocol $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound.*

We instantiate this theorem with a SNARK constructed in the works of [BCI+13, BCCT13], whose soundness follows from a knowledge of exponent assumption in bilinear groups (along with a more standard "power" discrete log assumption, see Assumption A.3). Such knowledge assumptions are very strong (and are not known to be falsifiable), but are still plausible, and in particular can be proven to hold in the generic group model [Nec94, Sho97, Mau05].[3] In Appendix A, we show that under the same set of assumptions (we need collision resistant hashing as well, but this follows from either of the assumptions on groups), this SNARK has computationally unique proofs. However, as discussed in [BCCT13], the soundness notion satisfied by this SNARK is slightly weaker than standard soundness. We overcome this difficulty by additionally assuming the existence of injective one-way functions that are exponentially hard.

Moving on, Theorem 1 still leaves open the possibility that a careful choice of $\mathcal{H}_{\mathsf{CRHF}}$ and $\mathcal{H}_{\mathsf{FS}}$ suffices to establish soundness of $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$. Our second main result shows a significant obstacle to this possibility. Specifically, we show that there exists a PCIP such that for any collision-resistant hash function CRHF *and* any Fiat-Shamir hash function, the resulting FSKM protocol is not sound. We refer to such a PCIP as being an FSKM-incompatible PCIP. The existence of FSKM-incompatible PCIPs implies that the soundness of any FSKM protocol must rely on specific properties of the underlying PCIP. In contrast to Theorem 1, this result is unconditional (i.e., does not rely on any cryptographic assumptions).

Theorem 2 (Informally Stated, see Theorem 4.1)**.** *There exists a* PCIP $\widetilde{\Pi}_{\mathsf{PCIP}}$ *such that for all hash families $\mathcal{H}_{\mathsf{CRHF}}$ and $\mathcal{H}_{\mathsf{FS}}$, the protocol $\mathsf{FSKM}[\widetilde{\Pi}_{\mathsf{PCIP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound.*

Actually, in the proof of Theorem 2, we show that the soundness of $\mathsf{FSKM}[\widetilde{\Pi}_{\mathsf{PCIP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is broken in an extremely strong sense. Namely, there exists a polynomial-time adversary that convinces the FSKM verifier to accept *any* input $x \notin L$ with probability 1.

Interpretation of Our Results. We emphasize that our construction of CRHF (in Theorem 1) and PCIP (in Theorem 2) are highly contrived. Thus, it certainly remains a possibility that some special structure of known CRHFs and PCPs/PCIPs might be identified that will allow for the FSKM protocol to be securely instantiated. Indeed, we hope that our results will lead to the identification of special structure that can be leveraged to securely instantiate FSKM.

In fact, we give some initial progress towards this goal. In Section 5, we give a sound instantiation (under standard assumptions) of the FSKM protocol when the underlying PCP is a specific PCP for the empty language. In order to bypass our impossibility, we make use of a collision-resistant hash function with special structure: the *somewhere statistically binding* hash function of Hubáček and Wichs [HW15b]. For the Fiat-Shamir hash function, we use a *correlation-intractable* hash function for efficiently searchable relations, recently constructed under the Learning with Errors assumption by Peikert and Shiehian [PS19]. Needless to say, SNARK constructions for the empty language are not particularly interesting. However,

[3] The work of [BCPR14] showed that if indistinguishability obfuscation exists, then SNARKs where extraction holds with respect to *arbitrary* unbounded polynomial length auxiliary input do not exist. We therefore rely on a version of the [BCI+13] knowledge of exponent assumption which only requires extraction to hold with respect to auxiliary input from a "benign" distribution (e.g. a uniform distribution); a similar approach was taken in [CFH+15, FFG+16, Gro16, BCC+17]. We are able to rely on this relaxed version since the auxiliary input in our construction essentially just consists of the key for some arbitrary collision resistant hash function. We discuss this issue in further detail in Appendix A.1.

we hope that this blueprint will be useful in the future for proving instantiations of FSKM sound when the underlying PCP is defined for more expressive languages.

1.3 Additional Prior Work

Goldwasser and Kalai [GK03] showed that the original application of the Fiat-Shamir heuristic is not sound; there exists a contrived identification protocol such that no matter what hash function is used in the Fiat-Shamir transform, the resulting digital signature scheme is insecure. Since they use a very particular protocol, their result does not yield a negative result for applying Fiat-Shamir to the FSKM protocol (indeed, as mentioned above, finding such a negative result was posed as an open problem in [GK03]).

Another very related work is that of Gentry and Wichs [GW11], who showed a substantial barrier to constructing SNARGs. Our work is incomparable to that of [GW11]. On the one hand [GW11] rule out a very general class of SNARG constructions whereas we focus on a very particular approach (i.e., applying Fiat-Shamir to FSKM with a generic CRHF). On the other hand, when restricting to the foregoing approach, we overcome some significant limitations of [GW11]. First, in contrast to [GW11], our result is not limited to SNARGs whose security holds under a *black-box* reduction from a falsifiable assumption. Second, it applies also to constructions based on *non-falsifiable* assumptions. Third, it rules out protocol achieving standard (i.e., non-adaptive) soundness, whereas [GW11] only rules out adaptively sound protocols. And fourth, our work applies to any NP language whereas [GW11] only rules out SNARGs for particular (extremely) hard NP languages.

A recent line of work [KRR17, CCRR18, HL18, CCH+19, PS19] constructs hash functions that are compatible with Fiat-Shamir, when applied to *statistically sound* interactive proofs. Still, the question of whether the Fiat-Shamir transform can be securely applied to preserve the *computational* soundness of Kilian's argument scheme has remained open.

1.4 Technical Overview

We proceed to an overview of our two main results. First, in Section 1.4.1 we give an overview of our FSKM-incompatible CRHF and then, in Section 1.4.2, we give an overview of our FSKM-incompatible PCIP.

1.4.1 An FSKM-incompatible CRHF

For simplicity, in this overview we describe a weaker result. Specifically, we construct an FSKM-incompatible CRHF for a particular choice of the language $\mathcal{L}$ and for a PCP for $\mathcal{L}$ (rather than handling all languages $\mathcal{L}$ and all "reasonable" PCPs). Nevertheless, this weaker result demonstrates the main ideas that go into the proof of Theorem 1. Specifically, we focus on the empty language $\mathcal{L} = \emptyset$. While this language has a trivial PCP Π_{PCP} (of length 0) in which the verifier always rejects, we will consider a *different* PCP Π_{PCP} for $\mathcal{L}$ (parameterized by a security parameter λ): the PCP proof string is expected to have length 2λ and the verifier uses a random string of length λ, and accepts if the first λ bits of the PCP are equal to its random string.[4] Completeness holds in an empty sense whereas the soundness error is clearly $2^{-\lambda}$. We construct a contrived collision-resistant hash function $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that $\mathsf{FSKM}[\widetilde{\Pi}_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound for any hash family $\mathcal{H}_{\mathsf{FS}}$.

We will construct $\widetilde{\mathcal{H}}_{\mathsf{CRHF}} = \{\widetilde{H}^{(\lambda)}_{\mathsf{CRHF}} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}\}$ so that it satisfies the following property: Given $\widetilde{h}_{\mathsf{CRHF}} \leftarrow \widetilde{H}^{(\lambda)}_{\mathsf{CRHF}}$ and any efficiently computable function f, it is possible to efficiently find $(x_0 \| x_1) \in \{0,1\}^{2\lambda}$ such that $x_0 = f(\widetilde{h}_{\mathsf{CRHF}}(x_0 \| x_1))$. This property immediately allows us to break the soundness of $\mathsf{FSKM}[\widetilde{\Pi}_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ as follows. We view $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}$ as the function f, and so a cheating prover can produce a valid commitment rt to a string $(x_0 \| x_1)$ such that the verifier's randomness is $h_{\mathsf{FS}}(\mathsf{rt}) = x_0$. The prover sends rt as the Merkle root but can now decommit to x_0, which makes the PCP verifier accept. We refer to a CRHF having the foregoing property as a circular tractable (CT-) CRHF.

[4] Note that this is not the same empty language PCP that we use in Section 5 - our result in Section 5 holds for a more general class of empty language PCPs.

A CT-CRHF from Ideal Obfuscation. We first illustrate how it is possible to construct a CT-CRHF assuming ideal Turing-machine obfuscation. For readability, we will use collision-resistant hash *functions* (rather than ensembles) in this section.

We will start with any CRHF $h'_{\mathsf{CRHF}} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda-1}$, and we construct a CT-CRHF hash function $\widetilde{h}_{\mathsf{CRHF}} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$. The hash function $\widetilde{h}_{\mathsf{CRHF}}$ will have two types of outputs: normal outputs, which end in a 1, and special outputs, which end in $0^{\lambda/2}$. On almost all inputs $x_0\|x_1$, we will have

$$\widetilde{h}_{\mathsf{CRHF}}(x_0\|x_1) = h'_{\mathsf{CRHF}}(x_0\|x_1)\|1.$$

However, we will guarantee that for every x_0 and special output $y\|0^{\lambda/2}$, x_0 can be extended into a (not efficiently computable) "special" $x_0\|x_1$ such that $\widetilde{h}_{\mathsf{CRHF}}(x_0\|x_1) = y\|0^{\lambda/2}$. This is easy to achieve if we augment the description of $\widetilde{h}_{\mathsf{CRHF}}$ to include a verification of a (public-key) digital signature scheme, and $\widetilde{h}_{\mathsf{CRHF}}$ is defined as

$$\widetilde{h}_{\mathsf{CRHF}}(x_0|x_1) = \begin{cases} y\|0^{\lambda/2} & \text{if } x_1 = y\|\sigma, \text{ for } \sigma \text{ a valid signature of } (x_0, y). \\ h'_{\mathsf{CRHF}}(x_0\|x_1)\|1 & \text{otherwise.} \end{cases}$$

In order to actually (efficiently) use the added structure of $\widetilde{h}_{\mathsf{CRHF}}$, we will also augment the description of $\widetilde{h}_{\mathsf{CRHF}}$ to include an obfuscation $\widehat{P}$ of a program P that has the signature signing key hard-wired, and on input the description $\langle f \rangle$ of a function f acts as follows:

1. Computes $y = h''_{\mathsf{CRHF}}(\langle f \rangle)$, where $h''_{\mathsf{CRHF}} : \{0,1\}^* \to \{0,1\}^{\lambda/2}$ is a generic CRHF.
2. Compute $x_0 = f(y\|0^{\lambda/2})$.
3. Compute a signature σ of (x_0, y).
4. Output (x_0, y, σ).

It is clear that the inclusion of $\widehat{P}$ in the description of $\widetilde{h}_{\mathsf{CRHF}}$ makes $\widetilde{h}_{\mathsf{CRHF}}$ circular tractable, but why is $\widetilde{h}_{\mathsf{CRHF}}$ collision-resistant?

Suppose that an efficient adversary $\mathcal{A}$ were to output a colliding pair $(x_0\|x_1)$ and $(x'_0\|x'_1)$ of $\widetilde{h}_{\mathsf{CRHF}}$. The only new collisions that $\widetilde{h}_{\mathsf{CRHF}}$ has compared to h'_{CRHF} (and thus that $\mathcal{A}$ might possibly output) are collisions for special outputs (standard outputs can never collide with special outputs due to their last bit being different). That is, we may assume that $x_1 = y\|\sigma$ and $x'_1 = y\|\sigma'$. The security of the signature scheme and of the ideal obfuscator imply that if $\mathcal{A}$ produced such a collision, it must have queried $\widehat{P}$ on two distinct inputs $\langle f \rangle$, $\langle f' \rangle$ such that $P(\langle f \rangle) = (x_0, y, \sigma)$ and $P(\langle f' \rangle) = (x'_0, y, \sigma')$. But this would in particular imply that $h''_{\mathsf{CRHF}}(\langle f \rangle) = h''_{\mathsf{CRHF}}(\langle f' \rangle) = y$, meaning that $\mathcal{A}$ found a collision in h''_{CRHF}, which is a contradiction.

A CT-CRHF from Unique-Proof SNARKs. The above construction is tantalizing, but unfortunately we do not know how to prove security (collision-resistance) from any general-purpose notion of obfuscation security (e.g., indistinguishability obfuscation) that is not known to be unachievable. Instead, we show how to use similar ideas to obtain a CT-CRHF using special SNARKs that can be constructed based on a knowledge of exponent assumption.

Taking a closer look at the obfuscation-based construction, we observe that ideal obfuscation was used to ensure that if an adversary $\mathcal{A}$ could come up with an input $x_0\|x_1$ such that $\widetilde{h}_{\mathsf{CRHF}}(x_0\|x_1)$ is the special output $y\|0^{\lambda/2}$, then $\mathcal{A}$ must "know" an f such that $h''_{\mathsf{CRHF}}(\langle f \rangle) = y$ (and $x_0\|x_1$ are a fixed function of f).

With this observation in mind, an alternative way of defining special inputs is as the set of $x_0\|x_1$ that contain a SNARK of this fact. That is, let special inputs be strings x of the form $x_0\|y\|\pi$, where π is a valid proof of knowledge of f satisfying $h''_{\mathsf{CRHF}}(\langle f \rangle) = y \wedge x_0 = f(y)$, and on such inputs let $\widetilde{h}_{\mathsf{CRHF}}(x) = y\|0^{\lambda/2}$.

Is the resulting $\widetilde{h}_{\mathsf{CRHF}}$ collision-resistant? There are two types of collisions that we need to consider. The first type is collisions of the form $x_0\|y\|\pi$, $x'_0\|y\|\pi'$ with $x_0 \neq x'_0$. The second type is collisions in which $x_0 = x'_0$ (but $\pi \neq \pi'$).

The first type of collision is ruled out by the standard SNARK proof-of-knowledge property. If an adversary produces such a collision, then there is an extractor that produces $\langle f \rangle$, $\langle f' \rangle$ such that $h''_{\mathsf{CRHF}}(\langle f \rangle) = h''_{\mathsf{CRHF}}(\langle f' \rangle) = y$ but $f(y) = x_0 \neq x'_0 = f'(y)$. The latter inequality implies that $\langle f \rangle \neq \langle f' \rangle$, which means that the extractor is finding a collision in h''_{CRHF}.

To rule out the second type of collision, we require a new "unique proofs" property for the SNARK to ensure that the extracted $\langle f \rangle$, $\langle f' \rangle$ are distinct. Informally, this property says that for any adversary $\mathcal{A}$ that comes up with two distinct valid SNARK-proofs of the same NP claim, there is an extractor $\mathcal{E}$ that comes up with two distinct NP witnesses for the same claim.

1.4.2 An FSKM-incompatible PCIP

For every language $\mathcal{L} \in \mathsf{NP}$ and collision resistant hash function ensemble $\mathcal{H}_{\mathsf{CRHF}}$, we present a contrived PCIP $\widetilde{\Pi}_{\mathsf{PCIP}}$, such that for *any* choice of Fiat-Shamir hash function ensemble $\mathcal{H}_{\mathsf{FS}}$, the resulting protocol $\mathsf{FSKM}[\widetilde{\Pi}_{\mathsf{PCIP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound.

The PCIP construction is inspired by and builds on Barak's [Bar01] beautiful protocol, while taking steps to make the approach compatible with Kilian's protocol. Roughly speaking, our approach is to take an arbitrary PCP for $\mathcal{L}$ (say the one established by the classical PCP theorem) and "tweaking" it so as to maintain its soundness while enabling an attack on the resulting FSKM protocol. Since the tweaking of the PCP will add an additional round, we only obtain a FS-incompatible PCIP rather than a PCP.

In more detail, the first message sent by the honest PCIP prover is $\pi' = b \| \pi$ where b is a single bit and π is a string. The honest PCIP prover *always* sets the bit b to 0 but we add the option of having a malicious prover set b to 1 to facilitate the attack on FSKM.

The PCIP verifier, given this string, first reads the value of b. In case $b = 0$, the verifier simply treats π as a PCP proof string and runs the underlying PCP verifier while redirecting its proof queries to π. This concludes the entire interaction and the PCIP verifier accepts if and only if the PCP verifier accepts. Completeness of the entire protocol as well as soundness for the case that $b = 0$ follow immediately from the construction.

Note however that a malicious prover may indeed send the value $b = 1$. While the verifier could immediately reject in this case, we intentionally make our PCIP verifier do something different. Ignoring π for a moment, the verifier now chooses a random string $r \in \{0,1\}^\lambda$ and sends r to the PCIP prover. The key observation is that when the protocol is compiled via FSKM using a CRHF h_{CRHF} and FS hash function h_{FS}, in the resulting non-interactive argument the value r is fully determined. More specifically, it will always be the case that $r = h_{\mathsf{FS}}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$, where MerkleCom simply computes a Merkle tree of the string π' using the hash function h_{CRHF} and outputs its root. Thus, in order to facilitate the attack, we would like to design our PCIP verifier to accept if it happens to be the case that $r = h_{\mathsf{FS}}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$.

What may seem initially problematic is that it is unclear how the PCIP verifier can know which CRHF and FS hash functions will be used in the FSKM protocol. We handle this by simply letting the PCIP prover specify these functions as part of $\pi = (h_{\mathsf{CRHF}}, h_{\mathsf{FS}})$. Thus, after sampling r, we would like for our PCIP verifier to check that $\pi = (h_{\mathsf{CRHF}}, h_{\mathsf{FS}})$ such that $r = h_{\mathsf{FS}}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$. Suppose for now that the PCIP verifier does this explicitly (i.e., by reading all of π'). Observe that the PCIP remains sound since r is chosen after the value $h_{\mathsf{FS}}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$ is fully determined (and so the probability that r is equal to this value is exponentially vanishing).

On the other hand, we can now demonstrate an attack on the resulting FSKM protocol. Consider a cheating FSKM prover that works as follows. Recall that the FSKM verifier gives the prover descriptions of a CRHF h_{CRHF} and an FS hash function h_{FS}. The prover now sets $\pi = (h_{\mathsf{CRHF}}, h_{\mathsf{FS}})$ and continues as in the FSKM protocol while using $\pi' = (1, \pi)$ as the first PCIP message. In more detail, it computes and sends a Merkle root $\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi')$ to the verifier. By design, the prover and verifier now agree to use the "random" string $r = h_{\mathsf{FS}}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$ which makes all of the verifier's tests pass.

A final difficulty that we need to overcome is that the PCIP verifier as described so far has *linear* query complexity since in case $b = 1$ it reads the entire message π. We resolve this by replacing the explicit test done by the verifier with another round of interaction. In more detail, when $b = 1$, after receiving r, the prover is expected to send an *additional* PCP *proving* that $r = h_{FS}(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi'))$ holds. Actually, a standard PCP will not suffice since a PCP verifier reads its entire input (which in our case is the first PCIP

message π). Rather, we will use a PCP *of proximity* (PCPP) [BGH+06, DR06] which is a PCP in which the verifier only reads a few bits of its input and is required to reject inputs that are *far* from the language. To make this approach work, the prover will actually send π encoded under an error-correcting code. We defer further details to the technical sections.

1.5 Organization

In Section 2 we give preliminaries. The proof of Theorem 1, via a construction of an FSKM-Incompatible CRHF is presented in Section 3. The proof of Theorem 2, via a construction of an FSKM-Incompatible PCIP is presented in Section 4. In Section 5, we give a sound instantiation of the FSKM protocol for a specific PCP for the empty language. Lastly, in Appendix A we give the candidate construction of a SNARK with computationally unique proofs.

2 Preliminaries

We let λ denote the security parameter. Let $[n] = \{1, \ldots, n\}$. Throughout, we will use $\langle P \rangle$ to denote the description of a function/machine/program P. A function $\epsilon(\lambda)$ is said to be negligible, if for every $c \in \mathbb{N}$ it holds that $\epsilon(\lambda) = O(\lambda^{-c})$.

We let $\mathcal{H} = \{\mathcal{H}^{(\lambda)}\}_\lambda$, where $\mathcal{H}^{(\lambda)} = \{h : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}_h$ denote a hash function ensemble, where hash functions $h : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ are sampled as $h \leftarrow \mathcal{H}^{(\lambda)}$.

The *relative distance* between strings $x, y \in \Sigma^\ell$ is $\Delta(x,y) = |\{i \mid x_i \neq y_i\}|/\ell$. The *relative distance* of a string $x \in \Sigma^\ell$ from a (non-empty) set $S \subseteq \Sigma^\ell$ is $\Delta(x,S) = \min_{y \in S}\big(\Delta(x,y)\big)$.

2.1 Proof Systems

In this work we adhere to the convention in which all proof systems (as well as other cryptographic primitives) are relative to a security parameter λ (given in unary representation to all parties) and with soundness error that is negligible in λ.

2.1.1 Argument Systems (aka Computationally Sound Proofs)

The interaction between a prover $\mathcal{P}$, on input x and security parameter 1^λ, and a verifier $\mathcal{V}$, with input y and the same security parameter λ, is denoted by $\langle \mathcal{P}(x,\lambda) \leftrightarrow \mathcal{V}(y,\lambda) \rangle$ and includes a polynomial number of rounds in which each party sends the other a message. The interaction terminates when the verifier $\mathcal{V}$ decides whether to accept or reject its input y. The result of the interaction is the bit $b \in \{0,1\}$ returned by $\mathcal{V}$ indicating whether it accepted, which is denoted by $\langle \mathcal{P}(x) \leftrightarrow \mathcal{V}(y) \rangle$. If $b = 1$, then we say that $\mathcal{V}$ accepts.

Definition 2.1 (Argument system)**.** *An* argument system *for a language $\mathcal{L} \in$ NP, with soundness error $s : \mathbb{N} \to [0,1]$ is a pair of probabilistic polynomial-time algorithms $\mathcal{P}$ and $\mathcal{V}$ such that:*

1. **Completeness:** *If $x \in \mathcal{L}$ and w is a corresponding witness, then for every security parameter λ it holds that*
$$\Pr\left[\langle \mathcal{P}(x,w,1^\lambda) \leftrightarrow \mathcal{V}(x,1^\lambda) \rangle = 1\right] = 1.$$
2. **Computational soundness:** *If $x \notin \mathcal{L}$, then for every probabilistic polynomial-time malicious prover $\mathcal{P}^*$ and all sufficiently large security parameters λ, it holds that*
$$\Pr\left[\langle \mathcal{P}^*(x,1^\lambda) \leftrightarrow \mathcal{V}(x,1^\lambda) \rangle = 1\right] \leq s(\lambda).$$

An argument system $(\mathcal{P}, \mathcal{V})$ is said to be *public-coin* if all messages sent by the verifier $\mathcal{V}$ are random-coin tosses, and $\mathcal{V}$ does not toss any additional random coins.

2.1.2 Probabilistically Checkable Proofs (PCPs)

Roughly speaking, a *probabilistically checkable proof (*PCP*)* is an encoding of an NP witness that can be verified by reading only a few of its bits. More formally:

Definition 2.2 (Probabilistically checkable proof)**.** *A* probabilistically checkable proof *(*PCP*) for a language $\mathcal{L} \in$ NP consists of a polynomial-time algorithm $\mathcal{P}$, which receives a main input x and witness w, and a probabilistic polynomial-time oracle machine $\mathcal{V}$, which receives x and a security parameter 1^λ as explicit inputs, and oracle access to a proof π. The* PCP *has soundness-error $s : \mathbb{N} \to [0,1]$ if:*

1. **Completeness:** *If $x \in \mathcal{L}$ and w is a corresponding witness, then for $\pi = \mathcal{P}(x, w)$ and every λ holds that*
$$\Pr\left[\mathcal{V}^\pi(x, 1^\lambda) = 1\right] = 1.$$
2. **Soundness:** *If $x \notin \mathcal{L}$, for every proof π^* and security parameter λ it holds that*
$$\Pr\left[\mathcal{V}^{\pi^*}(x, 1^\lambda) = 1\right] < s(\lambda).$$

In order to query π, the verifier $\mathcal{V}$ tosses $r = r(|x|, \lambda)$ random coins and generates $q = q(|x|, \lambda)$ queries. It will often be convenient to view $\mathcal{V}$ as two separate algorithms $(\mathcal{V}_0, \mathcal{V}_1)$. The first, $\mathcal{V}_0(x; r)$, runs on the instance x and randomness r and outputs the set of queries $\{q_i\}_i$ that $\mathcal{V}$ makes to π. The second algorithm, $\mathcal{V}_1(\{b_i\}_i, x; r)$, takes the corresponding responses $\{b_i\}_i$ as input (as well as the instance x and the same randomness r as $\mathcal{V}_0$), and decides whether to accept or reject.

The following celebrated theorem by [ALM+98] establishes the expressive power of PCPs.

Theorem 2.3 (PCP theorem)**.** *Every language $\mathcal{L} \in$ NP has a* PCP *with soundness error $\frac{1}{2}$, constant query complexity, and logarithmic randomness complexity.*

Note that a PCP with negligible soundness error can be easily obtained from Theorem 2.3 by having the verifier generate $\mathsf{polylog}(\lambda)$ independent query sets.

2.1.3 Probabilistically Checkable Proofs of Proximity (PCPP)

In a standard PCP, the verifier is explicitly given the entire input x along with access to an oracle that encodes a "probabilistically checkable" witness. In contrast, in a PCP of proximity (PCPP) [BGH+06, DR06] the goal is for the verifier to decide without even reading the entire input. Thus, the verifier is given oracle access to the input and we count the total number of queries to both the input and the proof.

Since the verifier cannot even read the entire input, the notion of soundness in PCPP is relaxed: the verifier must only reject inputs that "far" from the language (i.e. where distance is measured in Hamming distance).

Following [BGH+06], we define PCPPs with respect to *pair-languages*, which are simply a subset of $\{0,1\}^* \times \{0,1\}^*$. The *projection of a pair-language* $\mathcal{L}$ on x is $\mathcal{L}(x) = \{y \mid (x, y) \in \mathcal{L}\}$.

In our context, we view the first component x of a pair $(x, y) \in \mathcal{L}$ as an *explicit* input for the verifier whereas the second component, y, is an *implicit* input (i.e., the verifier only has oracle access to y). We count the total number of queries to the oracle y, as well as the proof string π. The soundness requirement is that the verifier has to reject words in which the implicit input is far from the projection of $\mathcal{L}$ onto x.

Definition 2.4 (PCPP)**.** *A* probabilistically checkable proof of proximity *(*PCPP*) for a pair-language $\mathcal{L} \in$ NP consists of a polynomial-time prover $\mathcal{P}$ that gets as input a pair (x, y) as well as a witness w, and a probabilistic polynomial-time oracle machine $\mathcal{V}$ that receives x as an explicit input, oracle access to y and oracle access to a proof string π. The verifier also receives (explicitly) a proximity parameter $\delta > 0$ and security parameter 1^λ. The* PCPP *has soundness error $s : \mathbb{N} \to [0,1]$ if for every proximity parameter $\delta \in [0,1]$, security parameter $\lambda > 0$ and input (x, y):*

1. **Completeness:** *If* $(x, y) \in \mathcal{L}$ *and* w *is the corresponding witness, for* $\pi = \mathcal{P}((x, y), w)$ *it holds that*

$$\Pr[\mathcal{V}^{y,\pi}(x, |y|, |\pi|, 1^\lambda, \delta) = 1] = 1.$$

2. **Soundness:** *If* $\Delta(y, \mathcal{L}(x)) > \delta$ *and oracle* π^**, it holds that*

$$\Pr[\mathcal{V}^{y,\pi^*}(x, |y|, |\pi^*|, 1^\lambda, \delta) = 1] < s(\lambda).$$

The verifier $\mathcal{V}$ generates $r = r(|x|, \lambda)$ random coins and makes $q = q(|x|, \lambda)$ queries for both oracles. We omit the lengths of the implicit input y and the proof π from the input of the verifier when these are clear from the context.

Ben-Sasson *et al.* [BGH+05] give a construction of PCPP for all of NP (with a suitably efficient verifier).

Theorem 2.5 ([BGH+05])**.** *For every language* $\mathcal{L} \in \mathit{NTIME}(T)$ *and every constant* $\delta \in [0, 1]$*, there exists a* PCPP *for* $\mathcal{L}$ *with respect to proximity parameter* δ*, soundness-error of* $\frac{1}{2}$ *and proof length* $\mathsf{poly}(T)$*. The verifier runs in time* $\mathsf{polylog}(T)$ *and the prover runs in time* $\mathsf{poly}(T)$ *(i.e., the* PCPP *proof can be generated in time* $\mathsf{poly}(T)$ *given the* NP *witness).*

2.1.4 Probabilistically Checkable Interactive Proofs

Probabilistically checkable interactive proofs (PCIPs) [BCS16, RRR16] (also known as *interactive oracle proofs*) are generalizations of both interactive proofs and PCPs. They allow for multi-round interactions, in which the prover provides the verifier with oracle access to long proof strings, but we only count the number of bits that were actually queried by the verifier. In this work, we will only consider public-coin PCIPs.

Definition 2.6 (Probabilistically checkable interactive proof)**.** *A* probabilistically Checkable Interactive Proof *(*PCIP*) for a language* $\mathcal{L} \in$ NP *consists of a pair of interactive probabilistic machines* $(\mathcal{P}, \mathcal{V})$*. The prover* $\mathcal{P}$ *is a deterministic polynomial-time algorithm, which gets as input* x *a witness* w *and a security parameter* λ*, and the verifier* $\mathcal{V}$ *is a PPT algorithm, which gets as input* x *and* λ*. The interaction consists of the following* 3 *phases:*

1. **Communication phase:** *The two parties interact for* $k = k(|x|, \lambda)$ *rounds, in which* $\mathcal{V}$ *only sends random strings of total length* $r = r(|x|, \lambda)$ *and* $\mathcal{P}$ *sends proofs strings* $\pi_1, ..., \pi_{k(|x|,\lambda)}$*, where* π_i *is sent in the* i*-th round.*

2. **Query phase:** *In which* $\mathcal{V}$ *sends makes a total of* $q = q(|x|, \lambda)$ *queries to the messages sent by* $\mathcal{P}$ *in the communication phase.*

3. **Decision phase:** *Based on its random messages in the communication phase, and the answers to its queries in the query phase, the verifier* $\mathcal{V}$ *decides whether to accept or reject.*

The PCIP $(\mathcal{P}, \mathcal{V})$ *has soundness* $s : \mathbb{N} \to [0, 1]$ *if:*

1. **Completeness:** *If* $x \in \mathcal{L}$ *and* w *is the corresponding witness, then for every security parameter* λ *it holds that*

$$\Pr\left[\langle \mathcal{P}(x, w, \lambda) \leftrightarrow \mathcal{V}(x, \lambda)\rangle = 1\right] = 1.$$

2. **Soundness:** *If* $x \notin \mathcal{L}$*, then* $\forall \mathcal{P}^*$ *and security parameters* λ*, it holds that*

$$\Pr\left[\langle \mathcal{P}^*(x, \lambda) \leftrightarrow \mathcal{V}(x, \lambda)\rangle = 1\right] < s(\lambda).$$

2.2 Kilian's Protocol

Before describing Kilian's protocol, we first recall the definition of collision resistant hash functions and Merkle trees.

2.2.1 CRHF and Merkle Trees

An efficiently computable hash function ensemble $\mathcal{H} = \{\mathcal{H}^{(\lambda)}\}_\lambda$ where $\mathcal{H}^{(\lambda)} = \{h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda\}$ is *collision resistant* (CRHF), if there exists a key generation algorithm Gen that on input 1^λ samples h from $\mathcal{H}^{(\lambda)}$ such that for every PPT adversary $\mathcal{A}$ it holds that

$$\Pr_{\substack{h \leftarrow \mathsf{Gen}(1^\lambda) \\ (x,x') \leftarrow \mathcal{A}(h)}} [h(x) = h(x') \wedge x \neq x'] \leq \mathsf{negl}(|\lambda|).$$

Remark 2.7. *The above definition of* CRHF *is sometimes referred to as a* private-coin CRHF *[HR04] since security is not guaranteed if the adversary sees the coins used by* Gen*. Nevertheless, for sake of conciseness we will sometimes avoid mentioning* Gen *explicitly and simply write* $h \leftarrow \mathcal{H}^{(\lambda)}$*.*

We will use the following syntax to describe Merkle tree commitments, which can be built from any CRHF family $\mathcal{H} = \{\mathcal{H}^{(\lambda)}\}_\lambda$. For each of the following algorithms, the input hash function h is drawn uniformly at random from $\mathcal{H}^{(\lambda)}$. For any $d \geq 1$, a Merkle tree commitment allows us to commit to a message $s \in \{0,1\}^m$ where $m := \lambda \cdot 2^d$. That is, we view s as 2^d blocks of λ bits.

- $\mathsf{MerkleCom}(h, s \in \{0,1\}^m)$. Write s as $(\ell_1 \| \ell_2 \| \ldots \| \ell_{2^d})$ where each $\ell_j \in \{0,1\}^\lambda$. Build a binary tree of hash evaluations, starting from the 2^d leaves $(\ell_1 \| \ell_2 \| \ldots \| \ell_{2^d})$. Output the root $\mathsf{com} \in \{0,1\}^\lambda$ of the resulting tree.

A commitment to s can be *locally opened* to the reveal the bits in the ith block by revealing the siblings along the root-to-ith-leaf path:

- $\mathsf{MerkleOpen}(h, s \in \{0,1\}^m, i \in [2^d])$. Write s as $(\ell_1 \| \ell_2 \| \ldots \| \ell_{2^d})$ where each $\ell_j \in \{0,1\}^\lambda$. Determine the path from ℓ_i to the root in the tree of hash evaluations under h, denoted $\{\widehat{c}_j\}_{j \in [d]}$ where $\widehat{c}_d = \ell_i$. For each $i \in [d]$, determine the sibling sib_i of $\widehat{c}_i$. Output $\mathsf{open} = \{(\widehat{c}_i, \mathsf{sib}_i, p_i)\}_{i \in [d]}$ where $p_i \in \{\mathsf{left}, \mathsf{right}\}$ denotes whether sib_i is a left or right sibling of $\widehat{c}_i$.

  For $I \subseteq [2^d]$ we define $\mathsf{MerkleOpen}(h, s \in \{0,1\}^m, I)$ as $(\mathsf{MerkleOpen}(h, s \in \{0,1\}^m, i))_{i \in I}$.

These openings can easily be verified by verifying the hash computations with h:

- $\mathsf{MerkleVer}(h, \mathsf{com}, \mathsf{open})$ first writes open as $\{(\widehat{c}_i, \mathsf{sib}_i, p_i)\}_{i \in [d]}$. Let $\widehat{c}_0 = \mathsf{com}$. For each $i \in [d]$, check that $h(\mathsf{sib}_i \| \widehat{c}_i) = \widehat{c}_{i-1}$ if $p_i = \mathsf{left}$ or that $h(\widehat{c}_i \| \mathsf{sib}_i) = \widehat{c}_{i-1}$ if $p_i = \mathsf{right}$. Output 1 (accept) if all checks pass, otherwise output 0.

2.2.2 Kilian's Protocol

While Kilian's original protocol relied on PCPs, a natural generalization to PCIPs was suggested by Ben-Sasson *et al.* [BCS16]. This extension proceeds by having the prover repeatedly commit to each of its oracles (rather than sending the entire oracle). At the end of the interaction, the verifier can specify which locations to open and the prover can use the Merkle tree structure to succinctly decommit to these specific locations.

Construction 2.8. *Let* $(\mathcal{P}, \mathcal{V})$ *be a public-coin* k*-round* PCIP *for* $\mathcal{L} \in$ NP*. Consider the following argument system for* $\mathcal{L}$*, denoted by* $(\mathcal{P}', \mathcal{V}') = \mathsf{Kilian}[(\mathcal{P}, \mathcal{V}), \mathcal{H}]$*, as described in Fig. 1, w.r.t. a* CRHF $\mathcal{H}$*.*

Theorem 2.9 (Kilian's protocol)**.** *If* (P, V) *is a* PCIP *and* $\mathcal{H}$ *is a* CRHF *family, then Construction 2.8 is a computationally sound argument system with negligible soundness error, communication complexity* $\mathsf{poly}(\lambda, \mathsf{log}|x|)$*. The verifier runs in time* $O(|x| \cdot \mathsf{poly}(\lambda, \mathsf{log}|x|))$ *and the prover runs in time* $\mathsf{poly}(|x|, |w|, \lambda)$*.*

Completeness follows from the correctness of the Merkle tree commitment scheme and the completeness of the PCIP for $\mathcal{L}$.

Soundness follows from the binding property of the commitment scheme and the soundness of the PCP. Note that the soundness of the PCIP was not enough on its own, as without committing to the PCIP proof strings, the prover could have engineered proof strings to make the verifier accept according to its queries.

Protocol 1: Kilian's protocol

Common input: $x, 1^\lambda$

Prover's auxiliary input: w

$\mathcal{V}'$ generates a hash function $h \leftarrow \mathcal{H}^{(\lambda)}$ and sends h to $\mathcal{P}'$.

for $j = 1, ..., k$ **do**

  $\mathcal{P}'$ sends $\mathsf{com}_j = \mathsf{MerkleCom}(h, \pi_j)$, where π_j is the message sent by $\mathcal{P}$ in $(\mathcal{P}(x, w, 1^\lambda), \mathcal{V}(x, 1^\lambda))$ on the j-th round.

  $\mathcal{V}'$ sends r_j, where r_j is the message sent by $\mathcal{V}$ on the j-th round of $(\mathcal{P}(x, w, 1^\lambda), \mathcal{V}(x, 1^\lambda))$.

  $\mathcal{P}'$ sends open_j to $\mathcal{V}'$ where $\mathsf{open}_j = \mathsf{MerkleOpen}(h, \pi_j, Q_j)$, where Q_j is the set of queries generated by $\mathcal{V}$ on round j with randomness r_j.

  $\mathcal{V}'$ computes $v_j \in \{0, 1\}$ where $v_j = \mathsf{MerkleVer}(h, \mathsf{com}_j, \mathsf{open}_j)$.

end

$\mathcal{V}'$ accepts if and only if $\bigwedge_j v_j = 1$ and $\mathcal{V}(x, \left\{b_1^{i_1}\right\}_{i_1}, ..., \left\{b_k^{i_k}\right\}_{i_k}) = 1$, where $\left\{b_j^{i_j}\right\}_{i_j}$ is the set result of the queries revealed in open_j.

Figure 1: Kilian's protocol.

2.3 Fiat-Shamir

2.3.1 The Fiat-Shamir Heuristic

The Fiat-Shamir heuristic [FS86] is a method for reducing the number of rounds in public-coin interactive proofs. Loosely speaking, the idea is that instead of having the verifier send their random coins, the prover uses a hash function in order to generate the verifier's randomness.

Definition 2.10 (Fiat-Shamir transform)**.** *Let* $\mathcal{H}_{\mathsf{FS}} = \left\{\mathcal{H}_{\mathsf{FS}}^{(\lambda)} : \{0,1\}^* \to \{0,1\}^*\right\}_{\lambda \in \mathbb{N}}$ *be a hash-function ensemble,* $\Pi = (\mathcal{P}, \mathcal{V})$ *be a public-coin protocol, and*

$$(\alpha_1, \beta_1, ..., \alpha_m, \beta_m)$$

be the set of exchanged messages between $(\mathcal{P}, \mathcal{V})$*, where* $\{\alpha_i\}_{i=1}^m$ *are messages sent by the prover and* $\{\beta_i\}_{i=1}^m$ *the messages sent by the verifier. The* Fiat-Shamir transform *of* Π*, denoted by* $(\mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}}) = \mathsf{FS}[\Pi, \mathcal{H}_{\mathsf{FS}}]$ *is defined in Fig. 2.*

Protocol 2: Fiat-Shamir transform

Common input: $x, 1^\lambda$

Prover's auxiliary input: w

$\mathcal{V}_{\mathsf{FS}}$ generates a key $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}^{(\lambda)}$ and sends it to $\mathcal{P}_{\mathsf{FS}}$.

$\mathcal{P}_{\mathsf{FS}}$ sends the following, all in a *single* message

$$\alpha_1,\ \beta_1 = h_{\mathsf{FS}}(\tau_1),\ \alpha_2,\ \beta_2 = h_{\mathsf{FS}}(\tau_2), ..., \alpha_m,\ \beta_m = h_{\mathsf{FS}}(\tau_m)$$

where $\tau_i = (\alpha_1 \| \beta_1 \| ... \| \alpha_i)$ is the transcript thus far.

$\mathcal{V}_{\mathsf{FS}}$ checks that $\forall i \in [m] : \beta_i = h_{\mathsf{FS}}(\tau_i)$, and accepts iff $\mathcal{V}(x, \alpha_1, \beta_1, ..., \alpha_m, \beta_m)$ accepts.

Figure 2: The Fiat-Shamir transform.

2.3.2 The FSKM Protocol

The FSKM protocol is obtained by applying the Fiat-Shamir transform to Kilian's protocol (or rather to its extension to PCIPs), to create succinct non-interactive argument systems for NP.

Recall that the FSKM protocol emulates Kilian's protocol, and replaces the verifier's randomness with the application of a FS hash function on the transcript thus far. Regardless of whether it is applied on a PCP or PCIP, the FSKM protocol is a two-round argument system.

Definition 2.11 (FSKM Protocol)**.** *Given a* PCIP Π*, a* CRHF *ensemble* $\mathcal{H}_{\mathsf{CRHF}}$*, and* FS *hash function ensemble* $\mathcal{H}_{\mathsf{FS}}$*, we define*

$$\mathsf{FSKM}[\Pi, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}] \triangleq \mathsf{FS}\left[\mathsf{Kilian}\left[\Pi, \mathcal{H}_{\mathsf{CRHF}}\right], \mathcal{H}_{\mathsf{FS}}\right].$$

3 An FSKM-Incompatible CRHF

In this section, we obtain our first main result by constructing a specific CRHF family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$, for which, loosely speaking, FSKM is not sound. Our CRHF will make use of a *publicly-verifiable succinct non-interactive argument of knowledge* (pv-SNARK) with an additional "unique proofs" property that we formalize in Section 3.1.1. For completeness, we start by providing some background on SNARKs.

3.1 Background on SNARKs

We first define the *universal relation* [BG08] relative to *random-access machines*.

Definition 3.1 (Universal Relation)**.** *The universal relation is the set* $\mathcal{R}_{\mathcal{U}}$ *of instance-witness pairs* $(y, w) = \Big((\langle M\rangle, x, t), w\Big)$*, where* $|y|, |w| \leq t$ *and* $\langle M\rangle$ *is the description of a random-access machine* M*, such that* M *accepts* (x, w) *after at most* t *steps. We denote by* $\mathcal{L}_{\mathcal{U}}$ *the universal language corresponding to* $\mathcal{R}_{\mathcal{U}}$*.*

We next define publicly-verifiable succinct non-interactive arguments of knowledge (pv-SNARKs), following [BCCT13]. The following definition is taken verbatim from Bitansky *et al.* [BCCT13], and for more in-depth discussion on SNARKs we refer the reader to [BCCT13].

Definition 3.2 (pv-SNARKs)**.** *A triple of algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$*, where* $\mathcal{G}$ *is probabilistic and* $\mathcal{V}$ *is deterministic, is a* pv-SNARK *if the following conditions are satisfied:*

- **Completeness:** *For every large enough security parameter* $\lambda \in \mathbb{N}$*, every time bound* $B \in \mathbb{N}$*, and every instance-witness pair* $(y, w) = ((\langle M\rangle, x, t), w) \in \mathcal{R}_{\mathcal{U}}$ *with* $t \leq B$*,*

$$\Pr\left[\mathcal{V}(\mathsf{crs}, y, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda, B) \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, y, w) \end{array}\right] = 1.$$

- **Adaptive Proof of Knowledge:** *For every polynomial-sized prover* $\mathcal{P}^*$ *there exists a polynomial-sized extractor* $\mathcal{E}_{\mathcal{P}^*}$ *such that for every auxiliary input* $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$*, every time bound* $B \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{crs}, y, \pi) = 1 \\ (y, w) \notin \mathcal{R}_{\mathcal{U}} \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda, B) \\ (y, \pi) \leftarrow \mathcal{P}^*(z, \mathsf{crs}) \\ w \leftarrow \mathcal{E}_{\mathcal{P}^*}(z, \mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

- **Efficiency:** *There exists a universal polynomial* p *such that, for every large enough security parameter* $\lambda \in \mathbb{N}$*, every time bound* $B \in \mathbb{N}$*, and every instance* $y = (\langle M\rangle, x, t)$ *with* $t \leq B$*,*
  - *the generator* $\mathcal{G}(1^\lambda, B)$ *runs in time* $\begin{cases} p(\lambda + B) & \text{for a preprocessing SNARK} \\ p(\lambda + \log(B)) & \text{for a fully-succinct SNARK} \end{cases}$,

– *the prover* $\mathcal{P}(\mathsf{crs}, y, w)$ *runs in time* $\begin{cases} p(\lambda + |M| + |x| + t + B) & \text{for a preprocessing SNARK} \\ p(\lambda + |M| + |x| + t + \log(B)) & \text{for a fully-succinct SNARK} \end{cases}$,

– *the verifier* $\mathcal{V}(\mathsf{crs}, y, \pi)$ *runs in time* $p(\lambda + |M| + |x| + \log(B))$,

– *and an honestly generated proof has size* $p(\lambda + \log(B))$.

Note that in a fully-succinct SNARK, we can remove the time bound by, e.g., setting $B = \lambda^{\log(\lambda)}$. Then $\mathcal{G}$ will run in time $p(\lambda)$, an honestly generated proof will have size $p(\lambda)$, and so on.

3.1.1 Computationally Unique SNARKs

In this work we introduce a new security property of SNARKs which we refer to as *computationally unique proofs* (which can be thought of as a particular computational variant of unambiguous proofs [RRR16]). The requirement here is that if a computationally bounded prover $\mathcal{P}$ can generate two valid proofs $\pi_1 \neq \pi_2$ for the same instance y, it must be possible to extract from $\mathcal{P}$ two distinct witnesses $w_1 \neq w_2$ for y.

Definition 3.3 (SNARKs with computationally unique proofs)**.** *A* SNARK *with computationally unique proofs is defined as in Definition 3.2, but with one additional requirement:*

- **Computationally Unique Proofs:** *For every polynomial-sized adversary* $\mathcal{A}^*$*, there exists a polynomial-sized "extractor"* $\mathcal{E}_{\mathcal{A}^*}$ *such that for every auxiliary input* $z \in \{0,1\}^{\mathsf{poly}(k)}$*, every time bound* $B \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{crs}, y, \pi_1) = 1 \\ \mathcal{V}(\mathsf{crs}, y, \pi_2) = 1 \\ \pi_1 \neq \pi_2 \\ (y, w_1) \notin \mathcal{R}_{\mathcal{U}} \vee (y, w_2) \notin \mathcal{R}_{\mathcal{U}} \vee w_1 = w_2 \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda, B) \\ (y, \pi_1, \pi_2) \leftarrow \mathcal{A}^*(z, \mathsf{crs}) \\ (w_1, w_2) \leftarrow \mathcal{E}_{\mathcal{A}^*}(z, \mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

In Appendix A, we prove that a preprocessing pv-SNARK constructed in Bitansky *et al.* [BCI+13] from a knowledge of exponent assumption satisfies our notion of computationally unique proofs. We then show that the generic transformation of [BCCT13] from a preprocessing pv-SNARK to fully-succinct pv-SNARK maintains the computationally unique proofs property. Thus, we obtain a fully-succinct pv-SNARK with computationally unique proofs from a knowledge of exponent assumption (and additionally the existence of exponentially-secure one-way functions to address a subtlety in the definition of adaptive proof of knowledge).

3.2 An FSKM-Incompatible CRHF

To formally state our result, we first define a trivial PCP-based 2-message protocol — a protocol that, intuitively, should not be sound. Jumping ahead, at a high level, our main result shows that there exists a collision-resistant hash family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that for any Π_{PCP} and any $\mathcal{H}_{\mathsf{FS}}$, the corresponding FSKM protocol is *no more secure* than the corresponding trivial protocol.

The first message of the trivial protocol will be a random string r drawn from some distribution $\mathcal{S}$, which will serve as $\mathcal{V}_{\mathsf{PCP}}$'s randomness. The prover takes r as input and outputs a PCP proof π that will be then verified by $\mathcal{V}_{\mathsf{PCP}}$ using randomness r. Intuitively, since a cheating prover is aware of the verifier's randomness, it can answer queries *adaptively*, so we do not expect the trivial protocol to be sound.

Suppose we have some PCP $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$. It will be convenient for us to split the verifier $\mathcal{V}_{\mathsf{PCP}}$ into two algorithms: $\mathcal{V}^{(0)}_{\mathsf{PCP}}$ (which outputs the set of query locations) and $\mathcal{V}^{(1)}_{\mathsf{PCP}}$ (which decides whether or not to accept after seeing the prover responses).

Construction 3.4 (Trivial Protocol)**.** *Let* $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{S}] = (\mathcal{P}_{\mathsf{Trivial}}, \mathcal{V}_{\mathsf{Trivial}})$ *be the following 2-message protocol, for some* PCP $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}^{(0)}_{\mathsf{PCP}}, \mathcal{V}^{(1)}_{\mathsf{PCP}})$ *for a language* $\mathcal{L}$*, and some sampling algorithm* $\mathcal{S}$*. The verifier* $\mathcal{V}_{\mathsf{Trivial}}$ *generates a random string* r *from* $\mathcal{S}$ *and sends* r *to the prover. The prover* $\mathcal{P}_{\mathsf{Trivial}}$*, on input* (x, w, r) *for* $x \in \mathcal{L}$ *runs* $\mathcal{V}^{(0)}_{\mathsf{PCP}}(x; r)$ *to obtain a set of query locations* $\{q_i\}_i$*.* $\mathcal{P}_{\mathsf{Trivial}}$ *then computes* $b_i \leftarrow \mathcal{P}_{\mathsf{PCP}}(x, w, q_i)$ *for each* i *and sends* $\{b_i\}$ *to the verifier* $\mathcal{V}_{\mathsf{Trivial}}$*. The verifier* $\mathcal{V}_{\mathsf{Trivial}}$ *computes* $\mathcal{V}^{(0)}_{\mathsf{PCP}}(x; r) = \{q_i\}_i$ *and accepts if and only if* $\mathcal{V}^{(1)}_{\mathsf{PCP}}\big(x, \{(q_i, b_i)\}_i; r\big)$ *accepts.*

In what follows, we will sometimes view $\mathcal{S}$ as an algorithm that explicitly takes its randomness u as input, and outputs $r = \mathcal{S}(u)$.

For non-contrived choices of a PCP and sampling algorithm we do not expect Construction 3.4 to be sound. For example, consider Håstad's PCP [Hås01] in which the verifier queries 3 bits of the proof and checks whether their parity is some known fixed value b. Soundness of the trivial protocol can now be violated by having the prover send the answers $(0, 0, b)$.[5]

Recall (see Section 2.3.2) that $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ denotes the 2-message argument that results from applying the Fiat-Shamir transform with hash function ensemble $\mathcal{H}_{\mathsf{FS}}$ to $\mathsf{Kilian}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}]$.

The main theorem of this section is the following.

Theorem 3.5. *Assume the existence of a fully-succinct* pv-SNARK *with computationally unique proofs, where honestly generated proofs have size at most $p(\lambda)$, and collision resistant hash functions. Define $m := 2\lambda \cdot p(\lambda)$. Then, there exists a collision resistant hash family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}} = \left\{\widetilde{\mathcal{H}}^{(\lambda)}_{\mathsf{CRHF}} : \{0,1\}^{2m} \to \{0,1\}^m\right\}_{\lambda \in \mathbb{N}}$ such that for any* PCP $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$ *with proof length at most 2^λ, and any hash function ensemble $\mathcal{H}_{\mathsf{FS}}$, if $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is computationally sound, then $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{FS}}]$ is computationally sound.*

We believe that for natural choices of PCPs, the trivial protocol will not be sound which, by Theorem 3.5, means that the corresponding FSKM protocol is not sound. However, actually proving that the trivial protocol is not sound seems to be difficult in case the sampling algorithm generates a peculiar distribution of random strings.[6]

Nevertheless, we can exhibit a *specific* (trivial) PCP for which the trivial protocol is provably not sound. The immediate implication is that for every FS hash function, there exists a PCP and a bounded size CRHF for which soundness of the corresponding FSKM is violated. This is formalized in the following corollary.

Corollary 3.6. *Assume the existence of a fully-succinct* pv-SNARK *with computationally unique proofs. There exists a language $\mathcal{L} \in \mathsf{NP}$, a* PCP *for $\mathcal{L}$ (with $\mathsf{polylog}(\lambda)$ query complexity) and a fixed polynomial $p(\cdot)$ such that for all efficiently computable hash function ensembles $\mathcal{H}_{\mathsf{FS}}$, there exists a* CRHF *ensemble $\mathcal{H}_{\mathsf{CRHF}} = \left\{\mathcal{H}^{(\lambda)}_{\mathsf{CRHF}} : \{0,1\}^{2s(\lambda)} \to \{0,1\}^{s(\lambda)}\right\}_{\lambda \in \mathbb{N}}$ with $s(\lambda) \leq p(\lambda)$, such that $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound.*

We first prove Corollary 3.6 and then go back to the main part — proving Theorem 3.5.

Proof of Corollary 3.6. We exhibit a contrived PCP $\Pi_\emptyset$ for the empty language for which the statement holds. Specifically, consider a PCP verifier that samples at random $r \in \{0,1\}^{\log^2(\lambda)}$ and checks whether the $\log^2(\lambda)$-long prefix of the proof is exactly equal to r (by making $\log^2(\lambda)$ queries).

Completeness holds vacuously, and this PCP is sound since the proof must be specified before r was sampled. However, the protocol $\mathsf{Trivial}[\Pi_\emptyset, \mathcal{H}_{\mathsf{FS}}]$ for any sampler $\mathcal{H}_{\mathsf{FS}}$ is clearly not sound, since the cheating prover receives the verifier randomness r as input and simply returns r as its proof. □

In Section 5, we actually give a secure instantiation of FSKM for a variant of the PCP $\Pi_\emptyset$ that was used to prove Corollary 4.3. This does not contradict our impossibility, which only rules out security of FSKM with a *generic* CRHF. In particular, our instantiation requires the collision-resistant hash function to also be somewhere statistically binding [HW15a]. Unfortunately, we do not know how to instantiate the FSKM protocol to construct an argument scheme for a non-trivial language.

The remainder of this section will be devoted to a proof of Theorem 3.5. As described in Section 1.4, our strategy centers on a carefully-designed hash function family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$, built using two CRHFs and a fully-succinct pv-SNARK with computationally unique proofs (Definition 3.3). The result then follows immediately from combining Lemma 3.7, which states that $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ is a CRHF family, and Lemma 3.8, which establishes the soundness implication.

[5] Note that Håstad's PCP only has constant soundness. Nevertheless, the attack can be generalized to the sequential repetition of Håstad's PCP as long as the sampler $\mathcal{S}$ generates random query sets.

[6] One would assume that a random choice of FS hash function from the collection would produce a uniformly random string for the verifier. However, since we want to deal with *arbitrary* candidate FS hash functions, we cannot assume that this is the case.

3.3 CRHF Construction

Throughout this construction we will use the notation from the statement of Theorem 3.5; recall that $p(\lambda)$ is a bound on the proof size of our pv-SNARK, $m := 2\lambda \cdot p(\lambda)$, and $c > 0$ is an arbitrary constant independent of $p(\lambda)$.

We prove Theorem 3.5 by carefully constructing a CRHF family $\widetilde{\mathcal{H}}_{\mathsf{CRHF}} = \{\widetilde{\mathcal{H}}^{(\lambda)}_{\mathsf{CRHF}} : \{0,1\}^{2m} \to \{0,1\}^m\}_{\lambda \in \mathbb{N}}$. Our construction requires the following:

- A fully-succinct pv-SNARK $\mathcal{S} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ with computationally unique proofs (Definition 3.3), where honestly generated proofs have size exactly $p(\lambda)$ (assume that shorter proofs are appropriately padded with zeros).
- A CRHF family $\mathcal{H}_{\mathsf{rt}} = \{\mathcal{H}^{(\lambda)}_{\mathsf{rt}} : \{0,1\}^* \to \{0,1\}^{m/2-\lambda-\log(\lambda)-2}\}_{\lambda \in \mathbb{N}}$.
- A CRHF family $\mathcal{H}_{\mathsf{tree}} = \{\mathcal{H}^{(\lambda)}_{\mathsf{tree}} : \{0,1\}^{2m} \to \{0,1\}^{m-2}\}_{\lambda \in \mathbb{N}}$.

Construction Overview. Before giving the full details of the construction, we continue the discussion from Section 1.4.1 and explain the various aspects of our CRHF. Essentially, our goal is to design a CRHF $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that for any PCP $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$ and $\mathcal{H}_{\mathsf{FS}}$, an adversary $\mathcal{A}_{\mathsf{Trivial}}$ that breaks the soundness of $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{FS}}]$ enables a cheating prover $\mathcal{P}^*$ to break the soundness of $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$. Recall that in the $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{S}]$ protocol, $\mathcal{V}_{\mathsf{PCP}}$'s randomness r is fixed to be the output of sampling algorithm $\mathcal{S}$. Letting $\mathcal{S}$ explicitly take its randomness u as input, we can write $r = \mathcal{S}(u)$. In order to break soundness, the adversary $\mathcal{A}_{\mathsf{Trivial}}$ must produce a statement x and a series of answer bits $\{b_i\}_i$ such that $\mathcal{V}^{(0)}_{\mathsf{PCP}}(x; r) = \{q_i\}_i$ and $\mathcal{V}^{(1)}_{\mathsf{PCP}}(x, \{(q_i, b_i)\}_i; r)$ accepts. So, in the $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{FS}}]$, protocol, the verifier's randomness is fixed to be $h_{\mathsf{FS}}(u)$, for a uniformly random u, and $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}$. On the other hand, in the FSKM protocol, $\mathcal{V}_{\mathsf{PCP}}$'s randomness is computed by applying h_{FS} to the prover's Merkle hash, which we denote by rt.

Thus, in order to enable $\mathcal{P}^*$ to make use of $\mathcal{A}_{\mathsf{Trivial}}$'s functionality, we would like to design $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ in such a way that allows $\mathcal{P}^*$ to produce a uniformly random commitment rt, which it can then later open to an arbitrary PCP proof string π of its choice. If this were possible, $\mathcal{P}^*$ could feed $r = h_{\mathsf{FS}}(\mathsf{rt})$ to $\mathcal{A}_{\mathsf{Trivial}}$, and $\mathcal{A}_{\mathsf{Trivial}}$ will return some x and $\{b_i\}_i$. The prover $\mathcal{P}^*$ could then compute query locations $\{q_i\}_i = \mathcal{V}_{\mathsf{PCP}}(x; r)$ with the guarantee that $\mathcal{V}_{\mathsf{PCP}}(x, \{(q_i, b_i)\}_i; r)$ will accept. Thus, to violate the soundness of $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$, we would only need that rt is a valid commitment to the proof string π that contains the bit b_i at location q_i, for each i (and is 0 elsewhere). We denote this string by $\pi^{\{(q_i,b_i)\}_i}$.

Unfortunately, designing $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ such that rt will be a valid commitment to *any* such $\pi^{\{(q_i,b_i)\}_i}$ will clearly violate collision resistance. Instead, we will determine the value of the cheating prover's commitment based on the Fiat-Shamir hash function h_{FS}. In particular, we will hash its description $\langle h_{\mathsf{FS}} \rangle$, using a collision resistant hash function h_{rt},[7] to obtain a string α. We then form the commitment rt by XORing α with a uniformly random string $\$_{\mathsf{rt}}$ that will be fixed in the hash key of $\widetilde{h}_{\mathsf{CRHF}}$. That is, we set $\mathsf{rt} = \$_{\mathsf{rt}} \oplus h_{\mathsf{rt}}(\langle h_{\mathsf{FS}} \rangle) = \$_{\mathsf{rt}} \oplus \alpha$.

We allow $\mathcal{P}^*$ to produce rt as a valid commitment to the proof string $\pi^{\{(q_i,b_i)\}_i}$ by augmenting $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ with the ability to recognize and behave differently on "special" inputs, which will correspond to the right-most path of hash computations in $\mathcal{P}^*$'s Merkle tree. The rest of the tree will hash $\pi^{\{(q_i,b_i)\}_i}$ with standard Merkle hashing using some collision resistant hash function h_{tree}. However, the special path will allow $\mathcal{P}^*$ to "bubble up" the value of α, from a leaf up to the root, at which point it triggers $\widetilde{h}_{\mathsf{CRHF}}$ to output the root $\$_{\mathsf{rt}} \oplus \alpha$. The input to a hash computation along this path consists of two parts $(\mathsf{left}, \mathsf{right})$, where left corresponds to the Merkle hash of (a substring of) $\pi^{\{(q_i,b_i)\}_i}$, and right contains α and some other information. This additional information includes a depth parameter d and a succinct proof of knowledge of some h_{FS} such that $h_{\mathsf{FS}}(\$_{\mathsf{rt}} \oplus \alpha) = r$, $\mathcal{A}_{\mathsf{Trivial}}(r) = (x, \{b_i\}_i)$, $\mathcal{V}^{(0)}_{\mathsf{PCP}}(x; r) = \{q_i\}_i$, and $\mathcal{V}^{(1)}_{\mathsf{PCP}}(x, \{(q_i, b_i)\}_i; r)$ accepts. At a high level, collision resistance of our construction follows from the fact that as long as the SNARK has computationally

[7] Looking ahead, the function h_{rt} will play a crucial role in the proof of the collision resistance of $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$, as finding certain types of collisions for $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ will imply the ability to find collisions for $\mathcal{H}_{\mathsf{rt}}$.

unique proofs, finding a collision among "special" inputs implies the knowledge of two different $\langle h_{\mathsf{FS}}^{(1)} \rangle, \langle h_{\mathsf{FS}}^{(2)} \rangle$ that hash to the same α under h_{rt} (which we had assumed to be collision resistant).

In order to clearly differentiate the hash outputs (which will be useful for showing collision resistance), we introduce three distinct 2-bit prefixes $\mathsf{pre}_{\mathsf{rt}}, \mathsf{pre}_{\mathsf{path}}$, and $\mathsf{pre}_{\mathsf{tree}}$, where $\mathsf{pre}_{\mathsf{rt}}$ prefixes the special root computed by a cheating prover, $\mathsf{pre}_{\mathsf{path}}$ prefixes any node on the right-most special path, and $\mathsf{pre}_{\mathsf{tree}}$ prefixes any internal node in the rest of the Merkle tree computed by h_{tree}. We will also draw a uniform $\$_{\mathsf{path}}$ to include in special path inputs, in order to ensure that with overwhelming probability, non-special inputs are not accidentally parsed as special inputs. Finally, we include a set of Merkle commitments under h_{tree} to all zero strings of varying lengths, which will be useful for efficiently generating particular nodes in the Merkle tree of $\pi^{\{(q_i,b_i)\}_i}$.

3.4 CRHF Key Generation.

We sample a hash function $\widetilde{h}_{\mathsf{CRHF}} \leftarrow \widetilde{\mathcal{H}}_{\mathsf{CRHF}}^{(\lambda)}$ as follows.

1. Sample uniformly random $\$_{\mathsf{rt}} \leftarrow \{0,1\}^m$.
   - Let $\mathsf{pre}_{\mathsf{rt}}$ denote the first two bits of $\$_{\mathsf{rt}}$.
   - Define $\mathsf{pre}_{\mathsf{tree}} \coloneqq \mathsf{pre}_{\mathsf{rt}} \oplus 10$ and $\mathsf{pre}_{\mathsf{path}} \coloneqq \mathsf{pre}_{\mathsf{rt}} \oplus 01$.
2. Sample uniformly random $\$_{\mathsf{path}} \leftarrow \{0,1\}^\lambda$.
3. Sample $h_{\mathsf{rt}} \leftarrow \mathcal{H}_{\mathsf{rt}}^{(\lambda)}$ and $h_{\mathsf{tree}} \leftarrow \mathcal{H}_{\mathsf{tree}}^{(\lambda)}$.
4. Define $h'_{\mathsf{tree}} : \{0,1\}^{2m} \to \{0,1\}^m$ such that $h'_{\mathsf{tree}}(x) = (\mathsf{pre}_{\mathsf{tree}} \| h_{\mathsf{tree}}(x))$.
5. Sample $\mathsf{crs} \leftarrow \mathcal{G}(1^\lambda)$.
6. For each $j \in [\lambda]$, compute $\mathsf{com}_j^{(\mathsf{zero})} = \mathsf{MerkleCom}(h'_{\mathsf{tree}}, 0^{m \cdot 2^j})$.
7. Output

$$(\$_{\mathsf{rt}}, \$_{\mathsf{path}}, \mathsf{pre}_{\mathsf{tree}}, \mathsf{pre}_{\mathsf{path}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}, \mathsf{crs}, \{\mathsf{com}_j^{(\mathsf{zero})}\}_{j \in [\lambda]}),$$

   as the description (hash key) of $\widetilde{h}_{\mathsf{CRHF}}$.

3.4.1 CRHF Evaluation

Before we describe how to compute $\widetilde{h}_{\mathsf{CRHF}}(x)$, we need to introduce some specialized notation and definitions.

Notation. We will assume without loss of generality that $m = 2^{m'}$ is a power of 2, and that the PCP Π_{PCP} in fact has proof length bounded by $2^\lambda - m$. Throughout, $\{(q_i, b_i)\}_i$ will denote a set of (index, bit) pairs (representing PCP query/response pairs) where for each i, $q_i \in [2^\lambda - m]$ and $b_i \in \{0,1\}$. We assume without loss of generality that no index appears more than once.

For any set $\{(q_i, b_i)\}_i$ satisfying these conditions, we let $\pi^{\{(q_i,b_i)\}_i}$ denote the length $2^\lambda - m$ bitstring defined bit-wise for each $j \in [2^\lambda - m]$ as:

$$(\pi^{\{(q_i,b_i)\}_i})_j \coloneqq \begin{cases} b & \text{if } (j,b) \in \{(q_i,b_i)\}_i, \\ 0 & \text{else.} \end{cases}$$

In other words, $\pi^{\{(q_i,b_i)\}_i}$ is a PCP proof string that consists of the responses in $\{(q_i,b_i)\}_i$ and 0s everywhere else.

We divide $\pi^{\{(q_i,b_i)\}_i}$ into $2^{\lambda - m'} - 1$ words ℓ_k of $m = 2^{m'}$ bits each, i.e. $\pi^{\{(q_i,b_i)\}_i} = (\ell_1 \| \ell_2 \| \ldots \| \ell_{2^{\lambda-m'}-1})$, where each word ℓ_k is in $\{0,1\}^m$. Next, group the words as follows. The first $2^{\lambda - m' - 1}$ words will form the

first block L_1, the next $2^{\lambda-m'-2}$ words will form the second block L_2, and so on until the last block only consists of only 1 word. We can now write $\pi^{\{(q_i,b_i)\}_i}$ as

$$\Big(L_1\Big|\Big|\ldots\Big|\Big|L_{\lambda-m'}\Big) \coloneqq \Big((\ell_1\|\ldots\|\ell_{2^{\lambda-m'-1}})\Big|\Big|(\ell_{2^{\lambda-m'-1}+1}\|\ldots\|\ell_{2^{\lambda-m'-1}+2^{\lambda-m'-2}})\Big|\Big|\ldots\Big|\Big|(\ell_{2^{\lambda-m'}-1})\Big),$$

where the jth block L_j is exactly twice the length of the $(j+1)$th block L_{j+1}.

We define the helper functions $t(q)$ and $s(q)$ for any $q \in [2^\lambda - m]$ so that the qth bit in $\pi^{\{(q_i,b_i)\}_i}$ is the $t(q)$th bit in block $L_{s(q)}$.

Now define

- $\mathsf{block\text{-}com}(h'_{\mathsf{tree}}, \{(q_i,b_i)\}_i, j) \coloneqq \mathsf{MerkleCom}(h'_{\mathsf{tree}}, L_j)$, and
- $\mathsf{block\text{-}open}(h'_{\mathsf{tree}}, \{(q_i,b_i)\}_i, i) \coloneqq \mathsf{MerkleOpen}(h'_{\mathsf{tree}}, L_{s(q_i)}, t(q_i))$.

Note that given $\{\mathsf{com}_j^{(\mathsf{zero})}\}_{j\in[\lambda]}$, if $|\{(q_i,b_i)\}_i| = \mathsf{poly}(\lambda)$, it is easy to compute $\mathsf{block\text{-}com}(h'_{\mathsf{tree}}, \{(q_i,b_i)\}_i, j)$ and $\mathsf{block\text{-}open}(h'_{\mathsf{tree}}, \{(q_i,b_i)\}_i, i)$ in time $\mathsf{poly}(\lambda)$.

Language. We also define a language $\mathcal{L}_{\$_{\mathsf{rt}},h_{\mathsf{rt}},h'_{\mathsf{tree}}}$ based on $\$_{\mathsf{rt}}, h_{\mathsf{rt}}$, and h'_{tree} (all given in the CRHF description $\widetilde{h}_{\mathsf{CRHF}}$). Throughout, we use $\mathsf{bit}_\lambda(j)$ to denote the $\log(\lambda)$-bit binary representation of an integer j.

$\mathcal{L}_{\$_{\mathsf{rt}},h_{\mathsf{rt}},h'_{\mathsf{tree}}}$ will be defined by relation $\mathcal{R}_{\$_{\mathsf{rt}},h_{\mathsf{rt}},h'_{\mathsf{tree}}}$, which consists of all (instance, witness) pairs of the form

$$\Big((\alpha\|\mathsf{bit}_\lambda(j)\|\mathsf{sib}), (\langle h_{\mathsf{FS}}\rangle\|\langle\mathcal{A}_{\mathsf{Trivial}}\rangle\|\langle\mathcal{V}_{\mathsf{PCP}}\rangle)\Big),$$

which satisfy all of the following conditions:

1. $\langle h_{\mathsf{FS}}\rangle$ and $\langle\mathcal{A}_{\mathsf{Trivial}}\rangle$ can be parsed as descriptions of the (deterministic) circuits h_{FS} and $\mathcal{A}_{\mathsf{Trivial}}$. When used in the proof of Lemma 3.7, h_{FS} will correspond to the Fiat-Shamir hash function $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}$, and $\mathcal{A}_{\mathsf{Trivial}}$ will correspond to the adversary breaking the soundness of $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{FS}}]$.
2. $\langle\mathcal{V}_{\mathsf{PCP}}\rangle$ can be parsed as the description of a two-part PCP verifier $\mathcal{V}^{(0)}_{\mathsf{PCP}}, \mathcal{V}^{(1)}_{\mathsf{PCP}}$, where $\mathcal{V}^{(0)}_{\mathsf{PCP}}$ outputs a set of query locations, and $\mathcal{V}^{(1)}_{\mathsf{PCP}}$ takes the query responses and outputs a bit indicating accept/reject (see the discussion in Section 2.1.2).
3. $\alpha = h_{\mathsf{rt}}\big(\langle h_{\mathsf{FS}}\rangle\|\langle\mathcal{A}_{\mathsf{Trivial}}\rangle\|\langle\mathcal{V}_{\mathsf{PCP}}\rangle\big)$.
4. $\mathsf{sib} = \mathsf{block\text{-}com}(h'_{\mathsf{tree}}, \{(q_i,b_i)\}_i, j)$ where j is the integer represented by $\mathsf{bit}_\lambda(j)$, and for

$$r \coloneqq h_{\mathsf{FS}}(\mathsf{rt}) \text{ where } \mathsf{rt} \coloneqq \$_{\mathsf{rt}} \oplus (0^{\lambda+\log(\lambda)+2}\|\alpha\|0^{m/2}),$$

$\{(q_i,b_i)\}_i$ satisfies the requirements

- $\mathcal{A}_{\mathsf{Trivial}}(r) = x, \{b_i\}_i$,
- $\mathcal{V}^{(0)}_{\mathsf{PCP}}(x;r) = \{q_i\}_i$,
- $\mathcal{V}^{(1)}_{\mathsf{PCP}}(x, \{b_i\}_i; r) = 1$.

Since α is the result of applying the CRHF h_{rt} to the witness $\langle h_{FS}\rangle\|\langle\mathcal{A}_{\mathsf{Trivial}}\rangle\|\langle\mathcal{V}_{\mathsf{PCP}}\rangle$, an efficient adversary will only be able to find a single witness corresponding to any given α. The string α fully determines rt and r, which also determines $\{(q_i,b_i)\}_i$ (where $\{q_i\}$ is the set of PCP indices that $\mathcal{V}_{\mathsf{PCP}}$ would check when running on randomness r, and $\{b_i\}_i$ are the PCP responses output by $\mathcal{A}_{\mathsf{Trivial}}$ which cause $\mathcal{V}_{\mathsf{PCP}}$ to accept). This specifies a unique "cheating" PCP proof string consisting of 0's in almost every position, except with b_i's in indices corresponding to the q_i's. sib then corresponds to the label of the off-path node at level j for

the rightmost root-to-leaf path in the Merkle tree, and is obtained by applying h'_{tree} to this cheating PCP proof string.

In the proof of Lemma 3.8, we will rely on the fact that for any j and any witness $\langle h_{FS}\rangle \| \langle \mathcal{A}_{\mathsf{Trivial}}\rangle \| \langle \mathcal{V}_{\mathsf{PCP}}\rangle$, a cheating prover can efficiently compute α, sib such that $((\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}), (\langle h_{\mathsf{FS}}\rangle \| \langle \mathcal{A}_{\mathsf{Trivial}}\rangle \| \langle \mathcal{V}_{\mathsf{PCP}}\rangle))$ is in the relation (by simply applying $h_{\mathsf{FS}}, \mathcal{A}_{\mathsf{Trivial}}, \mathcal{V}^{(0)}_{\mathsf{PCP}}, h'_{\mathsf{tree}}$ in the specified way). In the proof of Lemma 3.7, we use the fact that for each (α, j) pair, an efficient adversary can only find one (sib, w) pair such that $((\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}), w)$ is in the relation (due to the collision-resistance of h_{rt}).

Hash Computation. Parse input $x \in \{0,1\}^{2m}$ as

$$\Big(\mathsf{sib} \;\Big\|\; (\mathsf{pre} \| \$ \| \mathsf{bit}_\lambda(j) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_j \| z)\Big),$$

where

- $\mathsf{sib} \in \{0,1\}^m$,
- $(\mathsf{pre} \| \$ \| \mathsf{bit}_\lambda(j) \| \alpha) \in \{0,1\}^{m/2}$ ($\mathsf{pre} \in \{0,1\}^2$, $\$ \in \{0,1\}^\lambda$, $\mathsf{bit}_\lambda(j) \in \{0,1\}^{\log(\lambda)}$, and $\alpha \in \{0,1\}^{m/2-\lambda-\log(\lambda)-2}$),
- and $(\pi_1 \| \ldots \| \pi_j \| z) \in \{0,1\}^{m/2}$ ($\pi_i \in \{0,1\}^{p(\lambda)}$ and $z \in \{0,1\}^{m/2-jp(\lambda)}$).

If $\mathsf{pre} = \mathsf{pre}_{\mathsf{path}}, \$ = \$_{\mathsf{path}}$, and $z = 0^{m/2-jp(\lambda)}$, run the SNARK verifier for language $\mathcal{L}_{\$_{\mathsf{rt}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}}$ on $(\tau, (\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}), \pi_j)$. If $j \geq 1$ and the verifier accepts, then

- if $j \geq 2$, output

$$\Big((\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(j-1) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_{j-1} \| 0^{m/2-(j-1)p(\lambda)})\Big) \in \{0,1\}^m,$$

- and if $j = 1$, output

$$\Big(\$_{\mathsf{rt}} \oplus (0^{\lambda+\log(\lambda)+2} \| \alpha \| 0^{m/2})\Big) \in \{0,1\}^m.$$

Otherwise, if the input does not parse as above, or the verifier does not accept, output $h'_{\mathsf{tree}}(x)$.

3.5 Proof of Theorem 3.5

Lemma 3.7. *Assuming $\mathcal{H}_{\mathsf{rt}}$ and $\mathcal{H}_{\mathsf{tree}}$ are CRHF families and that $\mathcal{S} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a fully-succinct SNARK with computationally unique proofs, then the above construction of $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$ is a CRHF family.*

Proof. Every hash function output begins with $\mathsf{pre}_{\mathsf{tree}}$, $\mathsf{pre}_{\mathsf{path}}$, or $\mathsf{pre}_{\mathsf{rt}}$. We show separately that an adversary cannot find a collision for an image with each of these three prefixes.

First, an adversary that finds a collision whose image begins with $\mathsf{pre}_{\mathsf{tree}}$ immediately breaks the collision-resistance of $\mathcal{H}_{\mathsf{tree}}$.

Next, consider the case where the adversary produces a collision whose image begins with $\mathsf{pre}_{\mathsf{path}}$. In full generality, the image point can be written as,

$$\Big((\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(j-1) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_{j-1} \| 0^{m/2-(j-1)p(\lambda)})\Big),$$

for some $j \in [\lambda]$. The collision must therefore be of the form

$$\Big(\mathsf{sib}^{(1)} \;\Big\|\; (\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(j) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_{j-1} \| \pi_j^{(1)} \| 0^{m/2-jp(\lambda)})\Big),$$
$$\Big(\mathsf{sib}^{(2)} \;\Big\|\; (\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(j) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_{j-1} \| \pi_j^{(2)} \| 0^{m/2-jp(\lambda)})\Big),$$

where $(\mathsf{sib}^{(1)}, \pi_j^{(1)}) \neq (\mathsf{sib}^{(2)}, \pi_j^{(2)})$, and the SNARK verifier accepts $\pi_j^{(1)}$ as a proof that $(\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}^{(1)}) \in \mathcal{L}_{\$_{\mathsf{rt}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}}$ and accepts $\pi_j^{(2)}$ as a proof that $(\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}^{(2)}) \in \mathcal{L}_{\$_{\mathsf{rt}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}}$.

Suppose that $\mathsf{sib}^{(1)} \neq \mathsf{sib}^{(2)}$. We can view the adversary as a prover $\mathcal{P}^*$ that produces proofs for two different instances $(\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}^{(1)})$ and $(\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}^{(2)})$, and can thus run the SNARK extractor $\mathcal{E}_{\mathcal{P}^*}$ twice to produce two valid witnesses $w_1 = (\langle h_{\mathsf{FS}}^{(1)} \rangle, \langle \mathcal{A}_{\mathsf{Trivial}}^{(1)} \rangle, \langle \mathcal{V}_{\mathsf{PCP}}^{(1)} \rangle)$ and $w_2 = (\langle h_{\mathsf{FS}}^{(2)} \rangle, \langle \mathcal{A}_{\mathsf{Trivial}}^{(2)} \rangle, \langle \mathcal{V}_{\mathsf{PCP}}^{(2)} \rangle)$. It must be the case that $w_1 \neq w_2$ since given α, j, and a single witness $w = (\langle h_{\mathsf{FS}} \rangle, \langle \mathcal{A}_{\mathsf{Trivial}} \rangle, \langle \mathcal{V}_{\mathsf{PCP}} \rangle)$, there is only one possible sib that would satisfy the relation $\mathcal{R}_{\$_{\mathsf{rt}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}}$ (by definition). Thus $\mathcal{E}_{\mathcal{P}^*}$ produces $w_1 \neq w_2$ such that $h_{\mathsf{rt}}(w_1) = \alpha = h_{\mathsf{rt}}(w_2)$, breaking the collision resistance of $\mathcal{H}_{\mathsf{rt}}$.

We must therefore have $\mathsf{sib}^{(1)} = \mathsf{sib}^{(2)}$ but $\pi_j^{(1)} \neq \pi_j^{(2)}$. But then the adversary can be viewed as a prover $\mathcal{P}^*$ that produces two different accepting proofs for the same statement. Since $\mathcal{S}$ has computationally unique proofs, $\mathcal{E}_{\mathcal{P}^*}$ also produces $w_1 \neq w_2$ such that $h_{\mathsf{rt}}(w_1) = \alpha = h_{\mathsf{rt}}(w_2)$, which breaks the collision resistance of $\mathcal{H}_{\mathsf{rt}}$.

The last case to consider is when the adversary produces a collision whose image begins with $\mathsf{pre}_{\mathsf{rt}}$. The collision must be of the form

$$\Big(\mathsf{sib}^{(1)} \;\Big\|\; (\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(1) \| \alpha) \;\Big\|\; (\pi^{(1)} \| 0^{m/2 - p(\lambda)})\Big),$$
$$\Big(\mathsf{sib}^{(2)} \;\Big\|\; (\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(1) \| \alpha) \;\Big\|\; (\pi^{(2)} \| 0^{m/2 - p(\lambda)})\Big),$$

where $(\mathsf{sib}^{(1)}, \pi^{(1)}) \neq (\mathsf{sib}^{(2)}, \pi^{(2)})$. Exactly the same argument strategy used for the previous case applies here. □

Lemma 3.8. *For any* PCP $\Pi_{\mathsf{PCP}} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$ *and* $\mathcal{H}_{\mathsf{FS}}$*, if* $\mathsf{Trivial}[\Pi_{\mathsf{PCP}}, \mathcal{H}_{\mathsf{FS}}]$ *is not sound, then* $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ *is not sound.*

Proof. By assumption, there exists a deterministic (by fixing the best randomness) poly-time $\mathcal{A}_{\mathsf{Trivial}}$, such that on input $r = h_{\mathsf{FS}}(u)$ for $h_{\mathsf{FS}} \leftarrow \mathcal{H}_{\mathsf{FS}}$ and uniform u, $\mathcal{A}$ outputs $x, \{b_i\}_i$ such that $\mathcal{V}_{\mathsf{PCP}}^{(0)}(x; r) = \{q_i\}_i$, and $\mathcal{V}_{\mathsf{PCP}}^{(1)}(x, \{(q_i, b_i)\}_i; r)$ accepts with noticeable probability.

The prover $\mathcal{P}^*$ that violates soundness of $\mathsf{FSKM}[\Pi_{\mathsf{PCP}}, \widetilde{\mathcal{H}}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ takes as input

$$\widetilde{h}_{\mathsf{CRHF}} = (\$_{\mathsf{rt}}, \$_{\mathsf{path}}, \mathsf{pre}_{\mathsf{tree}}, \mathsf{pre}_{\mathsf{path}}, h_{\mathsf{rt}}, h'_{\mathsf{tree}}, \mathsf{crs}, \{\mathsf{com}_j^{(\mathsf{zero})}\}_{j \in [\lambda]}),$$

and $\langle h_{\mathsf{FS}} \rangle$. $\mathcal{P}^*$ does the following.

1. Compute $\alpha = h_{\mathsf{rt}}(\langle h_{\mathsf{FS}} \rangle \| \langle \mathcal{A}_{\mathsf{Trivial}} \rangle \| \langle \mathcal{V}_{\mathsf{PCP}} \rangle)$.
2. Compute $\mathsf{rt} = \$_{\mathsf{rt}} \oplus (0^{\lambda + \log(\lambda) + 2} \| \alpha \| 0^{m/2})$.
3. Compute $r = h_{\mathsf{FS}}(\mathsf{rt})$.
4. Run $\mathcal{A}_{\mathsf{Trivial}}$ on r to produce $x, \{b_i\}_i$.
5. Compute $\{q_i\}_i = \mathcal{V}_{\mathsf{PCP}}^{(0)}(x; r)$.
6. Compute $\{\mathsf{sib}_j = \mathsf{block\text{-}com}(h'_{\mathsf{tree}}, \{(q_i, b_i)\}_i, j)\}_{j \in [\lambda - m']}$ and $\{\mathsf{open}'_i = \mathsf{open}(h'_{\mathsf{tree}}, \{(q_i, b_i)\}_i, i)\}_i$.
7. For each $j \in [\lambda - m']$, compute $\pi_j = \mathcal{P}\Big(\mathsf{crs}, \big(\alpha \| \mathsf{bit}_\lambda(j) \| \mathsf{sib}_j\big), \big(\langle h_{\mathsf{FS}} \rangle \| \langle \mathcal{A} \rangle \| \langle \mathcal{V}_{\mathsf{PCP}} \rangle\big)\Big)$, where $\mathcal{P}$ is the SNARK prover.
8. For each i, let d_i be the highest level reached by the partial opening open'_i. Complete each opening

$$\left\{\mathsf{open}_i = \mathsf{open}'_i \cup \left\{\mathsf{sib}_j, \Big((\mathsf{pre}_{\mathsf{path}} \| \$_{\mathsf{path}} \| \mathsf{bit}_\lambda(j) \| \alpha) \;\Big\|\; (\pi_1 \| \ldots \| \pi_j \| 0 \ldots 0)\Big)\right\}_{j \in [d_i - 1]}\right\}_i.$$

9. Output $\left(\mathsf{rt}, r, \{\mathsf{open}_i\}_i\right)$ as an argument that $x \in \mathcal{L}$.

By definition of $\mathcal{A}_{\mathsf{Trivial}}$ (noting that rt is a uniformly random string), the verifier will accept this argument with noticeable probability as long as the Merkle openings are valid. By definition of $\widetilde{\mathcal{H}}_{\mathsf{CRHF}}$, we know that the second part of each opening $\mathsf{open}_i \setminus \mathsf{open}'_i$ will verify properly, since each input will parse with the special structure and contain valid SNARK proofs. The only remaining difficulty is that an input in open'_i (where all hashes are computed by h'_{tree}) might accidentally parse with the special structure, meaning that the verifier would not use h'_{tree} to check consistency. However, this is only possible at a leaf, since all hashes at higher levels will start with $\mathsf{pre}_{\mathsf{tree}}$. Moreover, since $\$_{\mathsf{path}}$ is chosen uniformly at random, the probability of a wrong parse at any given leaf is at most $1/2^\lambda$. Thus, by a union bound, the verifier will fail to accept with probability at most $\mathsf{poly}(\lambda)/2^\lambda = \mathsf{negl}(\lambda)$. □

4 An FSKM-Incompatible PCIP

In this section we show that for every language in NP there exists a probabilistically checkable interactive proof such that for every Fiat-Shamir hash function and every collision-resistant hash function, the resulting FSKM protocol is *not* sound.

Theorem 4.1. *Let $\mathcal{L} \in \mathsf{NP}$. There exists a PCIP Π for $\mathcal{L}$ with negligible soundness error such that for every collision resistant hash function family $\mathcal{H}_{\mathsf{CRHF}}$ and every Fiat-Shamir hash function family $\mathcal{H}_{\mathsf{FS}}$, the protocol $\mathsf{FSKM}[\Pi, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ is not sound.*

As a matter of fact, our attack breaks soundness of $\mathsf{FSKM}[\Pi, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ in an extremely strong sense — it is a $\mathsf{poly}(n, \lambda)$-time attack that causes the verifier to accept *any* input $x \notin \mathcal{L}$, with probability 1.

4.1 Proof of Theorem 4.1

Let $\mathcal{L} \in \mathsf{NP}$ and let $\mathcal{H}_{\mathsf{CRHF}} = \left\{\mathcal{H}^{(\lambda)}_{\mathsf{CRHF}}\right\}_\lambda$, $\mathcal{H}^{(\lambda)}_{\mathsf{CRHF}} = \left\{h_{\mathsf{CRHF}} : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda\right\}$ be a collision resistant hash family. The proof will take the following steps.

1. First, we construct a sound PCIP $(\mathcal{P}, \mathcal{V})$ for $\mathcal{L}$. The PCIP will be constructed in a contrived manner (to facilitate the attack in Step 3). This step is done in Section 4.1.1.
2. We briefly describe the argument system $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}}) = \mathsf{Kilian}[\mathsf{PCIP}, \mathcal{H}_{\mathsf{CRHF}}]$. Its soundness follows from [Kil88, BCS16] (since we are instantiating Kilian's protocol with a sound PCIP and a CRHF). This step is done in Section 4.1.2.
3. Finally, for every hash family $\mathcal{H}_{\mathsf{FS}}$, we present the protocol $\mathsf{FSKM}[\mathsf{PCIP}, \mathcal{H}_{\mathsf{CRHF}}, \mathcal{H}_{\mathsf{FS}}]$ and show an attack that causes the verifier to accept every possible input $x \notin \mathcal{L}$ with probability 1. This last step is done in Section 4.1.3.

4.1.1 A Contrived PCIP for $\mathcal{L}$

We next construct our PCIP $(\mathcal{P}, \mathcal{V})$ for the language $\mathcal{L}$. Before doing so, we first set up some tools that we be used in the construction.

Some Useful Ingredients. Since $\mathcal{L} \in \mathsf{NP}$, by the PCP Theorem (Theorem 2.3), there exists a PCP $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ for $\mathcal{L}$ with soundness error $\epsilon_{\mathcal{L}}(\lambda) = \mathsf{neg}(\lambda)$, query complexity $q_{\mathcal{L}} = \mathsf{polylog}(\lambda)$ and randomness complexity $r_{\mathcal{L}} = \mathsf{poly}(\log(n), \log(\lambda))$.

Recall that $\mathsf{MerkleCom}(h, s)$ is the Merkle tree root generated from s using the hash function h (see Section 2.2.1). Let C be an efficiently computable and decodable error correcting code ensemble with constant rate and constant $\delta_0 > 0$ relative distance (for more background on codes and the fact that such

codes exist, see Appendix B and specifically Theorem B.3). We define an auxiliary pair language $\mathcal{L}'$ as follows:

$$\mathcal{L}' = \left\{ \left(r, \pi = C\big(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle\big)\right) : \begin{array}{l} r \in \{0,1\}^\lambda, \\ \langle h_{\mathsf{CRHF}}\rangle \text{ and } \langle h_{\mathsf{FS}}\rangle \text{ are Boolean circuits, and} \\ h_{\mathsf{FS}}\big(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (1\|\pi))\big) = r \end{array} \right\}$$

In other words, $\mathcal{L}' \subseteq \{0,1\}^* \times \{0,1\}^*$ is a pair language, in which the first part of the input r is a binary string of length λ (which will be the security parameter in the upcoming PCIP), and the second part of the input π is an encoding under C of two Boolean circuits. Jumping ahead, the first circuit h_{CRHF} will play the part of the CRHF used in Kilian's protocol, while the circuit h_{FS} will be the FS hash function. For such (r, π), it holds that $(r, \pi) \in \mathcal{L}'$ if and only if $h_{\mathsf{FS}}\big(\mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (1\|\pi))\big) = r$.

Observe that $\mathcal{L}'$ can be decided by a polynomial time Turing machine (i.e., polynomial in the input length $|r| + |\pi|$). Therefore, by Theorem 2.5 there exists a PCPP proof-system for $\mathcal{L}'$, denoted by $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ such that for input (r, π) and every proximity parameter $\delta > 0$, the soundness error is $\epsilon_{\mathcal{L}'}(\lambda) = \mathsf{neg}(\lambda)$, query complexity $q_{\mathcal{L}'} = \mathsf{poly}(\log(\lambda), \frac{1}{\delta})$ and randomness complexity $r_{\mathcal{L}'} = \mathsf{poly}(\log(\lambda), \log(t)/\delta)$, where $t = |r| + |\pi|$. Furthermore, the prover $\mathcal{P}_{\mathcal{L}'}$ runs in time $\mathsf{poly}(t, \log(\lambda))$, and $\mathcal{V}_{\mathcal{L}'}$ runs in time $\mathsf{poly}(\log(t), \log(\lambda), \frac{1}{\delta})$.

PCIP for $\mathcal{L}$. With these tools in hand, we are ready to present the PCIP. Intuitively (and as hinted on in Section 1.4), the verifier $\mathcal{V}$ will accept input x in one of two possible scenarios: The first scenario (denoted by having the prover initially send a bit $b = 0$), is that the prover provides $\mathcal{V}$ with an honest PCP proof for $x \in \mathcal{L}$, thus allowing the honest prover $\mathcal{P}$ to convince the verifier with probability 1. The second scenario (denoted by having the prover send $b = 1$), is that the prover manages to pass the following test (which will act as a backdoor once we compile the PCIP with FSKM):

The prover $\mathcal{P}$ is required to send a description of a Fiat-Shamir hash function h_{FS} which manages to accurately predict r_1, the random coins of $\mathcal{V}$, which *have yet to be sampled*. A cheating PCIP prover has $2^{-|r_1|} = \mathsf{negl}(\lambda)$ probability of passing the challenge.

In contrast, once the FSKM transform is applied, the challenge becomes easy to beat. A malicious prover will simply commit to the FS hash function provided by the verifier, as described in the FSKM transform. Therefore, the malicious prover will be able to predict the randomness of the verifier, thus passing the test.

In order to make the number of queries by $\mathcal{V}$ polylogarithmic in λ, we shall have the prover send its messages via an error-correcting code and run a PCPP checking that the verifier would have accepted had it explicitly read all of π_1.

The PCIP $(\mathcal{P}, \mathcal{V})$ is formally described in Fig. 3. In the protocol we use the convention that messages received by the verifier from the prover, which might be maliciously crafted, are denoted with a tilde.

We emphasize that while the *honest* prover always sends $b = 0$ and π_2 as the empty string, a cheating prover might not. Indeed, we added the possibility of sending different values here as a kind of backdoor. As we shall show, this backdoor does not violate the soundness of $(\mathcal{P}, \mathcal{V})$ as a PCIP (see Lemma 4.2) but completely breaks the soundness of the construction after applying Kilian and Fiat-Shamir (see Corollary 4.3).

Lemma 4.2. *The protocol $(\mathcal{P}, \mathcal{V})$ is a PCIP for $\mathcal{L}$ with negligible soundness error.*

Proof. We show that the protocol satisfies the required complexity restrictions and that completeness and soundness hold.

Complexity. The verifier $\mathcal{V}$ first reads b. This means that 1 bit is queried, in addition to the bits queried by both of the verifiers $\mathcal{V}_{\mathcal{L}}$ and $\mathcal{V}_{\mathcal{L}'}$.[8] Therefore the total query complexity is $O(q_{\mathcal{L}} + q_{\mathcal{L}'}) = \mathsf{polylog}(\lambda)$.

The verifier $\mathcal{V}$ only uses randomness to simulate $\mathcal{V}_{\mathcal{L}}$ and $\mathcal{V}_{\mathcal{L}'}$, therefore its randomness complexity is $O(r_{\mathcal{L}} + r_{\mathcal{L}'}) = \mathsf{polylog}(n, \lambda)$.

Both verifiers $\mathcal{V}_{\mathcal{L}}$ and $\mathcal{V}_{\mathcal{L}'}$ run in time $\mathsf{polylog}(n, \lambda)$ and so $\mathcal{V}$ runs in time $\mathsf{polylog}(n, \lambda)$ as well. The prover $\mathcal{P}$ also runs in time $\mathsf{poly}(n, \log(\lambda))$, as simulating $\mathcal{P}_{\mathcal{L}}(x, w, 1^\lambda)$ takes time $\mathsf{poly}(n, \log(\lambda))$.

[8] The verifier can be easily made non-adaptive by having it make both query sets and (at most) doubling its query complexity.

Protocol 3: PCIP for $\mathcal{L} \in \mathsf{NP}$

Common input: Input $x \in \{0,1\}^n$ and security parameter 1^λ

Prover's auxiliary input: Witness w

1 $\mathcal{P}$ sends $(b \| \pi_1)$, where $b = 0$ and $\pi_1 = \mathcal{P}_{\mathcal{L}}(x, w)$ is the PCP proof.

2 $\mathcal{V}$ receives $(\tilde{b} \| \tilde{\pi}_1)$, where $\tilde{b} \in \{0,1\}$. If $\tilde{b} = 0$, $\mathcal{V}$ generates $r_1 \in_R \{0,1\}^{r_{\mathcal{L}}}$ (i.e., randomness for the PCP verifier $\mathcal{V}_{\mathcal{L}}$). Otherwise, (i.e., $b = 1$) $\mathcal{V}$ chooses uniformly $r_1 \in_R \{0,1\}^\lambda$.[a]

3 $\mathcal{V}$ sends r_1 to $\mathcal{P}$.

4 $\mathcal{P}$ sends an empty string π_2.

5 $\mathcal{V}$ receives the string $\tilde{\pi}_2$.

6 If $b = 0$, then $\mathcal{V}$ runs $\mathcal{V}_{\mathcal{L}}^{\tilde{\pi}_1}(x, 1^\lambda)$, and accepts iff $\mathcal{V}_{\mathcal{L}}$ accepted.

7 Otherwise, (i.e., $b = 1$) the verifier $\mathcal{V}$ runs $\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}\big(r_1, 1^\lambda, \delta_0/2\big)$, where $\tilde{\pi}_1$ is used as the implicit input, $\tilde{\pi}_2$ is used as the proof, and $\delta_0/2$ is the proximity parameter. The verifier $\mathcal{V}$ accepts iff $\mathcal{V}_{\mathcal{L}'}$ accepted.

[a] Formally, for the protocol to be public coin, we need r_1 to be completely random and not depend on prior messages. This can be achieved by first having the prover send b in a separate message, which the verifier will then query (thus learning the value of $\tilde{b}$), and having the verifier pad r_1 to be of length $\max\{r_{\mathcal{L}}, \lambda\}$. Indeed, we only make this distinction to highlight the different role played by r_1 when $b = 0$ and when $b = 1$.

Figure 3: $(\mathcal{P}, \mathcal{V})$ a PCIP for $\mathcal{L}$.

Completeness. Let $x \in \mathcal{L}$. By the completeness of the PCP $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ for $\pi_1 = \mathcal{P}_{\mathcal{L}}(x, w, 1^\lambda)$ it holds that $\Pr[\mathcal{V}_{\mathcal{L}}^{\pi_1}(x, 1^\lambda) = 1] = 1$. By construction, since $\mathcal{V}_{\mathcal{L}}$ accepts, the verifier $\mathcal{V}$ will also accept.

Soundness. Let $x \notin \mathcal{L}$ and $\mathcal{P}^*$ be a malicious prover. Without loss of generality we can assume that $\mathcal{P}^*$ is deterministic (by fixing the best possible choice of random string). Denote the first message sent by $\mathcal{P}^*$ on input x by $(\tilde{b}, \tilde{\pi}_1)$, where $\tilde{b} \in \{0,1\}$.

Assume first that $\tilde{b} = 0$. Note that since $\mathcal{P}^*$ provides $\tilde{\pi}_1$ before $\mathcal{V}$ sends its queries, by the soundness of the PCP $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ and the fact that $x \notin \mathcal{L}$, it holds that

$$\Pr\left[\mathcal{V}_{\mathcal{L}}^{\tilde{\pi}_1}(x, 1^\lambda) = 1\right] = \mathsf{negl}(\lambda),$$

and so in this case $\mathcal{V}$ accepts with negligible probability as required.

Consider now the case that $\tilde{b} = 1$. We now further distinguish between two cases. First, let us assume that $\Delta(\tilde{\pi}_1, C) \geq \delta_0/2$, where recall that δ_0 is the relative distance of the code C. Given r_1, by the definition of $\mathcal{L}'$, the projection of $\mathcal{L}'$ on r_1 is

$$\mathcal{L}'(r_1) = \Big\{\pi \text{ s.t. } \pi = C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle) \text{ and } h_{\mathsf{FS}}(\mathsf{com}) = r_1, \text{ for } \mathsf{com} = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (1\|\pi))\Big\}.$$

In particular, $\mathcal{L}'(r_1) \subseteq C$, which means that $\Delta(\tilde{\pi}_1, \mathcal{L}'(r_1)) \geq \Delta(\tilde{\pi}_1, C) \geq \delta_0/2$. Therefore, by the soundness of $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$, it holds that

$$\Pr\left[V_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}(r_1, 1^\lambda, \delta_0/2) = 1\right] = \mathsf{negl}(\lambda),$$

as desired.

Thus, we may now assume that $\Delta(\tilde{\pi}_1, C) < \delta_0/2$. Since C has relative distance δ_0, there exists a *unique* codeword $w \in C$ such that $\Delta(\tilde{\pi}_1, w) < \delta_0/2$.

The codeword w can be parsed as $w = C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle)$ for $\langle h_{\mathsf{CRHF}}\rangle$ the circuit description of a CRHF and $\langle h_{\mathsf{FS}}\rangle$ the circuit description of a FS hash function. Let $\mathsf{com} = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (1\|w))$. The probability that the malicious adversary $\mathcal{P}^*$ can choose h_{FS} such that $h_{\mathsf{FS}}(\mathsf{com}) = r_1$, *before* r_1 *was even sampled* is $2^{-|r_1|} = 2^{-\lambda}$. Assuming $h_{\mathsf{FS}}(\mathsf{com}) \neq r_1$, by the relative distance of the code C it holds that $\Delta\big(\tilde{\pi}_1, \mathcal{L}'(r_1)\big) \geq \Delta\big(C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle), \mathcal{L}'(r_1)\big) \geq \delta_0 > \delta_0/2$. Therefore, by the soundness of $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ it holds that

$$\Pr\left[\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}\big(r_1, 1^\lambda, \delta_0/2\big) = 1 | h_{\mathsf{FS}}(\mathsf{com}) \neq r_1\right] = \mathsf{negl}(\lambda).$$

Protocol 4: Kilian of $(\mathcal{P}, \mathcal{V})$

Common input: Input $x \in \{0,1\}^n$ and security parameter 1^λ

Prover's auxiliary input: Witness w

1 $\mathcal{V}_{\mathsf{Kilian}}$ generates a hash function $h_{\mathsf{CRHF}} \leftarrow \mathcal{H}^{(\lambda)}_{\mathsf{CRHF}}$ and sends $\langle h_{\mathsf{CRHF}} \rangle$ to $\mathcal{P}_{\mathsf{Kilian}}$.

2 $\mathcal{P}_{\mathsf{Kilian}}$ sends $\mathsf{com}_1 = \mathsf{MerkleCom}\big(h_{\mathsf{CRHF}}, (b \| \pi_1)\big)$, where $b = 0$ and $\pi_1 = \mathcal{P}_{\mathcal{L}}(x, w)$.

3 $\mathcal{V}_{\mathsf{Kilian}}$ generates r_1 as defined by $\mathcal{V}$, and sends r_1 to $\mathcal{P}_{\mathsf{Kilian}}$.

4 $\mathcal{P}_{\mathsf{Kilian}}$ reveals the queried values, by sending $\mathsf{open}_1 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, \pi_1, Q_1)$, where Q_1 is the set of queries generated by the verifier $\mathcal{V}$ with input x randomness r_1.

5 $\mathcal{P}_{\mathsf{Kilian}}$ sends $\mathsf{com}_2 = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi_2)$ where π_2 is an empty string.

6 $\mathcal{V}_{\mathsf{Kilian}}$ sends r_2 as described by $\mathcal{V}$.

7 $\mathcal{P}_{\mathsf{Kilian}}$ sends $\mathsf{open}_2 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, \pi_2, Q_2)$, where Q_2 is the set of queries generated by the verifier $\mathcal{V}$ with explicit input r_1 and randomness r_2.

8 $\mathcal{V}_{\mathsf{Kilian}}$ first computes $v_j = \mathsf{MerkleVer}(h_{\mathsf{CRHF}}, \mathsf{com}_j, \mathsf{open}_j)$ for $j \in \{1,2\}$. If $v_1 \wedge v_2 \neq 1$, then $\mathcal{V}_{\mathsf{Kilian}}$ rejects.

$\mathcal{V}_{\mathsf{Kilian}}$ extracts the following values from the openings open_1 and open_2:

- $\tilde{b}$ the first bit of $(\tilde{b} \| \tilde{\pi}_1)$, as committed to in com_1, and revealed in open_1.
- $\left\{b_1^{(i_1)}\right\}_{i_1}$ the answers to the queries to $\tilde{\pi_1}$, as specified by r_1, and revealed in open_1.
- $\left\{b_2^{(i_2)}\right\}_{i_2}$ the answers to the queries to $\tilde{\pi}_2$, as specified by r_2, and revealed in open_2.

If $\tilde{b} = 0$, then $\mathcal{V}_{\mathsf{Kilian}}$ emulates $\mathcal{V}_{\mathcal{L}}(x, 1^\lambda)$ provided the answers $\left\{b_1^{i_1}\right\}_{i_1}$ to the queries in Q_1, and accepts if and only if $\mathcal{V}_{\mathcal{L}}$ accepted.

Otherwise, (for $\tilde{b} = 1$) $\mathcal{V}_{\mathsf{Kilian}}$ emulates $\mathcal{V}_{\mathcal{L}'}(r_1, 1^\lambda, \delta_0/2)$ provided the answers $\left\{b_1^{i_1}\right\}_{i_1}, \left\{b_2^{i_2}\right\}_{i_2}$ to the queries in Q_1 and queries in Q_2, respectively, and accepts if and only if $\mathcal{V}_{\mathcal{L}'}$ accepted.

Figure 4: $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}})$, an argument system for $\mathcal{L}$.

Hence, by the soundness of $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$

$$\begin{aligned}
\Pr[\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}(r_1, 1^\lambda, \delta_0/2) = 1] &\leq \Pr[\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}(r_1, 1^\lambda, \delta_0/2) = 1 | h_{\mathsf{FS}}(\mathsf{com}) = r_1] \cdot \Pr[h_{\mathsf{FS}}(\mathsf{com}) = r_1] \\
&\quad + \Pr[\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1, \tilde{\pi}_2}(r_1, 1^\lambda, \delta_0/2) = 1 | h_{\mathsf{FS}}(\mathsf{com}) \neq r_1] \cdot \Pr[h_{\mathsf{FS}}(\mathsf{com}) \neq r_1] \\
&\leq 1 \cdot 2^{-\lambda} + \mathsf{negl}(\lambda) \cdot 1 \\
&= \mathsf{negl}(\lambda).
\end{aligned}$$

Thus, in all cases the verifier $\mathcal{V}$ has a negligible probability of accepting. □

4.1.2 Applying Kilian's Protocol to $(\mathcal{P}, \mathcal{V})$

As our next step (the second step in the outline), we present the protocol resulting from applying Kilian's protocol to the PCIP $(\mathcal{P}, \mathcal{V})$. The resulting protocol $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}})$ is presented in Fig. 4.

Ben-Sasson *et al.* [BCS16] showed that applying Kilian's protocol to *any* sound PCIP results in a sound interactive argument. Thus, we obtain the following result as an immediate corollary of Lemma 4.2:

Corollary 4.3. *The argument system $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}})$ has negligible soundness error.*

4.1.3 Attack on Fiat-Shamir of $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}})$

Lastly, consider $(\mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$, as presented in Fig. 5, the result of applying Fiat-Shamir to the previous protocol for hash function ensemble $\mathcal{H}_{\mathsf{FS}} = \left\{\mathcal{H}^{(\lambda)}_{\mathsf{FS}}\right\}_\lambda$:

Protocol 5: FSKM for $\mathcal{L} \in \mathsf{NP}$

Common input: Input $x \in \{0,1\}^n$ and security parameter 1^λ

Prover's auxiliary input: Witness w

$\mathcal{V}_{\mathsf{FS}}$ generates hash functions $h_{\mathsf{CRHF}} \leftarrow \mathcal{H}^{(\lambda)}_{\mathsf{CRHF}}$ and $h_{\mathsf{FS}} \leftarrow \mathcal{H}^{(\lambda)}_{\mathsf{FS}}$ and sends their description to $\mathcal{P}_{\mathsf{FS}}$.

$\mathcal{P}_{\mathsf{FS}}$ sends the following in a single message:

- $\mathsf{com}_1 = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (b \| \pi_1))$, where $b = 0$ and $\pi_1 = \mathcal{P}_{\mathcal{L}}(x, w, 1^\lambda)$.
- $r_1 = h_{\mathsf{FS}}(\mathsf{com}_1)$.
- $\mathsf{open}_1 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, (b \| \pi_1), Q_1)$, where Q_1 is the set of queries generated by the verifier $\mathcal{V}$ with input x randomness r_1.
- $\mathsf{com}_2 = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, \pi_2)$, where π_2 is an empty string.
- $r_2 = h_{\mathsf{FS}}(\mathsf{com}_1 \| r_1 \| \mathsf{open}_1 \| \mathsf{com}_2)$.
- $\mathsf{open}_2 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, \pi_2, Q_2)$, where Q_2 is the set of queries generated by the verifier $\mathcal{V}$ with explicit input r_1 and randomness r_2.

$\mathcal{V}_{\mathsf{FS}}$ first checks that $r_1 = h_{\mathsf{FS}}(\mathsf{com}_1)$ and $r_2 = h_{\mathsf{FS}}(\mathsf{com}_1 \| r_1 \| \mathsf{open}_1 \| \mathsf{com}_2)$, if not, it rejects.
$\mathcal{V}_{\mathsf{FS}}$ computes $v_j = \mathsf{MerkleVer}(h_{\mathsf{CRHF}}, \mathsf{com}_j, \mathsf{open}_j)$ for $j \in \{1,2\}$. If $v_1 \wedge v_2 \neq 1$, then $\mathcal{V}_{\mathsf{FS}}$ rejects.
Lastly, $\mathcal{V}_{\mathsf{FS}}$ extracts the following values from the openings open_1 and open_2:

- $\tilde{b}$ the first bit of $(\tilde{b} \| \tilde{\pi}_1)$, as committed to in com_1, and revealed in open_1.
- $\left\{b_1^{i_1}\right\}_{i_1}$ the answers to the queries to $\tilde{\pi}_1$, as specified by r_1, and revealed in open_1.
- $\left\{b_2^{i_2}\right\}_{i_2}$ the answers to the queries to $\tilde{\pi}_2$, as specified by r_2, and revealed in open_2.

If $\tilde{b} = 0$, then $\mathcal{V}_{\mathsf{FS}}$ emulates $\mathcal{V}_{\mathcal{L}}(x, 1^\lambda)$ provided the answers $\left\{b_1^{i_1}\right\}_{i_1}$ to the queries in Q_1, and accepts if and only if $\mathcal{V}_{\mathcal{L}}$ accepted.
Otherwise, (for $\tilde{b} = 1$) $\mathcal{V}_{\mathsf{FS}}$ emulates $\mathcal{V}_{\mathcal{L}'}(r_1, 1^\lambda, \delta_0/2)$ provided the answers $\left\{b_1^{i_1}\right\}_{i_1}, \left\{b_2^{i_2}\right\}_{i_2}$ to the queries in Q_1 and Q_2, respectively, and accepts if and only if $\mathcal{V}_{\mathcal{L}'}$ accepted.

Figure 5: Fiat-Shamir of $(\mathcal{P}_{\mathsf{Kilian}}, \mathcal{V}_{\mathsf{Kilian}})$.

Lemma 4.4. *There exists a malicious prover $\mathcal{P}^*$ such that for every input $x \in \{0,1\}^n$ and security parameter λ, it holds that $\mathcal{P}^*$ runs in time $\mathsf{poly}(n,\lambda)$ and*

$$\Pr\left[\langle \mathcal{P}^*(x,1^\lambda) \leftrightarrow \mathcal{V}_{\mathsf{FS}}(x,1^\lambda)\rangle = 1\right] = 1.$$

Proof. Let $x \in \{0,1\}^n$ and λ be a security parameter. Consider the following construction of a malicious prover $\mathcal{P}^*$:

1. Upon receiving $\langle h_{\mathsf{CRHF}}\rangle$ and $\langle h_{\mathsf{FS}}\rangle$ from $\mathcal{V}_{\mathsf{FS}}$, the prover $\mathcal{P}^*$ computes $\pi_1 = C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle)$. The prover $\mathcal{P}^*$ sends $\mathsf{com}_1 = \mathsf{MerkleCom}\big(h_{\mathsf{CRHF}}, (b\|\pi_1)\big)$, with $b = 1$ (indeed $b = 1$ indicated that we are using the "backdoor").
2. $\mathcal{P}^*$ generates $r_1 = h_{\mathsf{FS}}(\mathsf{com}_1)$ in accordance with the Fiat-Shamir transform.
3. $\mathcal{P}^*$ reveals the first commitment by sending $\mathsf{open}_1 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, (1\|\pi_1), r_1)$.
4. For the second message, $\mathcal{P}^*$ sets $\pi_2 = \mathcal{P}_{\mathcal{L}'}((r_1,\pi_1), 1^\lambda)$.
5. Once more, by the Fiat-Shamir transform, $r_2 = h_{\mathsf{FS}}(\mathsf{com}_1 \| r_1 \| \mathsf{open}_1 \| \mathsf{com}_2)$.
6. $\mathcal{P}^*$ reveals the second commitment by sending $\mathsf{open}_2 = \mathsf{MerkleOpen}(h_{\mathsf{CRHF}}, \pi_2, r_2)$.

Note that $r_1 = h_{\mathsf{FS}}(\mathsf{com}_1)$, $r_2 = h_{\mathsf{FS}}(\mathsf{com}_1 \| r_1 \| \mathsf{open}_1 \| \mathsf{com}_2)$ as P^* generated randomness according to the FS transform, and that $v_1 = v_2 = 1$ as P^* performs the decommitments honestly.

Since $\pi_1 = C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle)$, and for $\mathsf{com}_1 = \mathsf{MerkleCom}(h_{\mathsf{CRHF}}, (1,\pi))$ it holds that $h_{\mathsf{FS}}(\mathsf{com}_1) = r_1$, it implies that $(r_1,\pi_1) \in \mathcal{L}'$. Therefore, by the completeness of $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ and the choice of $\pi_2 = \mathcal{P}_{\mathcal{L}'}((r_1,\pi_1),1^\lambda)$, it holds that

$$\Pr[\mathcal{V}_{\mathcal{L}'}^{\tilde{\pi}_1,\tilde{\pi}_2}(r_1, 1^\lambda, \delta_0/2) = 1] = 1$$

which by the definition of $\mathcal{V}_{\mathsf{FS}}$ implies that

$$\Pr[\langle \mathcal{P}^*(x,1^\lambda) \leftrightarrow \mathcal{V}_{\mathsf{FS}}(x,1^\lambda)\rangle = 1] = 1.$$

Observe that both $|\langle h_{\mathsf{CRHF}}\rangle| = \mathsf{poly}(n,\lambda)$ and $|\langle h_{\mathsf{FS}}\rangle)| = \mathsf{poly}(n,\lambda)$. Therefore, by the efficient encoding of C, the computation of $C(\langle h_{\mathsf{CRHF}}\rangle \| \langle h_{\mathsf{FS}}\rangle)$ takes $\mathsf{poly}(n,\lambda)$ time. The protocols $\mathsf{MerkleCom}$ and $\mathsf{MerkleOpen}$ both take $\mathsf{poly}(n,\lambda)$ time. Lastly, the emulation of $\mathcal{P}_{\mathcal{L}'}$ can also be performed in time $\mathsf{poly}(n,\lambda)$.

Therefore, the malicious prover $\mathcal{P}^*$ runs in time $\mathsf{poly}(n,\lambda)$, convincing the prover $\mathcal{V}_{\mathsf{FS}}$ to accept any input x with probability 1.

□

5 A Secure Fiat-Shamir Hash Function for a Trivial PCP

In this section we show that the FSKM protocol *can* be securely instantiated (based on standard assumptions), albeit only for a trivial PCP. We start by introducing the tools that we will use in the construction.

5.1 Technical Tools: Correlation Intractability and Somewhere Statistically Binding Hashing

The following definitions are taken verbatim from [CCH+19].

Definition 5.1 (Correlation Intractability)**.** *For a given relation ensemble $R = \{\mathcal{R}_\lambda \subseteq \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)}\}$, a hash family $\mathcal{H} = \{\mathcal{H}_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}$ is said to be δ-correlation intractable for R if for every polynomial-size adversary $\mathcal{A}$,*

$$\Pr_{h \leftarrow \mathcal{H}_\lambda, x \leftarrow \mathcal{A}(h)}[(x, h(x)) \in R] = O(\delta(\lambda)).$$

Definition 5.2 (Sparsity). *For any relation ensemble $R = \{\mathcal{R}_\lambda \subseteq \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)}\}$, we say that R is $\rho(\cdot)$-sparse if for $\lambda \in \mathbb{N}$ and any $x \in \{0,1\}^{n(\lambda)}$,*

$$\Pr_{y \leftarrow \{0,1\}^{m(\lambda)}}[(x,y) \in R] \leq \rho(\lambda).$$

Definition 5.3 (Efficiently Searchable Relation). *We say that a relation ensemble R is* efficiently searchable *if there exists a $\mathsf{poly}(\lambda)$-time function $f = f_R : \{0,1\}^* \to \{0,1\}^*$ that for any input x, if $(x,y) \in R$, then $y = f(x)$; that is, $f(x)$ is the unique y such that $(x,y) \in R$, provided such a y exists.*

Definition 5.4 (Efficiently Sampleable Relation). *We say that a relation ensemble R is* efficiently sampleable *if there exists a $\mathsf{poly}(\lambda)$-time algorithm $\mathsf{Samp}(x;r)$ such that for any $(x^*,y^*) \in R$,*

$$\Pr_r[\mathsf{Samp}(x^*;r) = y^*] = |\{y \in \{0,1\}^m : (x^*,y) \in R\}|^{-1}.$$

It is shown in [CCH+19] that correlation intractability for efficiently searchable relations directly implies correlation intractability for efficiently sampleable relations for which every input x has at most polynomially many outputs y for which $(x,y) \in R$.

Peikert and Shiehian [PS19] recently constructed a correlation intractability hash family for any efficiently searchable relation, assuming LWE with suitable polynomial approximation factors.

Definition 5.5 (SSB Hash [HW15a]). *A* somewhere statistically binding hash *(SSB) consists of polynomial time algorithms $\mathcal{H}_{\mathsf{SSB}} = (\mathsf{Gen}_{\mathsf{SSB}}, \mathsf{Eval}_{\mathsf{SSB}}, \mathsf{Open}_{\mathsf{SSB}}, \mathsf{Verify}_{\mathsf{SSB}})$ along with a block alphabet $\Sigma = \{0,1\}^{\ell_{blk}}$, output size ℓ_{hash} and opening size ℓ_{opn}, where $\ell_{blk}(\lambda), \ell_{hash}(\lambda), \ell_{opn}(\lambda)$ are some fixed polynomials in the security parameter. The algorithms have the following syntax:*

- $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, L, i)$ *takes as input an integer $L \leq 2^\lambda$ and index $i \in \{0,\ldots,L-1\}$ (both of these are in binary) and outputs a public hashing key h_{SSB}.*
- $\mathsf{Eval}(h_{\mathsf{SSB}}, x)$ *is a deterministic polynomial time algorithm that takes the hash key h_{SSB} and input $x = (x[0],\ldots,x[L-1]) \in \Sigma^L$ and outputs $y \in \{0,1\}^{\ell_{hash}}$.*
- $\pi \leftarrow \mathsf{Open}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, x, j)$ *: Given the hash key $h_{\mathsf{SSB}}, x \in \Sigma^L$, and an index $j \in \{0,\ldots,L-1\}$, creates an opening $\pi \in \{0,1\}^{opn}$.*
- $\mathsf{Verify}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, y, j, u, \pi)$ *: Given a hash key h_{SSB} and $y \in \{0,1\}^{\ell_{hash}}$, an integer index $j \in \{0,\ldots,L-1\}$, a value $u \in \Sigma$ and an opening $\pi \in \{0,1\}^{\ell_{opn}}$, outputs a decision $\in \{\mathsf{accept}, \mathsf{reject}\}$. This is intended to verify that a pre-image x of $y = \mathsf{Eval}(h_{\mathsf{SSB}}, x)$ has $x[j] = u$.*

We require the following properties:

Correctness: *For any integers $L \leq 2^\lambda$ and $i, j \in \{0,\ldots,L-1\}$, and $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, L, i), x \in \Sigma^L, \pi \leftarrow \mathsf{Open}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, x, j)$: we have $\mathsf{Verify}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, \mathsf{Eval}(h_{\mathsf{SSB}}, x), j, x[j], \pi) = \mathsf{accept}$.*

Index Hiding: *We consider the following game between an attacker $\mathcal{A}$ and a challenger:*

- *The attacker $\mathcal{A}(1^\lambda)$ chooses an integer L and two indices $i_0, i_1 \in \{0,\ldots,L-1\}$.*
- *The challenger chooses a bit $b \leftarrow \{0,1\}$ and sets $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, L, i_b)$.*
- *The attacker $\mathcal{A}$ gets h_{SSB} and outputs a bit b'.*

We require that for any polynomial time attacker $\mathcal{A}$ we have $|\Pr[b = b'] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$ in the above game.

Somewhere Statistically Binding: *We say that h_{SSB} is* statistically binding for an *index i if there do not exist any values $y, u \neq u', \pi, \pi'$ s.t. $\mathsf{Verify}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, y, i, u, \pi) = \mathsf{Verify}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, y, i, u', \pi') = \mathsf{accept}$. We require that for any integers $L \leq 2^\lambda, i \in \{0,\ldots,L-1\}$, the key $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, L, i)$ is statistically binding for i.*

Hubáček and Wichs [HW15a] construct an SSB hash from an FHE scheme $\mathcal{E} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$. We will not give all the details of the construction here, and instead just note that for an h_{SSB} generated with index i, the value of $\mathsf{Eval}(h_{\mathsf{SSB}}, x)$ is $\mathsf{FHE.Enc}_{\mathsf{sk}}(x[i])$, where the FHE key pair $(\mathsf{pk}, \mathsf{sk})$ is drawn during SSB key generation.

5.2 An FSKM-Compatible Construction

We consider a variant of the trivial PCP $\Pi_\emptyset$ defined in Section 3.2. In this variant, the PCP proof string is composed of $q(\lambda)$ blocks, each of length $\log^2(\lambda)$ bits. The verifier selects a random block $i \in [q]$ and a random string $r \in \{0,1\}^{\log^2(\lambda)}$ and check that the i-th block of the proof string is exactly equal to r. In the rest of this subsection we refer to this PCP as $\mathsf{PCP}_\emptyset$.

We show that under standard assumptions, it is possible to soundly instantiate the FSKM protocol with $\Pi_\emptyset$, if we replace the collision resistant hash function with an SSB hash. The Fiat-Shamir hash function we use will be a correlation intractable hash function for efficiently searchable relations. While FSKM for the empty language is not necessarily interesting on its own, we hope that this is a first step towards obtaining stronger possibility results for soundly instantiating FSKM. The reason that we do not contradict Corollary 3.6 is because our feasibility result requires the collision resistant hash function to satisfy an additional property.

Theorem 5.6. *Let $\mathcal{H}_{\mathsf{SSB}} = (\mathsf{Gen}_{\mathsf{SSB}}, \mathsf{Eval}_{\mathsf{SSB}}, \mathsf{Open}_{\mathsf{SSB}}, \mathsf{Verify}_{\mathsf{SSB}})$ be the somewhere statistically binding hash from [HW15a], and let $\mathcal{H}_{\mathsf{CI}}$ be a correlation intractable hash family for efficiently searchable relations. Then the protocol $\mathsf{FSKM}[\Pi_\emptyset, \mathcal{H}_{\mathsf{SSB}}, \mathcal{H}_{\mathsf{CI}}]$ is sound.*

Proof. Let $(\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be an FHE scheme and fix polynomials $q(\lambda)$ and $m(\lambda)$. For a given $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^\lambda)$, define the following relation $\mathcal{R}_{\mathsf{sk}}$:

$$\mathcal{R}_{\mathsf{sk}} = \{(x, (r_0 \mid r_1)) : r_0 \in \{0,1\}^{\log(q(\lambda))}, r_1 \in \{0,1\}^{m(\lambda)}, r_1 = \mathsf{FHE.Dec}_{\mathsf{sk}}(x)\}.$$

Note that $\mathcal{R}_{\mathsf{sk}}$ is efficiently sampleable given sk, and has polynomially many outputs $(r_0 \mid r_1)$ for each input x. Thus there exists a hash family $\mathcal{H}_{\mathsf{CI}}$ that is correlation intractable for $\mathcal{R}_{\mathsf{sk}}$, assuming LWE.

Now consider $\mathcal{H}_{\mathsf{SSB}}$ with block alphabet $\{0,1\}^{\log^2(\lambda)}$. The $\mathsf{FSKM}[\Pi_\emptyset, \mathcal{H}_{\mathsf{SSB}}, \mathcal{H}_{\mathsf{CI}}]$ protocol is written explicitly in Fig. 6.

Protocol 6: $\mathsf{FSKM}[\Pi_\emptyset, \mathcal{H}_{\mathsf{SSB}}, \mathcal{H}_{\mathsf{CI}}]$

Common input: 1^λ

1. $\mathcal{V}$ generates hash keys $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, q(\lambda), 0), h_{\mathsf{CI}} \leftarrow \mathcal{H}_{\mathsf{CI}}^{(\lambda)}$ and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ chooses arbitrary π, computes $\mathsf{rt} = \mathsf{Eval}_{\mathsf{SSB}}(h_{\mathsf{SSB}}, \pi), (r_0 \mid\mid r_1) = h_{\mathsf{CI}}(\mathsf{rt})$ where $r_0 \in \{0,1\}^{\log(q(\lambda))}$ and $r_1 \in \{0,1\}^{\log^2(\lambda)}$, and sends $(\mathsf{rt}, r_0, r_1, \mathsf{op} := \mathsf{Open}(h_{\mathsf{SSB}}, \mathsf{rt}, r_0))$ to $\mathcal{V}$.
3. $\mathcal{V}$ accepts iff $h_{\mathsf{CI}}(\mathsf{rt}) = (r_0 \mid\mid r_1)$ and $\mathsf{Verify}(h_{\mathsf{SSB}}, \mathsf{rt}, r_0, r_1, \mathsf{op})$ accepts.

Figure 6: The Fiat-Shamir transform applied to Kilian's protocol, instantiated with an SSB hash and correlation intractable hash function. Note that there is no common input x since $\Pi_\emptyset$ is for the empty language.

We argue that $\mathsf{FSKM}[\Pi_\emptyset, \mathcal{H}_{\mathsf{SSB}}, \mathcal{H}_{\mathsf{CI}}]$ is sound. Let $\mathcal{A}$ be a prover that convinces $\mathcal{V}$ to accept some statement x with $1/\mathsf{poly}(\lambda)$ probability. Since there are $q(\lambda) = \mathsf{poly}(\lambda)$ possible values of r_0, by averaging, there exists some $r_0^* \in \{0,1\}^{\log(q(\lambda))}$ such that on input h_{SSB} and h_{CI}, $\mathcal{A}$ returns $(\mathsf{rt}, r_0, r_1, \mathsf{op})$ such that $\mathsf{Verify}(h_{\mathsf{SSB}}, \mathsf{rt}, r_0, r_1, \mathsf{op})$ accepts *and* $r_0 = r_0^*$ with $1/\mathsf{poly}(\lambda)$ probability. Thus, by the index hiding property of $\mathcal{H}_{\mathsf{SSB}}$, if $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}_{\mathsf{SSB}}(1^\lambda, q(\lambda), r_0^*)$, (that is, the SSB hash key is sampled to be statistically binding at position r_0^*) then $\mathcal{A}$ also returns $(\mathsf{rt}, r_0, r_1, \mathsf{op})$ such that the above properties are satisfied with $1/\mathsf{poly}(\lambda)$ probability.

Now consider the following reduction $\mathcal{B}$ that breaks the correlation intractability of $\mathcal{H}_{\mathsf{CI}}$. $\mathcal{B}$ receives h_{CI} from its challenger and samples $h_{\mathsf{SSB}} \leftarrow \mathsf{Gen}(1^\lambda, q(\lambda), r_0^*)$. $\mathcal{B}$ sends $(h_{\mathsf{CI}}, h_{\mathsf{SSB}})$ to $\mathcal{A}$, which returns $(\mathsf{rt}, r_0, r_1, \mathsf{op})$. Then with $1/\mathsf{poly}(\lambda)$ probability, rt is a hash that opens to r_1 at location $r_0 = r_0^*$, so $r_1 = \mathsf{FHE.Dec}_{\mathsf{sk}}(\mathsf{rt})$, and $\mathcal{B}$ can output $(\mathsf{rt}, (r_0^* \mid r_1))$ which breaks the correlation intractability of $\mathcal{H}_{\mathsf{CI}}$. □

Acknowledgements

We thank the anonymous TCC 2019 reviewers for useful comments.

References

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, 2001.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBC+17] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 551–579, 2017.

[BBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 14:1–14:17, 2018.

[BBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017.

[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 2012*, pages 326–349. ACM, January 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 401–414. ACM, January 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, pages 103–128, 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[BEG+91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *32nd FOCS*, pages 90–99. IEEE Computer Society Press, October 1991.

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

[BGG17] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017. http://eprint.iacr.org/2017/602.

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short pcps verifiable in polylogarithmic time. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 120–134, 2005.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. http://eprint.iacr.org/2017/1050.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: From practice to theory. 2019.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society Press, May 2015.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[FFG+16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1304–1316. ACM Press, October 2016.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th FOCS*, pages 850–858. IEEE Computer Society Press, October 2018.

[HR04] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 92–105. Springer, Heidelberg, August 2004.

[HW15a] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[HW15b] Pavel Hub'avcek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172, 2015.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31, 1988.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

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

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. 2019.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Information Theory*, 42(6):1723–1731, 1996.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

A A SNARK with Computationally Unique Proofs

In this section, we argue that a fully-succinct SNARK implicit in [BCI+13, BCCT13] has computationally unique proofs. In particular, we show that a particular *preprocessing* SNARK of [BCI+13] built on a knowledge of exponent assumption (see Assumption A.3 below) has computationally unique proofs.[9] Then we argue that the bootstrapping procedure of [BCCT13] which converts a preprocessing SNARK into a fully-succinct SNARK preserves the computational unique proofs property. This implies that the impossibility stated in Theorem 3.5 holds assuming Assumption A.3. Throughout this section, all SNARKs will be publicly verifiable.

We will actually first show how to construct a SNARK satisfying a weaker soundness property, which we call leveled adaptive proof of knowledge.[10] This is a notion of soundness against cheating provers that are not only polynomial-size, but also only make *claims* whose NP verification time grows polynomially. In particular, there are no guarantees about polynomial-size provers that on security parameter λ, produce a false claim about a time-$\lambda^{\log \lambda}$ nondeterministic computation.

- **Leveled Adaptive Proof of Knowledge.** For every polynomial-sized prover $\mathcal{P}^*$ there exists a polynomial-sized extractor $\mathcal{E}_{\mathcal{P}^*}$ such that for every auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, every time bound

[9] We note that [BCI+13] gives more than one preprocessing SNARK construction, but we focus on a particular one.

[10] In previous work, SNARKs satisfying a variant of this property were called SNARKs for NP, in contrast to SNARKs for the universal relation.

$B \in \mathbb{N}$, and every constant $c > 0$,

$$\Pr\left[\begin{array}{cc} \mathcal{V}(\mathsf{crs}, (\langle M\rangle, x, T), \pi) = 1 & \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda, B) \\ T \leq |x|^c & : \ \big((\langle M\rangle, x, T), \pi\big) \leftarrow \mathcal{P}^*(z, \mathsf{crs}) \\ \big((\langle M\rangle, x, T), w\big) \notin \mathcal{R}_{\mathcal{U}} & w \leftarrow \mathcal{E}_{\mathcal{P}^*}(z, \mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We sketch (following [BCCT12, Theorem 5.5]) how to convert a SNARK that is *leveled* adaptive proof of knowledge with computationally unique proofs into a SNARK that is (non-leveled) adaptive proof of knowledge and has computationally unique proofs, under the assumption that there exists an exponentially secure injective one-way function f. That is, we assume there is an $\epsilon > 0$ such that, given $f(x)$ for $x \leftarrow \{0,1\}^n$, no $2^{\epsilon n}$-size adversary can find x with probability better than $2^{-\epsilon n}$.

We will augment the SNARK crs with $f(u^{(1)}), \ldots, f(u^{(\lambda)})$, where each $u^{(i)}$ is sampled uniformly at random from $\{0,1\}^i$. Now, we will require that a valid SNARK for $\big(\langle M\rangle, x, T\big)$ for $2^t \leq T < 2^{t+1}$ must also include $u^{(t)}$. This is feasible for the honest prover with little additional overhead — generating a SNARK would anyways have taken time T, and finding $u^{(t)}$ via brute-force takes time $2^t \cdot \mathsf{poly}(t) \leq \tilde{O}(T)$. This augmentation also clearly preserves the computationally unique proofs property.

On the other hand, let $P^* = \{P^*_\lambda\}$ be any ensemble of polynomial size provers such that for all λ in an infinite set $\Lambda \subseteq \mathbb{Z}^+$, the prover P^*_λ convinces the verifier of a false statement $(\langle M_\lambda\rangle, x_\lambda, T_\lambda)$ with non-negligible probability (where each M_λ, x_λ, and T_λ is a random variable). There are two cases: either there is some $O(\log \lambda)$ bound on t_λ that holds with overwhelming probability, in which case the leveled adaptive proof of knowledge property is applicable, or there is an $\omega(\log \lambda)$ lower bound that holds on t_λ with non-negligible probability for all λ in some infinite set $\Lambda' \subseteq \Lambda$, in which case the exponential security of f is violated.

Thus, in the rest of this section we focus on sketching why the [BCI+13, BCCT12] SNARK has computationally unique proofs.

A.1 Definitions and Assumptions

A linear probabilistically checkable proof (LPCP) of length m is an oracle that computes a linear function $\langle \boldsymbol{\pi}, \cdot \rangle : \mathbb{F}^m \to \mathbb{F}$, so that the answer to each oracle query $\mathbf{q}_i \in \mathbb{F}^m$ is $\langle \boldsymbol{\pi}, \mathbf{q}_i \rangle \in \mathbb{F}$. In what follows, we abuse notation and write such a linear function as $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$. For completeness, we state below the formal definition given in [BCI+13].

Definition A.1 (Linear PCP (LPCP) [BCI+13])**.** *Let $\mathbb{F}$ be a finite field, $\mathcal{P}_{\mathsf{LPCP}}$ be a polynomial-time algorithm, and $\mathcal{V}_{\mathsf{LPCP}}$ be a polynomial-time oracle machine. $(\mathcal{P}_{\mathsf{LPCP}}, \mathcal{V}_{\mathsf{LPCP}})$ is an input-oblivious k-query* linear PCP *for language $\mathcal{L} \in \mathsf{NP}$ over $\mathbb{F}$ with knowledge error ϵ and query length m if the following are satisfied.*

1. **Syntax:** *On any input x and oracle $\boldsymbol{\pi}$, the verifier $\mathcal{V}^{\boldsymbol{\pi}}_{\mathsf{LPCP}}$ makes k "input-oblivious" queries to $\boldsymbol{\pi}$ and then decides whether to accept or reject.*

   That is, $\mathcal{V}_{\mathsf{LPCP}}$ consists of a probabilistic query algorithm $\mathcal{Q}_{\mathsf{LPCP}}$ and a deterministic decision algorithm $\mathcal{D}_{\mathsf{LPCP}}$ working as follows. Based on its internal randomness, and independent of x, $\mathcal{Q}_{\mathsf{LPCP}}$ generates k queries $\mathbf{q}_1, \ldots, \mathbf{q}_k \in \mathbb{F}^m$ to $\boldsymbol{\pi}$ and state information $\mathbf{u}$. Then, given x, $\mathbf{u}$, and the k oracle answers $\langle \boldsymbol{\pi}, \mathbf{q}_1 \rangle, \ldots, \langle \boldsymbol{\pi}, \mathbf{q}_k \rangle$, $\mathcal{D}_{\mathsf{LPCP}}$ accepts or rejects.

   On any input $x \in \mathcal{L}$ with witness w, the prover $\mathcal{P}_{\mathsf{LPCP}}(x, w)$ outputs the description of a linear function $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$.

2. **Completeness:** *For every $x \in \mathcal{L}$ with witness w, the output of $\mathcal{P}_{\mathsf{LPCP}}(x, w)$ is a description of a linear function $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$ such that $\mathcal{V}^{\boldsymbol{\pi}}_{\mathsf{LPCP}}(x)$ accepts with probability 1.*

3. **Knowledge:** *There exists a polynomial-time oracle machine $\mathcal{E}_{\mathsf{LPCP}}$ (which we call a knowledge extractor) such that for every linear function $\boldsymbol{\pi}^* : \mathbb{F}^m \to \mathbb{F}$, if the probability that $\mathcal{V}^{\boldsymbol{\pi}^*}_{\mathsf{LPCP}}$ accepts is greater than ϵ, then $\mathcal{E}^{\boldsymbol{\pi}^*}_{\mathsf{LPCP}}$ outputs w such that w is a witness for $x \in \mathcal{L}$.*

We now give the [BCI+13] definition of a linear interactive proof (LIP). An LIP differs from an LPCP in that the answer to each oracle query is allowed to be an *affine* function of $\mathbf{q}_i$, and furthermore the affine function itself can depend on i (this relaxation applies to both honest and cheating provers). In contrast, an LPCP restricts the oracle responses to use the same coefficients $\boldsymbol{\pi}$ for each $\mathbf{q}_i$ and does not allow affine shifts.

Definition A.2 (Two-message Linear Interactive Proof [BCI+13])**.** *This is defined exactly like an* LPCP*, except that we consider more general linear prover strategies. For a k-query* LIP *over finite field $\mathbb{F}$, we consider affine strategies $\boldsymbol{\Pi}^* : \mathbb{F}^{km} \to \mathbb{F}^k$ where $\boldsymbol{\Pi}^*(x) = \mathbf{A} \cdot x + \mathbf{b}$. $\mathcal{V}_{\mathsf{LIP}}$ consists of a query algorithm $\mathcal{Q}_{\mathsf{LIP}}$ and a decision algorithm $\mathcal{D}_{\mathsf{LIP}}$, where $\mathcal{V}_{\mathsf{LIP}}$ queries its oracle non-adaptively with the concatenation $\mathbf{q} \in \mathbb{F}^{km}$ of k queries $\mathbf{q}_1, \ldots, \mathbf{q}_k \in \mathbb{F}^m$, and receives back $\mathbf{A} \cdot \mathbf{q} + \mathbf{b}$. For the knowledge property, we require that there exists a polynomial-time oracle machine $\mathcal{E}_{\mathsf{LIP}}$ (which we call a knowledge extractor) such that for every affine prover strategy $\boldsymbol{\Pi}^* = (\mathcal{A}, \mathbf{b})$, if the probability that $\mathcal{V}_{\mathsf{LIP}}^{\boldsymbol{\Pi}^*}$ accepts is greater than ϵ, then $\mathcal{E}_{\mathsf{LIP}}^{\boldsymbol{\Pi}^*}$ outputs w such that w is a witness for $x \in \mathcal{L}$.*

We will also require the following knowledge of exponent assumption over bilinear groups stated in [BCI+13] to construct a preprocessing SNARK (note this assumption is essentially identical to the assumptions used in [Gro10, Lip12, GGPR13]). We stress that while this is an extremely strong assumption, its security can be proven in the generic group model [Nec94, Sho97, Mau05].

Assumption A.3 (KEA and poly-power discrete log in bilinear groups [BCI+13])**.** *There exists an efficiently-sampleable group ensemble $\{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$, where for $(\mathbb{G}, \mathbb{G}_T) \in \mathcal{G}_\lambda$, $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order $p \in (2^{\lambda-1}, 2^\lambda)$ having a corresponding efficiently-computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, such that the following properties hold.*

1. **Knowledge of exponent:** *For any polynomial-size adversary $\mathcal{A}$, there exists a polynomial-size extractor $\mathcal{E}$ such that for any "benign" auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and any group element sampler $\mathcal{S}$,*

$$\Pr\left[\begin{array}{c} f' = f^\alpha \\ \Pi_{i \in [t]} g_i^{\pi_i} \neq f \end{array} \;:\; \begin{array}{r} (\mathbb{G}, \mathbb{G}_T) \leftarrow \mathcal{G}_\lambda \\ (g_1, \ldots, g_t) \leftarrow \mathcal{S}(\mathbb{G}, \mathbb{G}_T) \\ \alpha \leftarrow \mathbb{F}_p \\ (f, f') \leftarrow \mathcal{A}(\mathbb{G}, \mathbb{G}_T, g_1, g_1^\alpha, \ldots, g_t, g_t^\alpha; z) \\ (\pi_1, \ldots, \pi_t) \leftarrow \mathcal{E}(\mathbb{G}, \mathbb{G}_T, g_1, g_1^\alpha, \ldots, g_t, g_t^\alpha; z)) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2. **Hardness of poly-power discrete logarithms:** *For any polynomial-size adversary $\mathcal{A}$, polynomial $t = \mathsf{poly}(\lambda)$, and any group generator sampler $\mathcal{S}$ (which is a potentially randomized algorithm that on input a group description $\mathbb{G}$ outputs a group generator g),*

$$\Pr\left[s' = s \;:\; \begin{array}{r} (\mathbb{G}, \mathbb{G}_T) \leftarrow \mathcal{G}_\lambda \\ s \leftarrow \mathbb{F}_p \\ g \leftarrow \mathcal{S}(\mathbb{G}) \text{ where } \langle g \rangle = \mathbb{G} \\ s' \leftarrow \mathcal{A}(\mathbb{G}, \mathbb{G}_T, g, g^s, g^{s^2}, \ldots, g^{s^t}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We remark that our knowledge of exponent property in Assumption A.3 is formulated with respect to a "benign" auxiliary input z. If z can be arbitrary, the results of [BCPR14] show that this assumption is false assuming indistinguishability obfuscation exists. Since the [BCPR14] impossibility embeds an obfuscated program into z, it does not rule out the case where z is drawn from a "benign" distribution (such as a uniform distribution) which does not permit encoding of an obfuscated program. For our applications, this restriction to "benign" distributions suffices. A similar restriction to such distributions is also made in [CFH+15, FFG+16, Gro16, BCC+17].

A.2 A Preprocessing SNARK with Computationally Unique Proofs

In this section, we show that the Bitansky *et al.* [BCI+13] preprocessing SNARK constructed from a variant of the Hadamard-based LPCP of Arora *et al.* [ALM+98] (hereafter the Hadamard PCP) has *computationally*

unique proofs. That is, for every polynomial-size adversary $\mathcal{A}$, there exists a polynomial-size extractor $\mathcal{E}_{\mathcal{A}}$ such that

$$\Pr\left[\begin{array}{c} \mathcal{V}(\tau,x,\pi_1)=1 \\ \mathcal{V}(\tau,x,\pi_2)=1 \\ \pi_1\neq\pi_2 \\ (x,w_1)\notin\mathcal{R}\vee(x,w_2)\notin\mathcal{R}\vee w_1=w_2 \end{array} : \begin{array}{r} \mathsf{crs}\leftarrow G(1^\lambda) \\ (y,\pi_1,\pi_2)\leftarrow\mathcal{A}(\mathsf{crs}) \\ w_1,w_2\leftarrow\mathcal{E}_{\mathcal{A}}(\mathsf{crs}) \end{array}\right]\leq \mathsf{negl}(\lambda).$$

Note that, for simplicity, we have suppressed the time bound B, constant c, and auxiliary input z that appear in Definition 3.2, as they will not be relevant to the discussion here. Also note that as in Definition 3.2, we always assume τ is included in crs since we only consider public verifiability.

At a high level, the [BCI+13] preprocessing SNARK is constructed as follows. First, [BCI+13] give a simplified version of the Hadamard PCP, formulated as an LPCP. Next, they transform this LPCP into an LIP via a generic procedure ([BCI+13, Construction 3.1]). In the final step, they turn the resulting LIP into a preprocessing SNARK by using a special "linear-only encoding scheme." We show that the computationally unique proofs property holds for the original [BCI+13] LPCP and is maintained through the above transformations.

A.2.1 Step 1: An LPCP from Hadamard with Unique Proofs

We proceed to give a quick overview of the LPCP of [BCI+13] based on the Hadamard PCP. The construction is for a relation $\mathcal{R}$ that is decided by an s-gate arithmetic circuit $C:\mathbb{F}^n\times\mathbb{F}^h\to\mathbb{F}^\ell$ in the following sense. For $\mathbf{x}\in\mathbb{F}^n$ and $\mathbf{w}\in\mathbb{F}^h$, $(\mathbf{x},\mathbf{w})\in\mathcal{R}$ iff $C(\mathbf{x},\mathbf{w})=0^\ell$. We let z_i denote the value of the i-th wire of C on input statement $\mathbf{x}\in\mathbb{F}^n$ and witness $\mathbf{w}\in\mathbb{F}^h$, where we order starting at the input wires and ending with the output wires. Let $\mathbf{z}$ denote the length s vector consisting of all s wire values. The proof $\boldsymbol{\pi}$ is simply the length $s+s^2$ vector $(\mathbf{z},\mathbf{z}\otimes\mathbf{z})$, consisting of all wire values z_i and all pairwise products $z_i\cdot z_j$.

The verifier makes three queries $\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3$. The first query asks for $\langle\mathbf{r}_1,\mathbf{z}\rangle$, where $\mathbf{r}_1$ is sampled uniformly at random from $\mathbb{F}^h$. The second query asks for $(\mathbf{r}_1\otimes\mathbf{r}_1)\cdot(\mathbf{z}\otimes\mathbf{z})$. The decision algorithm $\mathcal{D}_{\mathsf{LPCP}}$ checks that the prover's answers to the second query is the square of its answer to the first. If the proof is not of the form $(\mathbf{z},\mathbf{z}\otimes\mathbf{z})$ then, by the Schwartz-Zippel Lemma, with probability $O(1/|\mathbb{F}|)$ the verifier rejects. Thus, we may assume that the proof $\boldsymbol{\pi}$ is of the form $(\mathbf{z},\mathbf{z}\otimes\mathbf{z})$, for some $\mathbf{z}\in\mathbb{F}^s$. Next, the verifier must confirm that the first n entries of $\mathbf{z}$ are equal to the statement $\mathbf{x}$, the last ℓ entries of $\mathbf{z}$ are 0, and for each gate, the output wire is consistent with the input wires. This is all accomplished with one linear query $\mathbf{q}_3$ (see [BCI+13, Section A.1] for a more precise description). Let $\mathbf{r}_{\mathbf{x}}$ be the first n elements of $\mathbf{q}_3$. $\mathbf{r}_{\mathbf{x}}$ is included in the state information $\mathbf{u}$, and the decision algorithm $\mathcal{D}_{\mathsf{LPCP}}$ checks that the prover's answer to the third query is equal to $\langle\mathbf{r}_{\mathbf{x}}\cdot\mathbf{x}\rangle$. Again, by Schwartz-Zippel, if this check passes, then $\mathbf{z}$ begins with $\mathbf{x}$, ends with 0's, and is consistent with all gates of C, with overwhelming probability.

Now we can argue that there exists a special knowledge extractor $\mathcal{E}^*_{\mathsf{LPCP}}$ such that for every statement $\mathbf{x}$ and pair of linear functions $\boldsymbol{\pi}_1,\boldsymbol{\pi}_2$ such that $\boldsymbol{\pi}_1\neq\boldsymbol{\pi}_2$,

$$\Pr\left[\begin{array}{c} \mathcal{D}_{\mathsf{LPCP}}(\mathbf{x},\mathbf{u},\{\langle\boldsymbol{\pi}_1,\mathbf{q}_i\rangle\}_i)=1 \\ \mathcal{D}_{\mathsf{LPCP}}(\mathbf{x},\mathbf{u},\{\langle\boldsymbol{\pi}_2,\mathbf{q}_i\rangle\}_i)=1 \\ (\mathbf{x},\mathbf{w}_1)\notin\mathcal{R}\vee(\mathbf{x},\mathbf{w}_2)\notin\mathcal{R}\vee\mathbf{w}_1=\mathbf{w}_2 \end{array} : \begin{array}{r} (\mathbf{u},\{\mathbf{q}_i\}_i)\leftarrow\mathcal{Q}_{\mathsf{LPCP}} \\ \mathbf{w}_1,\mathbf{w}_2\leftarrow\mathcal{E}^*_{\mathsf{LPCP}}(\mathbf{x},\boldsymbol{\pi}_1,\boldsymbol{\pi}_2) \end{array}\right]\leq\mathsf{negl}(\lambda).$$

The extractor simply returns the $(n+1)$th through $(n+h)$th elements of $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$, respectively. The (informal) arguments above establish that (with overwhelming probability) $\mathcal{D}_{\mathsf{LPCP}}$ will only accept a proof that has the form $(\mathbf{z},\mathbf{z}\otimes\mathbf{z})$, where $\mathbf{z}$ represents a consistent assignment to the circuit $C(\mathbf{x},\cdot)$ that outputs 0^ℓ. Now assume that the extracted $\mathbf{w}_1=\mathbf{w}_2$. Then each proof is of the form $(\mathbf{z},\mathbf{z}\otimes\mathbf{z})$, where $\mathbf{z}$ represents a consistent assignment to all the wires of the circuit $C(\mathbf{x},\mathbf{w})$ that outputs 0^ℓ (in particular implying that $(\mathbf{x},\mathbf{w}_1)\in\mathcal{R},(\mathbf{x},\mathbf{w}_2)\in\mathcal{R}$). However, there is only one consistent assignment once the entire input is fixed, meaning it must be the case that $\boldsymbol{\pi}_1=\boldsymbol{\pi}_2$, a contradiction.

A.2.2 Step 2: An LIP with Unique Proofs

Next, Bitansky *et al.* [BCI+13] show a generic transformation from a k-query LPCP to a $(k+1)$-query LIP. The goal is to force the prover to use the same linear function $\boldsymbol{\pi}$ (with no affine shift) to answer each query $\mathbf{q}_i$, so that any affine prover strategies that do not correspond to valid LPCP prover strategies will fail. This is accomplished by having the verifier's $(k+1)$th query be a random linear combination of $\mathbf{q}_1, \ldots, \mathbf{q}_k$, and having $\mathcal{D}_{\mathsf{LIP}}$ verify that the $(k+1)$th answer is the appropriate linear combination of the first k answers. Again, this is shown to be sound in [BCI+13] via an application of Schwartz-Zippel. In addition, the proof given also suffices to show that there exists a special extractor $\mathcal{E}^*_{\mathsf{LIP}}$ such that for every instance $\mathbf{x}$ and pair of (not necessarily unequal) affine prover strategies $\boldsymbol{\Pi}^*_1 = (\mathbf{A}_1, \mathbf{b_1}), \boldsymbol{\Pi}^*_2 = (\mathbf{A}_2, \mathbf{b_2})$,

$$\Pr\left[\begin{array}{c} \mathcal{D}_{\mathsf{LIP}}(\mathbf{x}, \mathbf{u}, \mathbf{a_1}) = 1 \\ \mathcal{D}_{\mathsf{LIP}}(\mathbf{x}, \mathbf{u}, \mathbf{a_2}) = 1 \\ \mathbf{a_1} \neq \mathbf{a_2} \\ \mathcal{D}_{\mathsf{LPCP}}(\mathbf{x}, \mathbf{u}, \{\langle \boldsymbol{\pi_1}, \mathbf{q_i} \rangle\}_{i \in [k]}) \neq 1 \\ \vee\ \mathcal{D}_{\mathsf{LPCP}}(\mathbf{x}, \mathbf{u}, \{\langle \boldsymbol{\pi_2}, \mathbf{q_i} \rangle\}_{i \in [k]}) \neq 1 \\ \vee\ \boldsymbol{\pi_1} = \boldsymbol{\pi_2} \end{array} : \begin{array}{r} (\mathbf{u}, \mathbf{q}) \leftarrow \mathcal{Q}_{\mathsf{LIP}} \\ \mathbf{a_1} = \mathbf{A}_1 \cdot \mathbf{q} + \mathbf{b_1} \\ \mathbf{a}_2 = \mathbf{A}_2 \cdot \mathbf{q} + \mathbf{b_2} \\ \mathbf{q} = (\mathbf{q_1} \| \mathbf{q_2} \| \ldots \| \mathbf{q_{k+1}}) \\ \boldsymbol{\pi_1}, \boldsymbol{\pi_2} \leftarrow \mathcal{E}^*_{\mathsf{LIP}}(\mathbf{x}, (\mathbf{A}_1, \mathbf{b_1}), (\mathbf{A}_2, \mathbf{b_2})) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

In particular, [BCI+13] show that if the prover strategy $(\mathbf{A}, \mathbf{b})$ causes the verifier $\mathcal{V}_{\mathsf{LIP}}$ to accept with answers $\mathbf{a}$, then with overwhelming probability, it is possible to extract a $\boldsymbol{\pi}$ such that $\mathbf{a}_i = \langle \boldsymbol{\pi}, \mathbf{q}_i \rangle$ for all $i \in [k]$, and that $\boldsymbol{\pi}$ causes $\mathcal{D}_{\mathsf{LPCP}}$ to accept with queries $\{\mathbf{q}_i\}_{i \in [k]}$. If $\mathbf{a_1} \neq \mathbf{a_2}$, then clearly the extracted $\boldsymbol{\pi_1}, \boldsymbol{\pi_2}$ must be unequal.

A.2.3 Step 3: A Preprocessing SNARK with Unique Proofs

Finally, Bitansky *et al.* [BCI+13] compile the Hadamard-based LIP into a preprocessing SNARK by using a special "linear-only one way encoding" scheme, which has two (deterministic) modes of encryption Enc and SEnc, where Enc is a *linear-only* mode of encoding and SEnc is a *standard* mode of encoding. Roughly, such a scheme is linearly homomorphic with respect to the Enc encodings, and additionally guarantees that if an adversary $\mathcal{A}$ is given some set of encodings $\mathsf{Enc}(a_1), \ldots, \mathsf{Enc}(a_m), \mathsf{SEnc}(\widetilde{a}_1), \ldots, \mathsf{SEnc}(\widetilde{a}_{\widetilde{m}})$, and produces new encodings $\mathsf{Enc}(a^*_1), \ldots, \mathsf{Enc}(a^*_{m^*})$, it is possible to extract from $\mathcal{A}$ an affine transformation $(\mathbf{A}, \mathbf{b})$ such that $(a^*_1, \ldots, a^*_{m^*})^\top = \mathbf{A} \cdot (a_1, \ldots, a_m)^\top + \mathbf{b}$. Bitansky *et al.* [BCI+13] provide a construction of linear-only one way encoding scheme from Assumption A.3.

At a high level, the SNARK crs consists of Enc encodings of the set of LIP verifier queries $\mathbf{q}$ and SEnc encodings of the LIP verifier state $\mathbf{u}$. The SNARK prover $\mathcal{P}$ uses the linear homomorphism property of the encoding scheme to compute answers to the queries $\mathbf{q}$, and the SNARK verifier $\mathcal{V}$ uses the decision algorithm $\mathcal{D}_{\mathsf{LIP}}$ and a special zero-test property of the encoding scheme to verify the proof. Now it follows from properties of the encoding scheme that for every polynomial-size prover $\mathcal{A}$, there exists a polynomial-size extractor $\mathcal{E}^*$ such that

$$\Pr\left[\begin{array}{c} \mathcal{V}(\tau, x, \pi_1) = 1 \\ V(\tau, x, \pi_2) = 1 \\ \pi_1 \neq \pi_2 \\ \mathcal{D}_{\mathsf{LIP}}(x, \tau, \mathbf{a_1}) \neq 1 \vee \mathcal{D}_{\mathsf{LIP}}(x, \tau, \mathbf{a_2}) \neq 1 \vee \mathbf{a_1} = \mathbf{a_2} \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow G(1^\lambda) \\ (x, \pi_1, \pi_2) \leftarrow \mathcal{A}(\mathsf{crs}) \\ (\mathbf{A}_1, \mathbf{b_1}), (\mathbf{A}_2, \mathbf{b_2}) \leftarrow \mathcal{E}^*(x, \pi_1, \pi_2) \\ \mathbf{a_1} = \mathbf{A}_1 \cdot \mathbf{q} + \mathbf{b_1}, \mathbf{a_2} = \mathbf{A}_2 \cdot \mathbf{q} + \mathbf{b_2} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

The soundness proof in [BCI+13] shows that for any prover that produces an accepting π, there is an extractor that produces $(\mathbf{A}, \mathbf{b})$ such that $\pi = \mathsf{Enc}(\mathbf{A} \cdot \mathbf{q} + \mathbf{b})$, and $(\mathbf{A}, \mathbf{b})$ is a perfectly convincing LIP prover strategy. This means that $\mathbf{a_1}$ and $\mathbf{a_2}$ will both cause $\mathcal{D}_{\mathsf{LIP}}$ to accept, and furthermore, $\pi_1 \neq \pi_2$ implies that $\mathbf{a_1} \neq \mathbf{a_2}$ since Enc is deterministic and injective.

A.3 Removing Preprocessing from SNARKs with Unique Proofs

The above construction only gives a preprocessing SNARK with computationally unique proofs, while we require a *fully-succinct* SNARK with computationally unique proofs for Theorem 3.5. To accomplish this, we

rely on the work of Bitansky *et al.* [BCCT13] who gave a generic transformation from a publicly verifiable preprocessing SNARK to a publicly verifiable fully-succinct SNARK. In this section, we outline why this generic transformation preserves computationally unique proofs. For the sake of completeness, we first recall some of the definitions given in [BCCT13] for notions of distributed computation, proof-carrying data, and compliance. The following Appendix A.3.1 is taken verbatim from [BCCT13].

A.3.1 [BCCT13] Background

A distributed computation is viewed as a directed acyclic graph $G = (V, E)$ with node labels $\mathsf{linp} : V \to \{0,1\}^*$ and edge labels $\mathsf{data} : E \to \{0,1\}^*$. The node label $\mathsf{linp}(v)$ of a node v represents the *local input* (which may include a local program) used by v in its local computation. (Whenever v is a source or a sink, we require that $\mathsf{linp}(v) = \perp$.) The edge label $\mathsf{data}(u, v)$ of a directed edge (u, v) represents the *message* sent from node u to node v. Typically, a party at node v uses the local input $\mathsf{linp}(v)$ and input messages $(\mathsf{data}(u_1, v), \ldots, \mathsf{data}(u_c, v))$ where $u_1, \ldots, u_c$ are the parents of v in lexicographic order, to compute an output message $\mathsf{data}(v, w)$ for a child node w; the party also similarly computes a message for every other child node. We can think of the messages of edges going out from sources as the "inputs" to the distributed computation, and the messages on edges going into sinks as the "ouptuts" of the distributed computation; for convenience we will want to identify a single distinguished output.

Definition A.4. *A* **(distributed computation) transcript** *is a triple* $\mathsf{T} = (G, \mathsf{linp}, \mathsf{data})$*, where* $G = (V, E)$ *is a directed acyclic graph,* $\mathsf{linp} : V \to \{0,1\}^*$ *are node labels, and* $\mathsf{data} : E \to \{0,1\}^*$ *are edge labels; we require that* $\mathsf{linp}(v) = \perp$ *whenever* v *is a source or a sink. The output of* T*, denoted* $\mathsf{out}(\mathsf{T})$*, is equal to* $\mathsf{data}(\widetilde{u}, \widetilde{v})$*, where* $(\widetilde{u}, \widetilde{v})$ *is the lexicogaphically first edge such that* $\widetilde{v}$ *is a sink.*

A proof-carrying transcript is a transcript where messages are augmented by proof strings, i.e., a function $\mathsf{proof} : E \to \{0,1\}^*$ provides for each edge (u, v) an additional label $\mathsf{proof}(u, v)$, to be interpreted as a proof string for the message $\mathsf{data}(u, v)$.

Definition A.5. *A* proof-carrying **(distributed computation) transcript** PCT *is a pair* $(\mathsf{T}, \mathsf{proof})$ *where* T *is a transcript and* $\mathsf{proof} : E \to \{0,1\}^*$ *is an edge label.*

Next, we define what it means for a distributed computation to be *compliant*, which is the notion of "correctness with respect to a given local property". Compliance is captured via an efficiently-computable *compliance predicate* $\mathbb{C}$, which must be locally satisfied at each vertex; here, "locally" means with respect to a node's local input, incoming data, and outgoing data. For convenience, for any vertex v, we let $\mathsf{children}(v)$ and $\mathsf{parents}(v)$ be the vector of v's children and parents respectively, listed in lexicographic order.

Definition A.6. *Given a polynomial-time predicate* $\mathbb{C}$*, we say that a distributed computation transcript* $\mathsf{T} = (G, \mathsf{linp}, \mathsf{data})$ *is* **$\mathbb{C}$-compliant** *(denoted by* $\mathbb{C}(\mathsf{T}) = 1$*) if, for every* $v \in V$ *and* $w \in \mathsf{children}(v)$*, it holds that*

$$\mathbb{C}(\mathsf{data}(v, w); \mathsf{linp}(v), \mathsf{inputs}(\mathsf{v})) = 1,$$

where $\mathsf{inputs}(v) := (\mathsf{data}(u_1, v), \ldots, \mathsf{data}(u_c, v))$ *and* $(u_1, \ldots, u_c) := \mathsf{parents}(v)$*. Furthermore, we say that a message* $\mathbf{z}$ *is* $\mathbb{C}$*-compliant if there is* T *such that* $\mathbb{C}(\mathsf{T}) = 1$ *and* $\mathsf{out}(\mathsf{T}) = 1$ *.*

Definition A.7. *Given a distributed computation transcript* $\mathsf{T} = (G, \mathsf{linp}, \mathsf{data})$ *and any edge* $(v, w) \in E$*, we denote by* $t_{\mathsf{T},\mathbb{C}}(v, w)$ *the time required to evaluate* $\mathbb{C}(\mathsf{data}(v, w); \mathsf{linp}(v), \mathsf{inputs}(v))$*. We say that* T *is* **B-bounded** *if* $t_{\mathsf{T},\mathbb{C}}(v, w) \leq B$ *for every edge* (v, w)*.*

Definition A.8. *The* **depth of a transcript** T*, denoted* $\mathsf{d}(\mathsf{T})$*, is the largest number of nodes on a source-to-sink path in* T *minus 2 (to exclude the source and the sink). The* **depth of a compliance predicate** $\mathbb{C}$*, denoted* $\mathsf{d}(\mathbb{C})$*, is defined to be the maximum depth of any transcript* T *compliant with* $\mathbb{C}$*.*

Definition A.9. *A* **proof-carrying data (PCD) system** *for a class of compliance predicates* $\boldsymbol{C}$ *is a triple of algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *that works as follows:*

- *The (probabilistic) generator $\mathcal{G}$, on input the security parameter λ and time bound B, outputs a* reference string σ *which includes a* verification state τ *(we only consider public verifiability).*
- *For any $\mathbb{C} \in \mathbf{C}$, the (honest) prover $\mathcal{P}_{\mathbb{C}} := \mathcal{P}(\mathbb{C}, \dots)$ is given a reference string σ, inputs $\vec{\mathsf{z}}_i$ with corresponding proofs $\vec{\pi}_i$, a local input linp, and an output z_o, and then produces a proof π_o attesting to the fact that z_o is consistent with some $\mathbb{C}$-compliant transcript.*
- *For any $\mathbb{C} \in \mathbf{C}$, the verifier $\mathcal{V}_{\mathbb{C}} := \mathcal{V}(\mathbb{C}, \dots)$ is given the verification state τ, an output $\vec{\mathsf{z}}_i$, and a proof string π_o, and accepts if it is convinced that z_o is consistent with some $\mathbb{C}$-compliant transcript.*

After the generator $\mathcal{G}$ has been run to obtain σ, an interactive protocol $\mathsf{ProofGen}(\mathbb{C}, \sigma, S, \mathcal{P})$ is run, where S samples a graph and corresponding transcript T, and $\mathcal{P}$ iteratively provides proofs for the $\mathbb{C}$-compliance of T. This process outputs a triple $(\mathsf{z}_o, \pi_o, \mathsf{T})$ where z_o is the output message with corresponding proof π_o.

We require that the PCD system satisfy the following:

1. **Completeness:** *For every compliance predicate $\mathbb{C} \in \mathbf{C}$ and (possibly unbounded) distributed computation generator S,*

$$\Pr\left[\begin{array}{c|r} \mathsf{T} \text{ is } B\text{-bounded} & \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ \mathbb{C}(\mathsf{T}) = 1 & (\mathsf{z}_o, \pi_o, \mathsf{T}) \leftarrow \mathsf{ProofGen}(\mathbb{C}, \sigma, S, \mathcal{P}) \\ \mathcal{V}_{\mathbb{C}}(\tau, \mathsf{z}_o, \pi_o) \neq 1 & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2. **Proof of Knowledge:** *For every polynomial-size prover $\mathcal{P}^*$ there exists a polynomial-size extractor $\mathcal{E}_{\mathcal{P}^*}$ such that for every compliance predicate $\mathbb{C} \in \mathbf{C}$, every large enough λ, every auxiliary input $x \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and every time bound $B \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c|r} \mathcal{V}_{\mathbb{C}}(\tau, \mathsf{z}, \pi) = 1 & \sigma \leftarrow \mathcal{G}(1^\lambda, B) \\ (\mathsf{out}(\mathsf{T}) \neq \mathsf{z} \vee \mathbb{C}(\mathsf{T}) \neq 1) & (\mathsf{z}, \pi) \leftarrow \mathcal{P}^*(\sigma, z) \\ & \mathsf{T} \leftarrow \mathcal{E}_{\mathcal{P}^*}(\sigma, z) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

3. **Efficiency:** *There exists a universal polynomial p such that, for every compliance predicate $\mathbb{C} \in \mathbf{C}$, every large enough security parameter $\lambda \in \mathbb{N}$, every time bound $B \in \mathbb{N}$, and every B-bounded distributed computation transcript T,*
   - *the generator $\mathcal{G}(1^\lambda, B)$ runs in time* $\begin{cases} p(\lambda + B) & \text{for a preprocessing PCD} \\ p(\lambda + \log(B)) & \text{for a fully-succinct PCD} \end{cases}$,
   - *the prover $\mathcal{P}_{\mathbb{C}}(\sigma, \mathsf{data}(v, w), \mathsf{linp}(v), \mathsf{inputs}(v), \vec{\pi}_i)$ runs in time*

$$\begin{cases} p(\lambda + |\mathbb{C}| + t_{\mathsf{T},\mathbb{C}}(v, w) + B) & \text{for a preprocessing PCD} \\ p(\lambda + |\mathbb{C}| + t_{\mathsf{T},\mathbb{C}}(v, w) + \log(B)) & \text{for a fully-succinct PCD} \end{cases},$$

   - *the verifier $\mathcal{V}_{\mathbb{C}}$ runs in time $p(\lambda + |\mathbb{C}| + |\mathsf{z}| + \log(B))$,*
   - *and an honestly generated proof has size $p(\lambda + \log(B))$.*

*We shall also consider a restricted notion of PCD system: a **path PCD system** is a PCD system where completeness is guaranteed to hold only for distributed computation transcripts T whose graph is a line.*

A.3.2 A Fully Succinct SNARK with Computationally Unique Proofs

Constant-depth PCD. Bitansky *et al.* [BCCT13] first show that a preprocessing SNARK can be used to construct a *constant-depth* preprocessing PCD system. At each node v of the graph, the prover takes as input the PCD crs σ, the output message $\mathsf{z}_v = \mathsf{data}(v, w)$ where w is any child of v, the local input $\mathsf{linp}(v)$, the input vector $\vec{\mathsf{z}}_i = \{\mathsf{data}(u_i, v)\}_{u_i \in \mathsf{parents}(v)}$, and corresponding proofs $\vec{\pi}_i$. It invokes the SNARK prover to generate an outgoing proof π_v attesting to the fact that

- $\mathbb{C}(\mathsf{z}_v; \mathsf{linp}(v), \vec{\mathsf{z}}_i) = 1$; that is, the computation occurring at node v is $\mathbb{C}$-compliant, and
- the SNARK verifier accepts each proof π in $\vec{\pi}_i$ as a valid proof of the $\mathbb{C}$-compliance of the corresponding z in $\vec{\mathsf{z}}_i$.

The constant depth restriction arises from the recursive proof that establishes proof of knowledge, which may blow up the size of the extractor by a $\mathsf{poly}(\lambda)$ factor at each step.

We argue that if the preprocessing SNARK has computationally unique proofs, then the resulting constant-depth preprocessing PCD has a similar "computational unique transcript" property. That is, for any prover that produces two outgoing proofs $(\pi_o^{(1)}, \mathsf{z}_o^{(1)})$ and $(\pi_o^{(2)}, \mathsf{z}_o^{(2)})$ such that $(\pi_o^{(1)}, \mathsf{z}_o^{(1)}) \neq (\pi_o^{(2)}, \mathsf{z}_o^{(2)})$, there is an extractor that produces two compliant transcripts $\mathsf{T}_1, \mathsf{T}_2$ such that $\mathsf{T}_1 \neq \mathsf{T}_2$.

Recall that a transcript T consists of a set of messages and local inputs $\{\mathsf{data}(v), \mathsf{linp}(v)\}_{v \in V}$, one for each node of a graph $G = (V, E)$. If $\mathsf{z}_o^{(1)} \neq \mathsf{z}_o^{(2)}$, then already $\mathsf{T}_1 \neq \mathsf{T}_2$ (since each z is a message), so assume otherwise, meaning that $\pi_o^{(1)} \neq \pi_o^{(2)}$. The witness extracted by the SNARK extractor given any outgoing proof π from any node v accompanying some message z will consist of $(\mathsf{linp}(v), \vec{\mathsf{z}}_i, \vec{\pi}_i)$, where $\vec{\mathsf{z}}_i$ is the set of input messages to v and $\vec{\pi}_i$ is the set of accompanying proofs. So the computationally unique proofs property of the SNARK implies that producing two different outgoing proofs from any node requires knowledge of two different witnesses $(\mathsf{linp}(v)^{(1)}, \vec{\mathsf{z}}_i^{(1)}, \vec{\pi}_i^{(1)}) \neq (\mathsf{linp}(v)^{(2)}, \vec{\mathsf{z}}_i^{(2)}, \vec{\pi}_i^{(2)})$. Now for each node in the graph, always assume that the extracted $(\mathsf{linp}(v)^{(1)}, \vec{\mathsf{z}}_i^{(1)}) = (\mathsf{linp}(v)^{(2)}, \vec{\mathsf{z}}_i^{(2)})$, since otherwise $\mathsf{T}_1 \neq \mathsf{T}_2$. By recursing backwards from the sink node, this implies that there will be some $\pi_1 \neq \pi_2$ accompanying the same message coming out of a source node x. However, the only witness satisfying the relation (is $\mathbb{C}$-compliant) at a source node is $(\mathsf{linp}(x), \vec{\mathsf{z}}_i, \vec{\pi}_i) = \perp$, so producing the two different proofs $\pi_1 \neq \pi_2$ at x violates the computationally unique proofs property of the SNARK.

RAM checking. Recall that the goal is to construct a SNARK for NP, which can prove membership of an instance $y = (\langle M \rangle, x, t)$ in the universal language. Bitansky *et al.* [BCCT13] take inspiration from *incrementally-verifiable computation* [Val08], and decompose the RAM computation $M(x)$ into a sequence of steps, each of which can be locally verified by checking consistency with the transition function of M. This computation is potentially expensive if M uses a large amount of memory. Thus, [BCCT13] use a result of [BCGT13], which states that, assuming collision resistant hashing, membership of an instance $y = (\langle M \rangle, x, t)$, of the universal language, can be computationally reduced to membership of an instance $y' = (\langle M' \rangle, x, t')$, where the space complexity of M' is bounded by a fixed polynomial $p(\lambda)$, and $t' = t \cdot \mathsf{poly}(\lambda)$. The idea is originally from [BEG+91] in the context of *online memory checking*, and it consists of maintaining and updating a Merkle hash of M's memory. At a high level, given a collision resistant hash h, extending a witness w to $(\langle M \rangle, x, t)$ into a witness w' for $(\langle M' \rangle, x, t')$ for the locally efficient instance consists of appending a sequence of Merkle openings under h.

We argue that no polynomial size adversary can find two different accepting witnesses w_1' and w_2' for the instance $(\langle M' \rangle, x, t')$ with the same prefix w (which would be the witness to the original RAM instance $(\langle M \rangle, x, t)$). In particular, if M is about to load from or store to a memory location i, then M' (who has the Merkle root stored, which was initially computed from a memory with all 0's) will access its w' tape and expect to see a Merkle opening for location i. It will immediately reject if the opening is not consistent with h. Note that on a store operation, M' will update its root accordingly. Thus in order to break the above claim, an adversary must find two Merkle openings for the same memory location i, breaking collision resistance of h.

Constant- to Polynomial-Depth PCD. By the arguments above, it suffices to construct a SNARK with computationally unique proofs for locally efficient RAM computation. Bitansky *et al.* [BCCT13] simply make use of a polynomial depth PCD system, where each node locally checks one step of the RAM computation. The local efficiency means that the time bound B of the PCD system can be set to a fixed polynomial $p(\lambda)$, which means that even using a preprocessing PCD results in a fully-succinct SNARK. Note that the transcript T of the PCD system consists entirely of information about the instance (M', x, t') and witness

w', so it suffices to have a polynomial-depth (preprocessing) PCD system with computationally unique transcripts. Above, we saw that a SNARK with computationally unique proofs gives rise to a *constant-depth* preprocessing PCD system with computationally unique transcripts.

Bitansky *et al.* [BCCT13] show that a PCD system for constant-depth compliance predicates can be used to construct a path PCD system for polynomial-depth compliance predicates. Here we describe the special case of their transformation corresponding to when the polynomial-depth PCD system is verifying a (locally efficient) RAM computation, which is sufficient for our purposes. The graph corresponding to the constant-depth PCD is a tree, with arity equal to the security parameter λ, where the leaves consist of the polynomial-depth computation path, and the internal nodes check that timestamps and execution states of their subtrees match appropriately. In more detail, let τ_i denote timestamps and S_i denote execution states of the RAM machine M'. Each leaf node takes an input message $((0, \tau_1, S_1), (0, \tau_2, S_2))$ and outputs $(1, \tau, S, \tau', S')$. The compliance predicate verifies that $\tau_2 = \tau_1 + 1$, $\tau = \tau_1$,$\tau' = \tau_2$, $S = S_1$, $S' = S_2$ and that S_2 follows S_1 in the execution of M' on input x. An internal tree node at height d takes an input message $((d, \tau_i, S_i, \tau'_i, S'_i))_{i=1}^{\lambda}$ and outputs $(d + 1, \tau, S, \tau', S')$. The compliance predicate verifies that $\tau'_{i-1} = \tau_i$ and $S'_{i-1} = S_i$ for all $i \in [\lambda]$, and that $\tau = \tau_1, \tau' = \tau'_\lambda, S = S_1, S' = S'_\lambda$.

Now, it is clear that given the inputs to the leaf nodes (corresponding to the sequence of RAM states during computation, along with their timestamps), the rest of the messages in this PCD system are fixed. Thus, two different transcripts for the constant-depth PCD system must correspond to two different transcripts for the polynomial-depth RAM checking PCD system. This completes the argument that [BCCT13] bootstrapping maintains the computationally unique proofs property (assuming collision resistant hashing).

B Error Correcting Codes

We give a brief reminder on error correcting codes.

Definition B.1 (Error correcting code)**.** *An* error correcting code C *over alphabet* Σ *is an injective function* $C : \Sigma^k \to \Sigma^n$*, where* k *is the message length and* n *is the block length. The* rate *of* C *is* $\frac{k}{n}$*, and its* relative distance *is* $\min_{x \neq y}\big(\Delta(C(x), C(y))\big)$*.*

Definition B.2 (Code ensemble)**.** *A* code ensemble C *over alphabet* Σ *is an ensemble* $C \triangleq \{C_k\}_{k \in \mathbb{N}}$ *s.t.* C_k *is a code for messages in* Σ^k *for every* $k \in \mathbb{N}$*.*

The ensemble C *has* rate ρ *(resp.* relative distance δ*) if for every* k*, the code* C_k *has rate* ρ *(resp. relative distance* δ*). We say that* C *is* efficiently encodable *(resp.* efficiently decodable*) if there exists a polynomial-time algorithm Encode (resp. Decode), s.t. for all* $k \in \mathbb{N}$ *and* $x \in \Sigma^k$ *holds that* $Encode(x) = C_k(x)$ *(resp.* $Decode(C_k(x)) = x$*).*

Codes that are efficiently encodable and decodable are known. In particular:[11]

Theorem B.3 (e.g., [Spi96])**.** *There exists an efficiently encodable and decodable binary code ensemble with constant relative distance and constant rate.*

[11] Note that the codes provided by [Spi96] are actually much stronger than what is actually needed for our results.