Indistinguishable Proofs of Work or Knowledge

Foteini Baldimtsi [1,2], Aggelos Kiayias [1], Thomas Zacharias [1], and Bingsheng Zhang [1,3]

[1]Dept. of Informatics and Telecommunications, University of Athens, Greece
[2]Boston University, USA
[3]Security Lancaster Research Centre, Lancaster University, UK ,
{bfoteini, aggelos, thzacharias, bzhang}@di.uoa.gr

Abstract

We introduce a new class of protocols called *Proofs of Work or Knowledge* (PoWorKs). In a PoWorK, a prover can convince a verifier that she has either performed work or that she possesses knowledge of a witness to a public statement *without* the verifier being able to distinguish which of the two has taken place.

We formalize PoWorK in terms of three basic properties, completeness, f-soundness and indistinguishability (where f is a function that determines the tightness of the proof of work aspect) and present a construction that transforms 3-move HVZK protocols into 3-move public-coin PoWorKs. To formalize the work aspect in a PoWorK protocol we define cryptographic puzzles that adhere to certain uniformity conditions, which may also be of independent interest. We instantiate our puzzles in the random oracle (RO) model as well as via constructing "dense" versions of suitably hard one-way functions.

We then showcase PoWorK protocols by presenting two applications. We first show how non-interactive PoWorKs can be used to *reduce spam email* by forcing users sending an e-mail to either prove to the mail server they are approved contacts of the recipient or to perform computational work. As opposed to previous approaches [DN92, DGN03] that applied proofs of work to this problem, our proposal of using PoWorKs is privacy-preserving as it hides the list of the receiver's approved contacts from the mail server. Our second application for PoWorK relates to zero-knowledge protocols. We show that PoWorK protocols imply straight-line quasi-polynomial simulatable arguments of knowledge; by applying this result to our construction we obtain an efficient straight-line concurrent 3-move statistically quasi-polynomial simulatable argument of knowledge, improving the round complexity of the previously known four-move protocols, [Pas03].

Keywords: proof of work, cryptographic puzzle, concurrent zero-knowledge, dense one-way functions.

1 Introduction

We introduce a new class of prover verifier protocols where the prover wishes to convince the verifier that it it is either in possession of a witness to a publicly known statement or that it has invested a certain amount of computational effort. A *Proof of Work or Knowledge* (PoWorK) enables the prover to achieve this objective while at the same time ensuring that the verifier is incapable of distinguishing which way the prover has followed : performing the work or exploiting her knowledge of the witness.

At an intuitive level a PoWorK protocol is a disjunction of a *proof of work* and a *proof of knowledge*. Proofs of knowledge are a fundamental notion in cryptography [GMR85] with a very wide array of applications in the design of cryptographic protocols. They have been studied extensively, both in terms of efficient constructions, e.g., [Sch89], as well as in terms of their composability with themselves or within larger protocols, see e.g., [DNS98, CGGM00, Can01, CF01, Pas03, Pas04]. Proofs of work on the other hand, were first introduced in [DN92], further studied in [RSW96, Bac97, JB99, DGN03, CMSW09], and were primarily applied as a denial of service network or spam protection mechanism; recently they have also found important applications in building decentralized cryptocurrencies (notably bitcoin [Nak08] but also many others).

In an interactive proof protocol, we are interested primarily in two basic properties, soundness and zero-knowledge, that represent the adversarial objectives of the prover and the verifier respectively: the prover must not be able to convince the verifier of false statements while the verifier should not extract any knowledge from interacting with the prover beyond what can be inferred by the public statement. An important class of prover verifier protocols is the 3-move honest-verifier zero knowledge (HVZK) protocols. They are three-move protocols that are "public-coin", i.e., the verifier in the second move merely selects a random value (that is drawn independently to the statement of the prover's first move) and submits it to the prover. 3-move HVZK protocols capture a very wide class of practical proofs of knowledge (including Schnorr's identification scheme [Sch89]) but also all language in $\mathcal{NP}$ can be shown with a (computational) HVZK protocol via reduction to e.g., the Hamilton cycle protocol [Blu87].

Given the above, one may construct a PoWorK protocol for a language $\mathcal{L}$ as follows: the verifier samples a cryptographic puzzle, puz, and submits it to the prover. The prover provides a commitment ψ and shows that she either possesses a witness w showing that the statement x belongs to $\mathcal{L}$ or that the commitment ψ contains a solution to puz. It is easy to prove that this is a general four-move protocol that implements a PoWorK for any language $\mathcal{L}$ and any cryptographic puzzle. On the other hand, it is known that for zero-knowledge proofs, two-round protocols do not exist for non-trivial languages [GO94] and this result remains true even if the zero-knowledge property is relaxed to $O(\lambda^{\log^c(\lambda)})$-simulatability [Pas03], in the sense that only languages decidable in quasi-polynomial time may have two-round quasi-polynomial-time simulatable protocols.

1.1 Our results.

We construct efficient *three-move* PoWorK protocols and we demonstrate how they can instantiate systems that reduce email spam while preserving user privacy, and how they can give rise to concurrent simulatable protocols. In more details:

1.1.1 Definition of PoWorKs.

Our formalization entails two definitions, f-soundness and (statistical) indistinguishability. In f-soundness we require that any prover that has running time (in number of steps) less than a specified parameter calibrated according to the function f of the running time of the puzzle solver, it is guaranteed to lead to a knowledge extractor. The importance of the function f is to provide a safe running time upper bound under which the complete protocol execution is successful only via an (a-priori) knowledge of the witness. Indistinguishability on the other hand, ensures that a malicious verifier is incapable of discerning whether the prover performs the proof of work or possesses the knowledge of the witness. We note that timing issues are not taken into account in our model (i.e., we assume that the prover always takes the same amount of time to finish no matter which one of the two strategies it follows).

What we do care about though, is that the prover who performs a proof of work spends at least a certain amount of computational resources. Note that indistinguishability easily implies witness indistinguishability [FS90], and thus any PoWorK is also a witness indistinguishable protocol.

1.1.2 PoWorK Constructions.

We present a three-move public-coin protocol instantiating a PoWorK given any 3-move HVZK protocol with special soundness. Our protocol transformation preserves the structure and round complexity of the given 3-move HVZK protocol. Observe that the verifier cannot simply provide a puzzle challenge since this would violate the public-coin characteristic of the protocol. To achieve our construction we require puzzle generation algorithms that have a suitable uniformity characteristics, specifically, we require that the domain of puzzles (the "puzzle space") and the challenge space of the 3-move HVZK protocol are statistically very close (in terms of the distributions induced by the puzzle sample algorithm and the verifier in the protocol). Given such suitable puzzle distribution we present a protocol where the prover is capable of generating a puzzle solution on the fly (utilizing the verifier's public coins) and solve it if she wishes. To establish the practicality of our approach we also construct puzzles that are "dense" within $\{0,1\}^l$ and hence consistent with the challenge space of many natural 3-move HVZK protocols. Our dense puzzle based PoWorK construction has the characteristic that is *black-box* with respect to the underlying puzzle system (which is suitable for puzzles whose security is argued, say, in the Random Oracle model). We also provide a less efficient 3-move PoWorK that works for general puzzles based on the [LS90] protocol; however this construction is non-black-box w.r.t. the puzzle (i.e., it needs to know the program of puzzle verification).

1.1.3 Definition and instantiations of puzzles.

We give formal definitions of cryptographic puzzle systems PuzSys that are easy to generate, hard to solve, and easy to verify. We define additional properties like density and amortization resistance and we give two instantiations. Our first instantiation utilizes the random oracle model [BR93] while the second relies on complexity assumptions. More specifically, we use *Universal One Way Hash Function* families (UOWHF) [NY89] to build extractors with special properties, invoking a variant of leftover hash lemma [Dod05]. We then combine this special extractor with suitably hard one-way functions to obtain our second puzzle instantiation; we present an instantiation of this methodology for the discrete-logarithm problem. As an intermediate result, which may be of independent interest, we show how to convert any arbitrary oneway function to a "dense" oneway function over $\{0,1\}^{\ell(\lambda)}$ for some $\ell(\cdot)$ and security parameter $\lambda \in \mathbb{Z}^+$ (cf. Theorem 4).

Our puzzle definitions are close in spirit to previous formalizations [RSW96, WJHF04, CMSW09, MMV11, BGJ+16] with the following distinctions. [CMSW09], defines the hardness of a puzzle as a monotonically increasing function that maps the running time of an adversary to the success rate of solving the puzzle. Contrary to this, our definition, motivated by our proof of knowledge application, imposes a sharp time threshold, below which the success rate of solving a puzzle becomes negligible. Also, contrary to time-lock puzzles [RSW96, WJHF04, MMV11, BGJ+16], we do not restrict the parallelizability of our puzzles as such feature does not hurt (and may even be desirable) in the PoWorK context. Parallelizable puzzles, like the ones we are focusing on here, have become very popular by their applications on cryptocurrencies. The requirement there is that the puzzle solver should spend a minimum of computational resources to find a solution to the puzzle.

1.1.4 Applications.

Generally speaking, PoWorKs can be used in applications where we would like to allow access to either "registered" or "approved" users (who know a witness) or to every user who is willing to invest computational effort. The key property of PoWorKs is that they enhance privacy since they do not leak the type of user (i.e. approved or not) to the entity that verifies access. A nice illustration of this type of application of PoWorKs is in regard to *reducing* *spam email*. Dwork and Naor proposed using proofs of work to control spam e-mails [DN92]. The gist of the idea is that every non-approved contact of a receiver would have to perform some work (i.e. invest computational effort) in order to send her an email. A downside of the method is that the mail server has to maintain an updated list of "approved-contacts" for every user; this can be a privacy concern for the users (not to mention the cost of updating the approved contacts database). We show how by using PoWorK's, one can still enforce the non-approved senders to perform work while preserving user privacy, since the mail server (who acts as a PoWorK verifier) will not be able to distinguish between approved and non-approved contacts because of PoWorK indistinguishability property.

Our second application relates to zero-knowledge protocols and concerns quasi-polynomial time straight-line simulatable arguments of knowledge. This class of protocols was introduced by [Pas03] and was motivated by the construction of concurrent zero-knowledge proofs in the plain model (as opposed to using a "setup" assumption). In [Pas03] a four-move argument of knowledge was presented that is quasi-polynomial time simulatable. We show that any suitable PoWorK protocol (see Theorem 1 for the precise formulation) implies quasi-polynomial time straight-line simulatable arguments of knowledge. Together with our PoWorK constructions this improves the four-move round complexity of the protocol given in [Pas03] and has optimal round complexity (due to the impossibility of two-round protocols in the same setting, at least for languages that are not decidable in quasi-polynomial time). Note that in [Pas03] a two-move protocol is also presented using Zaps [DN00] however this protocol is not an argument of knowledge.

Concurrent work. Concurrent to our work, [CPS$^+$15] introduced an efficient OR composition technique that can be used with "input-delayed" Σ-protocols, i.e., protocols where the statement need not be determined ahead of time. They also observed that their construction can be used to reduce the round complexity of straight-line perfect quasi-poly-time simulatable arguments of knowledge from 4 rounds to 3. In relation to PoWorKs, we observe that in the case that a puzzle accepts an input-delayed Σ proof of knowledge of the puzzle solution (e.g., the DLP based puzzle) one may use the OR-composition technique of [CPS$^+$15] to construct a PoWorK. We note that our dense puzzle based PoWorK construction also supports puzzles based on hash functions and the RO model for which no efficient Σ protocols exist.

Roadmap. The rest of this paper is organized as follows. In Section 2, we provide basic notation, and formalize cryptographic puzzles, the additional properties of dense samplable puzzles and the property of amortization resistance, as well as the notion of PoWorKs by defining completeness, f-soundness and indistinguishability. In Section 3, we present our efficient dense puzzle based construction built upon an arbitrary 3-move special sound HVZK protocol for a language $\mathcal{L}$ and some puzzle system, and prove that our construction achieves f-soundness and indistinguishability. In the same section, we present two dense puzzle instantiations. In Section 4, we describe two applications of PoWorKs. Namely, (i) a method to reduce the amount of spam email while preserving the privacy of the receiver and (ii) a more theoretical application by showing that any 3-move PoWorK which satisfies a couple of plausible assumptions is a 3-move straight-line concurrent statistically $\lambda^{\mathrm{poly}(\log \lambda)}$-simulatable argument of knowledge as defined in [Pas03, Pas04]. Finally, in Appendix C we provide a second PoWorK construction based on the Lapidot-Shamir 3-move special sound computationally special HVZK protocol [LS90], which is less efficient than the dense puzzle based construction but works for all puzzle systems; note that this construction is not black-box with respect to the puzzle and depending on the puzzle may not be public-coin.

2 Definitions

We start by setting the notation to be used in the rest of the paper. By λ we denote the security parameter and by $\mathsf{negl}(\cdot)$ the property that a function is negligible in some parameter. Let $z \stackrel{\$}{\leftarrow} \mathcal{Z}$ denote the uniformly at random selection of z from space $\mathcal{Z}$ and $\Delta[\mathbf{X}, \mathbf{Y}]$ the statistical distance of random variables (or distributions) $\mathbf{X}, \mathbf{Y}$. Composition of functions is defined by $\circ$.

Let $\langle \mathcal{P}(y) \leftrightarrow \mathcal{V} \rangle(x, z)$ denote the interaction between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ on common input x, auxiliary input z, and $\mathcal{P}$'s private input y. For an algorithm $\mathcal{B}$ that is part of an interactive protocol let $view_{\mathcal{B}}$ and $output_{\mathcal{B}}$

denote the views and the output of $\mathcal{B}$ respectively. Let $\mathsf{Steps}_{\mathcal{B}}(x)$ be the number of steps (i.e. machine/operation cycles) executed by algorithm $\mathcal{B}$ on input x, and $\mathsf{Steps}_{\mathcal{P}}(\langle \mathcal{P}(y) \leftrightarrow \mathcal{V}\rangle(x,z))$ be the number of steps of $\mathcal{P}$, when interacting on inputs x, y, z[1]. If $R_{\mathcal{L}}$ is a witness relation for the language $\mathcal{L} \in \mathcal{NP}$ (i.e. $R_{\mathcal{L}}$ polynomial-time-decidable and $(x,w) \in R_L$ implies that $|w| \leq \mathsf{poly}(|x|)$), we define the set of witnesses for the membership $x \in L$ as $R_L(x) = \{w : (x,w) \in R_L\}$.

2.1 Cryptographic Puzzles

Roughly speaking, a cryptographic puzzle should be easy to generate, hard to solve, and easy to verify. Given a specific security parameter λ, we denote the puzzle space as $\mathcal{PS}_\lambda$, the solution space as $\mathcal{SS}_\lambda$, and the hardness space as $\mathcal{HS}_\lambda$. We first define puzzles with a minimum set of properties, and then add extra properties that are useful in our constructions.

Definition 1 *A puzzle system* $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$ *consists of the following four algorithms:*

- $\mathsf{Sample}(1^\lambda, h)$ *is a probabilistic puzzle instance sampling algorithm. On input the security parameter* 1^λ *and a hardness factor* $h \in \mathcal{HS}_\lambda$*, it outputs a puzzle instance* $\mathsf{puz} \in \mathcal{PS}_\lambda$*.*
- $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$ *is a probabilistic puzzle solving algorithm. On input the security parameter* 1^λ*, a hardness factor* $h \in \mathcal{HS}_\lambda$ *and a puzzle instance* $\mathsf{puz} \in \mathcal{PS}_\lambda$*, it outputs a potential solution* $\mathsf{soln} \in \mathcal{SS}_\lambda$*.*
- $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln})$ *is a deterministic puzzle verification algorithm. On input the security parameter* 1^λ*, a hardness factor* $h \in \mathcal{HS}_\lambda$*, a puzzle instance* $\mathsf{puz} \in \mathcal{PS}_\lambda$ *and a potential solution* $\mathsf{soln} \in \mathcal{SS}_\lambda$ *it outputs* true *or* false.

Subsequently, we define the following properties for a puzzle system.

Completeness: We say that a puzzle system PuzSys is *complete*, if for every $h \in \mathcal{HS}_\lambda$:

$$\Pr\left[\begin{array}{l}\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); \mathsf{soln} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz}) : \\ \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{false}\end{array}\right] = \mathsf{negl}(\lambda).$$

g-Hardness: Let $g : \mathbb{N} \longrightarrow \mathbb{R}^+$ be a monotonically decreasing function. We say that a puzzle system PuzSys is *g-hard*, if for every adversary $\mathcal{A}$, for every auxiliary tape $z \in \{0,1\}^*$ and for every $h \in \mathcal{HS}_\lambda$:

$$\Pr\left[\begin{array}{l}\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); \mathsf{soln} \leftarrow \mathcal{A}(z, 1^\lambda, h, \mathsf{puz}) : \\ \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{true} \wedge \\ \wedge \mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \mathsf{puz}) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))\end{array}\right] = \mathsf{negl}(\lambda).$$

Dense Samplable Puzzles. In addition to the standard puzzle definition, for our PoWorK construction in Section 3 we need puzzles that can be sampled by just generating random strings (i.e. the puzzle instances should be "dense" over $\{0,1\}^{\ell(\lambda,h)}$ for some function ℓ and $\lambda, h \in \mathbb{Z}^+$). Formally it holds that for some function ℓ in λ and h,

$$\Delta[\mathsf{Sample}(1^\lambda, h), \mathbf{U}_{\ell(\lambda,h)}] = \mathsf{negl}(\lambda),$$

where $\mathbf{U}_{\ell(\lambda,h)}$ stands for the uniform distribution over $\{0,1\}^{\ell(\lambda,h)}$. For such puzzles we will require some additional properties. First there should be a puzzle sampler that outputs a valid solution together with puz:

- $\mathsf{SampleSol}(1^\lambda, h)$ is a probabilistic solved puzzle instance sampling algorithm. On input the security parameter 1^λ and a hardness factor $h \in \mathcal{HS}_\lambda$, it outputs a puzzle instance and solution pair $(\mathsf{puz}, \mathsf{soln}) \in \mathcal{PS}_\lambda \times \mathcal{SS}_\lambda$.

[1] In this work we focus on parallelizable puzzles so counting in number steps as opposed to actuall running time is more intuitive.

Correctness: We say that a puzzle system PuzSys is *correct*, if for every $h \in \mathcal{HS}_\lambda$, we have that:

$$\Pr\left[\ (\mathsf{puz}, \mathsf{soln}) \leftarrow \mathsf{SampleSol}(1^\lambda, h) : \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{false}\ \right] = \mathsf{negl}(\lambda).$$

Efficient Samplability: We say $\mathsf{SampleSol}$ is *efficient* with respect to the puzzle g-hardness, if for every $\lambda \in \mathbb{Z}^+$, $h \in \mathcal{HS}_\lambda$ and $\mathsf{puz} \in \mathcal{PS}_\lambda$, we have that:

$$\mathsf{Steps}_{\mathsf{SampleSol}}(1^\lambda, h)) < g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})).$$

Statistical Indistinguishability: We define the following two probability distributions

$$\mathbf{D}_{s,\lambda,h} \stackrel{def}{=} \left\{(\mathsf{puz}, \mathsf{soln}) \leftarrow \mathsf{SampleSol}(1^\lambda, h)\right\} \quad \text{and}$$

$$\mathbf{D}_{p,\lambda,h} \stackrel{def}{=} \left\{\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h), \mathsf{soln} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz}) : (\mathsf{puz}, \mathsf{soln})\right\}.$$

We say a PuzSys is *statistically indistinguishable*, if for every $\lambda \in \mathbb{Z}^+$ and $h \in \mathcal{HS}_\lambda$:

$$\Delta[\mathbf{D}_{s,\lambda,h}, \mathbf{D}_{p,\lambda,h}] = \mathsf{negl}(\lambda).$$

(τ, k)**-Amortization Resistance.** For certain applications it is important that the puzzle is not amenable to amortization. We say that a g-hard puzzle system, PuzSys, is (τ, k)*-amortization resistant* if for every adversary $\mathcal{A}$, for every auxiliary tape $z \in \{0,1\}^*$ and for every $h \in \mathcal{HS}_\lambda$:

$$\Pr\left[\begin{array}{l} \forall 1 \le i \le k : \mathsf{puz}_i \leftarrow \mathsf{Sample}(1^\lambda, h); \\ \{\mathsf{soln}_1, \ldots, \mathsf{soln}_k\} \leftarrow \mathcal{A}(z, 1^\lambda, h, \{\mathsf{puz}_1, \ldots, \mathsf{puz}_k\}) : \\ \big(\forall 1 \le i \le k : \quad \mathsf{Verify}(1^\lambda, h, \mathsf{puz}_i, \mathsf{soln}_i) = \mathsf{true} \quad \big)\wedge \\ \wedge\Big(\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \{\mathsf{puz}_1\}_{i=1}^k) \le \tau\big(\sum_{i=1}^k g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)\Big) \end{array}\right] = \mathsf{negl}(\lambda).$$

Informally, (τ, k)-amortization resistance implies a lower bound on the hardness preservation against adversaries that attempt to benefit from solving vectors of puzzles of length k.

2.2 Definition of PoWorK

In a PoWorK, the prover $\mathcal{P}$ may interact with the verifier $\mathcal{V}$ by running in either of the two following modes: (a) the *Proof of Knowledge (PoK)* mode, where $\mathcal{P}$ convinces $\mathcal{V}$ that she knows a witness for some statement x, or (b) the *Proof of WorK (PoW)* mode, where $\mathcal{P}$ makes calls to the puzzle solving algorithm to solve a certain puzzle. For some language in $\mathcal{NP}$ and a fixed puzzle system PuzSys, we define PoWorK to satisfy: (i) completeness, (ii) f-soundness (for some "computation-scaling" function f) and (iii) indistinguishability, as follows:

Definition 2 (PoWorK) *Let $\mathcal{L}$ be a language in $\mathcal{NP}$ and $R_{\mathcal{L}}$ be a witness relation for $\mathcal{L}$. Let* $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$ *be a puzzle system anf f be a function. We say that $(\mathcal{P}, \mathcal{V})$ is an f-*sound Proof of Work or Knowledge (PoWorK) *for $\mathcal{L}$ and* PuzSys*, if the following properties are satisfied:*

(i). ***Completeness:*** *for every $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$, $w \in R_{\mathcal{L}}(x)$, $z \in \{0,1\}^*$ and every hardness factor $h \in \mathcal{HS}_\lambda$, it holds that*

(i.a) $\Pr[output_{\mathcal{V}} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V}\rangle(x, z, h) : output_{\mathcal{V}} = \text{accept}] = 1 - \mathsf{negl}(\lambda)$ *and*

(i.b) $\Pr[output_{\mathcal{V}} \leftarrow \langle \mathcal{P}^{\mathsf{Solve}(1^\lambda, h, \cdot)} \leftrightarrow \mathcal{V}\rangle(x, z, h) : output_{\mathcal{V}} = \text{accept}] = 1 - \mathsf{negl}(\lambda)$.

*(ii). f-**Soundness:** For every $x \in \{0,1\}^{\mathrm{poly}(\lambda)}, y, z \in \{0,1\}^*$, every hardness factor $h \in \mathcal{HS}_\lambda$ and prover $\mathcal{P}^*$ define by $\pi_{x,y,z,h}$ the probability*

$$\Pr\left[\begin{array}{l}\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); output_\mathcal{V} \leftarrow \langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h) : (output_\mathcal{V} = \mathrm{accept}) \\ \wedge \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h)) \leq f(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))\end{array}\right].$$

f-Soundness holds if there are non-negligible functions s, q such that for any $\mathcal{P}^*$ there exists a PPT witness-extraction algorithm $\mathcal{K}$, such that if $\pi_{x,y,z,h} \geq s(\lambda)$ (representing the knowledge error) then the probability

$$\Pr[\mathcal{K}^{\mathcal{P}^*}(x,y,z,h) \in R_\mathcal{L}(x)] \geq q(\lambda) .$$

*(iii). **Statistical (resp. Computational) Indistinguishability:** for every $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$, $w \in R_\mathcal{L}(x)$, $z \in \{0,1\}^*$, for every hardness factor $h \in \mathcal{HS}_\lambda$ and for every verifier (resp. PPT verifier) $\mathcal{V}^*$, the following two random variables are statistically (resp. computationally) indistinguishable:*

$$\begin{array}{lcl}\mathbf{D}^{\mathcal{V}^*}_{PoK} & \stackrel{def}{=} & \{view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\} \\ \mathbf{D}^{\mathcal{V}^*}_{PoW} & \stackrel{def}{=} & \left\{view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}^{\mathsf{Solve}(1^\lambda,h,\cdot)} \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\right\}.\end{array}$$

Intuitively, soundness is related to the hardness of solving a presumably hard cryptographic puzzle. The hardness threshold T is set to be the (probabilistic) computational complexity (in number of steos) of the puzzle solver, when the latter is provided some output of the puzzle sampling algorithm, scaled to some function f. According to Definition 2, any prover who does not know a witness, cannot convince the verifier in steps less than $f(T)$ with some good probability. Observe that in the definition of f-soundness, the convincing capability of the prover is limited by the hardness of solving puzzle challenges. This implies that in an f-sound protocol, provers who do not know (per the knowledge extractor) are forced to "work" in order to convince the verifier. The indistinguishability property of PoWorKs implies that a (potentially malicious) verifier cannot distinguish the running mode (PoK or PoW) that $\mathcal{P}$ follows.

3 The Dense Puzzle Based PoWorK Construction

In this section, we show how to transform an arbitrary 3-move, public coin, special sound, honest verifier zero-knowledge (SS-HVZK) (cf. App. A.1) into a 3-move public-coin PoWorK. Our construction is lightweight and requires dense samplable puzzle systems that we formalized in Section 1. Additionally, we provide a second construction (cf. App. C) which is less efficient, non-black-box on the puzzle, but it works for all puzzle systems and may not be public-coin (depending on the puzzle).

3.1 Preliminaries

For both constructions, we consider a puzzle system PuzSys that achieves completeness and g-hardness for some function $g : \mathbb{N} \longrightarrow \mathbb{R}^+$. In addition, for dense samplable puzzle systems, we also require correctness, efficient samplability, and statistical indistinguishability. The puzzle, solution and hardness spaces are denoted by $\mathcal{PS}_\lambda, \mathcal{SS}_\lambda, \mathcal{HS}_\lambda$, as in Section 2.1. Our PoWorK protocols are interactive proofs between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, denoted by $(\mathcal{P}, \mathcal{V})$.

The challenge space of our dense puzzle based construction $(\mathcal{P}, \mathcal{V})$, denoted by $\mathcal{CS}_\lambda$, is determined by the security parameter λ. From an algebraic point of view, $\mathcal{CS}_\lambda$ is set to be a group with operation $\oplus$, where performing $\oplus$ and inverting an element should be efficient. For the first construction, we require that $\mathcal{PS}_\lambda \subseteq \mathcal{CS}_\lambda$. For instance, we may set $\mathcal{CS}_\lambda$ as the group $\left(\mathbb{GF}(2^{\ell(\lambda)}, \oplus\right)$, where $\ell(\lambda)$ is the length of the challenges and $\oplus$ is the bitwise XOR operation. Of course, one may select a different setting which could be tailor made to the algebraic properties of the underlying primitives.

Let ChSampler be the algorithm that samples a challenge from $\mathcal{CS}_\lambda$. For a fixed security parameter, we define the following random variables (r.v.):

- The challenge sampling r.v. $\mathbf{C}_{\lambda,h} \stackrel{def}{=} \mathsf{ChSampler}(1^\lambda, h)$.
- The puzzle sampling r.v. $\mathbf{P}_{\lambda,h} \stackrel{def}{=} \{\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h) : \mathsf{puz}\}$.

Finally, we denote by $x \oplus \mathbf{D}$ (resp. $\mathbf{D}^{\mathsf{Inv}}$) the r.v. of performing $\oplus$ on some fixed $x \in \mathcal{CS}_\lambda$ and an element y sampled from r.v. $\mathbf{D}$ (resp. inverting an element sampled from $\mathbf{D}$). The r.v. $\mathbf{D} \oplus x$ is defined similarly. Formally,

$$x \oplus \mathbf{D} \stackrel{def}{=} \{y \leftarrow \mathbf{D} : x \oplus y\},\ \mathbf{D} \oplus x \stackrel{def}{=} \{y \leftarrow \mathbf{D} : y \oplus x\},\ \mathbf{D}^{\mathsf{Inv}} \stackrel{def}{=} \{y \leftarrow \mathbf{D} : y^{-1}\}.$$

3.2 The Dense Puzzle Based Compiler

We now provide a detailed description of our protocol $(\mathcal{P}, \mathcal{V})$, which can be viewed as a compiler that can transform a SS-HVZK protocol $\Pi = (\mathsf{P1}_\Pi, \mathsf{P2}_\Pi, \mathsf{Ver}_\Pi)$ for $\mathcal{L} \in \mathcal{NP}$ (cf., App. A.1 for details) and a g-hard puzzle system PuzSys into a 3-move PoWorK. The resulting PoWorK protocol achieves $\Theta(g)$-hardness and statistical indistiguishability. From a syntax point of view, our compiler will set the challenge space of the PoWorK $\mathcal{CS}_\lambda$ to be equal to $\mathcal{CS}_\Pi$. We denote by Sim_Π the HVZK simulator of Π.

The protocol $(\mathcal{P}, \mathcal{V})$ can be executed in either of the two following modes:

1. **Proof of Knowledge (PoK) mode:** $\mathcal{P}$ has a witness $w \in \mathcal{R}_\mathcal{L}(x)$ as private input. In order to prove knowledge of w to $\mathcal{V}$, $\mathcal{P}$ runs $\mathsf{P1}_\Pi$ and $\mathsf{P2}_\Pi$ as described by the original SS-HVZK protocol, with the difference that instead of providing $\mathsf{P2}_\Pi$ with the challenge c from $\mathcal{V}$ directly, $\mathcal{P}$ runs the puzzle sampler algorithm to receive a pair of a puzzle and its solution, $(\mathsf{puz}, \mathsf{soln})$, computes the value $\tilde{c} = c \oplus \mathsf{puz}$ and runs $\mathsf{P2}_\Pi$ with challenge $\tilde{c}$.
2. **Proof of Work (PoW) mode:** $\mathcal{P}$ has no private input and tries to convince $\mathcal{V}$ that it has performed a minimum amount of computational "work" (i.e. at least some expected number of steps). To achieve this, $\mathcal{P}$ runs Sim_Π to simulate a transcript of the original SS-HVZK protocol. Then, it receives the challenge c from $\mathcal{V}$ and computes the value $\mathsf{puz} = c^{-1} \oplus \tilde{c}$. It runs the Solve algorithm on input puz, and if puz is a puzzle in $\mathcal{PS}_\lambda$ (which, as we argue later, must occur with high probability), then it obtains a solution soln of puz, except for some negligible error.

The verification mechanism, must be the same for both modes, so that indistinguishability can be achieved. Namely, the verifier checks that: (i) the relation $\tilde{c} = c \oplus \mathsf{puz}$ holds, (ii) the transcript of the SS-HVZK protocol is accepting and (iii) the prover has output a correct pair of a puzzle puz and some solution soln of puz. The protocol $(\mathcal{P}, \mathcal{V})$ is presented in detail in Figure 1.

3.3 Security of the Dense Puzzle Based Construction.

In order to prove that our protocol satisfies soundness and indistinguishability, we need to assume that the challenge and puzzle distributions satisfy some plausible properties and that the presumed g-hardness of the puzzle system dominates the step complexity of the group operation and challenge sampling algorithms. In detail, we require that:

(A). The challenge and puzzle sampling distributions are statistically close.

(B). The challenge sampling distribution is (statistically) *invariant* to any group operation, i.e. (a) inverting a challenge sampled from $\mathcal{CS}_\lambda$ and (b) performing $\oplus$ operations on some element x in $\mathcal{CS}_\lambda = \mathcal{CS}_\Pi$ and a sampled challenge. Observe that these two assumptions imply that the puzzle sampling distribution is also (statistically) $\oplus$-invariant.

(a) Knowing the witness (PoK)	(b) Doing work (PoW)
Statement: $x \in \mathcal{L} \cap \{0,1\}^{\text{poly}(\lambda)}$.	**Statement:** $x \in \mathcal{L} \cap \{0,1\}^{\text{poly}(\lambda)}$.
Prover's private input: $w \in R_{\mathcal{L}}(x)$.	**Prover's private input:** $-$
$\mathcal{P}$: $(\tilde{a}, \phi_1) \leftarrow \mathsf{P1}_{\Pi}(w, x)$.	$\mathcal{P}$: • execute $(\tilde{a}, \tilde{c}, \tilde{r}) \leftarrow \mathsf{Sim}_{\Pi}(x)$;
$\mathcal{P} \rightarrow \mathcal{V}$: $\tilde{a}$.	$\mathcal{P} \rightarrow \mathcal{V}$: $\tilde{a}$.
$\mathcal{P} \leftarrow \mathcal{V}$: $c \leftarrow \mathsf{ChSampler}(1^\lambda, h)$;	$\mathcal{P} \leftarrow \mathcal{V}$: $c \leftarrow \mathsf{ChSampler}(1^\lambda, h)$;
$\mathcal{P}$: • sample a puzzle-solution pair $(\mathsf{puz}, \mathsf{soln}) \leftarrow \mathsf{SampleSol}(1^\lambda, h)$; • set $\tilde{c} = c \oplus \mathsf{puz}$; • execute $\tilde{r} \leftarrow \mathsf{P2}_{\Pi}(\phi_1, \tilde{c})$;	$\mathcal{P}$: • set $\mathsf{puz} = c^{-1} \oplus \tilde{c}$; • compute a puzzle solution $\mathsf{soln} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz})$;
$\mathcal{P} \rightarrow \mathcal{V}$: $\tilde{c}, \tilde{r}, \mathsf{puz}, \mathsf{soln}$.	$\mathcal{P} \rightarrow \mathcal{V}$: $\tilde{c}, \tilde{r}, \mathsf{puz}, \mathsf{soln}$.
Verification:	**Verification:**
1. $\tilde{c} = c \oplus \mathsf{puz}$.	1. $\tilde{c} = c \oplus \mathsf{puz}$.
2. $\mathsf{Ver}_{\Pi}(x, \tilde{a}, \tilde{c}, \tilde{r}) = 1$.	2. $\mathsf{Ver}_{\Pi}(x, \tilde{a}, \tilde{c}, \tilde{r}) = 1$.
3. $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{true}$.	3. $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{true}$.

(a) Knowing the witness (PoK) (b) Doing work (PoW)

Figure 1: The Dense Puzzle Based PoWorK Construction for fixed security parameter λ and pre-determined hardness factor $h \in \mathcal{HS}_\lambda$, given a 3-move-SS-HVZK protocol Π for language $\mathcal{L}$ and a dense samplable puzzle system PuzSys satisfying that $\mathcal{PS}_\lambda \subseteq \mathcal{CS}_\lambda = \mathcal{CS}_\Pi$; $\mathsf{ChSampler}$ is the challenge sampling algorithm over $\mathcal{CS}_\lambda$.

(C). With high probability, the number of steps needed for $\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})$ to solve a g-hard puzzle puz according to $\mathbf{P}_{\lambda,h}$, scaled to the puzzle hardness function g, is more than the number of steps of performing group operations (inversion and $\oplus$ operation), or sampling from $\mathcal{CS}_\lambda$.

The assumptions described are stated formally in Figure 2. Assumptions (**A**) and (**B**) can be met for meaningful distributions, widely used in cryptographic protocols. For example, when $\mathbf{C}_{\lambda,h}$ and $\mathbf{P}_{\lambda,h}$ are close to uniform, it is straightforward that assumption (**A**) holds. Moreover, since the uniform distribution is invariant under group operations, we have that assumption (**B**) also holds. The assumption (**C**) is expected to hold for any meaningful cryptographic puzzle construction. Indeed, if we believe that solving a puzzle is hard (on average) within a bounded amount of steps T, then performing efficient tasks, such as group operations or sampling a challenge in the space where this puzzle belongs must be feasible in a number of steps much less than T.

We prove that our dense puzzle based construction is a PoWorK, assuming (**A**),(**B**),(**C**), the g-hardness of PuzSys and the soundness and ZK properties of the original SS-HVZK protocol. The soundness of our protocol is in constant relation with the hardness of PuzSys.

Theorem 1 *Let $\mathcal{L}$ be a language in $\mathcal{NP}$ and let $\Pi = (\mathsf{P1}_\Pi, \mathsf{P2}_\Pi, \mathsf{Ver}_\Pi)$ be a special-sound 3-move statistical HVZK protocol for $\mathcal{L}$, where the challenge sampling distribution is uniform. Let* $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{SampleSol}, \mathsf{Solve}, \mathsf{Verify})$ *be a dense samplable puzzle system that satisfies g-hardness for some function g. Define $(\mathcal{P}, \mathcal{V})$ as the protocol described in Figure 1 when built upon Π,* PuzSys *and assume that (**A**),(**B**),(**C**) in Figure 2 hold. Then, $(\mathcal{P}, \mathcal{V})$ is a $\big((1-\kappa)/2\big) \cdot g$-sound PoWorK for $\mathcal{L}$ and* PuzSys *with statistical indistiguishability, where κ is the constant defined in assumption (**C**).*

Proof:

Completeness. By the completeness of Π and the correctness of PuzSys, the dense puzzle based PoWorK construction is complete in the case that $\mathcal{P}$ executes the PoK mode of the protocol. Regarding the PoW mode, an honest execution of PuzSys is incorrect, only if either of the two following cases is true:

(**A**). For every hardness factor $h \in \mathcal{HS}_\lambda$, the r.v. $\mathbf{C}_{\lambda,h}$ and $\mathbf{P}_{\lambda,h}$ are ϵ_1-statistically close, where $\epsilon_1(\cdot)$ is a negligible function.

(**B**). For every $x \in \mathcal{CS}_\lambda$ and hardness factor $h \in \mathcal{HS}_\lambda$, the r.v. $\mathbf{C}_{\lambda,h}$ is ϵ_2-statistically close to the r.v. $x \oplus \mathbf{C}_{\lambda,h}$, $\mathbf{C}_{\lambda,h} \oplus x$ and $\mathbf{C}^{\mathsf{Inv}}_{\lambda,h}$, where $\epsilon_2(\cdot)$ is a negligible function.

(**C**). There exists a constant $\kappa < 1$ and a negligible function $\epsilon_3(\cdot)$ s.t. for every hardness factor $h \in \mathcal{HS}_\lambda$ and every $r, r' \in \mathcal{CS}_\lambda$

$$\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h) : \kappa \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) > \\ > \mathsf{Steps}_{\mathsf{ChSampler}}(1^\lambda, h) + \mathsf{Steps}_{\mathsf{Inv}}(r) + \mathsf{Steps}_{\oplus}(r, r')] \geq 1 - \epsilon_3(\lambda),$$

where $\mathsf{Steps}_{\mathsf{Inv}}, \mathsf{Steps}_{\oplus}$ denote the number of steps needed for inversion and group operation in $\mathcal{CS}_\lambda$.

Figure 2: Assumptions for our Dense Puzzle Based PoWorK Construction, where $\mathbf{C}_{\lambda,h}$ and $\mathbf{P}_{\lambda,h}$ are the challenge sampling and the puzzle sampling distributions respectively.

(i). $\mathsf{puz} = c^{-1} \oplus \tilde{c} \in \mathcal{CS}_\lambda \setminus \mathcal{PS}_\lambda$, i.e. puz is not a puzzle. By assumptions (**A**), (**B**) in Figure 2, this happens with negligible probability, since

$$\Delta[\mathbf{P}_{\lambda,h}, \mathbf{C}_{\lambda,h}] \leq \epsilon_1(\lambda) \wedge \Delta[\mathbf{C}_{\lambda,h}, \mathbf{C}^{\mathsf{Inv}}_{\lambda,h} \oplus \tilde{c}] \leq 2 \cdot \epsilon_2(\lambda) \Rightarrow \Delta[\mathbf{P}_{\lambda,h}, \mathbf{C}^{\mathsf{Inv}}_{\lambda,h} \oplus \tilde{c}] \leq \epsilon_1(\lambda) + 2 \cdot \epsilon_2(\lambda),$$

where we applied (**B**) two times (one for inversion and one for $\oplus$ operation).

(ii). puz is a puzzle, but the puzzle solver algorithm Solve does not output a solution for puz. Namely, we have that $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{false}$. By the completeness property of PuzSys, this also happens with negligible probability.

Therefore, $(\mathcal{P}, \mathcal{V})$ achieves completeness with high probability, as required in Definition 2.

$\big((1-\kappa)/2\big) \cdot g$-**Soundness.** First, we make use of the special soundness PPT extractor K_Π of Π to construct a knowledge extractor $\mathcal{K}$ that on input (x, y, z) and given oracle access to an arbitrary prover $\hat{\mathcal{P}}$, executes the following steps:

1. By applying standard rewinding, $\mathcal{K}$ interacts with $\hat{\mathcal{P}}(y)$ for statement x and auxiliary input z, using two challenges c_1, c_2 sampled from $\mathbf{C}_{\lambda,h}$ and receives two protocol transcripts $\langle \tilde{a}_1, c_1, (\tilde{c}_1, \tilde{r}_1, \mathsf{puz}_1, \mathsf{soln}_1)\rangle$ and $\langle \tilde{a}_1, c_2, (\tilde{c}_2, \tilde{r}_2, \mathsf{puz}_2, \mathsf{soln}_2)\rangle$.
2. $\mathcal{K}$ runs $\mathcal{K}_\Pi$ on input $(x, \langle \tilde{a}_1, \tilde{c}_1, \tilde{r}_1\rangle, \langle \tilde{a}_1, \tilde{c}_2, \tilde{r}_2\rangle)$.
3. $\mathcal{K}$ returns the output of $\mathcal{K}_\Pi$.

Since K_Π is a PPT algorithm, $\mathcal{K}$ also runs in polynomial time.

Assume that for some $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$, $y \in \{0,1\}^*, z \in \{0,1\}^*$, $h \in \mathcal{HS}_\lambda$, there exists a prover $\mathcal{P}^*$ and a non-negligible function $s(\cdot)$ s.t

$$\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); output_{\mathcal{V}} \leftarrow \langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z, h) : (output_{\mathcal{V}} = \text{accept}) \\ \wedge \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z, h)) \leq \big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))] \geq s(\lambda).$$

We will construct an algorithm $\mathcal{W}$ that will make use of $\mathcal{P}^*$ to break the g-hardness of PuzSys. The input that $\mathcal{W}$ receives is $\langle (x, y, z), 1^\lambda, h, \mathsf{puz}\rangle$, where (x, y, z) is the auxiliary input and puz sampled from $\mathsf{Sample}(1^\lambda, h)$. Then, $\mathcal{W}$ executes the following steps:

1. It samples c_1 by running $\mathsf{ChSampler}(1^\lambda, h)$.

2. It interacts with $\mathcal{P}^*(y)$ for statement x, auxiliary input z, hardness factor h and challenge c_1. It receives the transcript $\langle \tilde{a}_1, c_1, (\tilde{c}_1, \tilde{r}_1, \mathsf{puz}_1, \mathsf{soln}_1)\rangle$.

3. It computes the inverse of puz, denoted by puz^{-1}.

4. It computes $c_2 = \tilde{c}_1 \oplus \mathsf{puz}^{-1}$.

5. It rewinds $\mathcal{P}^*$ at the challenge phase and provides $\mathcal{P}^*$ with challenge c_2. It receives a second transcript $\langle \tilde{a}_1, c_2, (\tilde{c}_2, \tilde{r}_2, \mathsf{puz}_2, \mathsf{soln}_2)\rangle$.

6. It returns the value soln_2.

By the assumption for $\mathcal{P}^*$ and the splitting Lemma, we have that when $\mathcal{P}^*$ is challenged with two honestly selected c_1, c_2, it outputs two accepting transcripts by running in no more than $\big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))$ steps with at least $(s(\lambda)/2)^2$ probability. We denote by Equal, the event that this happens and $\tilde{c}_1 = \tilde{c}_2$ holds. Obviously, either Equal, or $\neg$Equal will occur with at least $(s(\lambda)/2)^2/2 = s(\lambda)^2/8$ probability.

Assume that Equal happens with at least $s(\lambda)^2/8$ probability. We will show that this case leads to a contradiction; namely, $\mathcal{W}$ will output a solution of puz while running in no more than $g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))$ steps, hence breaking the g-hardness of PuzSys.

We observe that for any puz, if both transcripts generated by the interaction with $\mathcal{P}^*$ are accepting and the values $\tilde{c}_1, \tilde{c}_2$ are equal, then we have that

$$(c_2 = \tilde{c}_1 \oplus \mathsf{puz}^{-1}) \wedge (\tilde{c}_2 = c_2 \oplus \mathsf{puz}_2) \wedge (\tilde{c}_1 = \tilde{c}_2) \Rightarrow \mathsf{puz}_2 = (\mathsf{puz}^{-1})^{-1} = \mathsf{puz},$$

where the second equality holds due to verification step 1. Therefore, it holds that

$$\mathsf{Verify}(1^\lambda, h, \mathsf{puz}_2, \mathsf{soln}_2) = \mathsf{true} \Leftrightarrow \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}_2) = \mathsf{true}. \tag{1}$$

By the assumptions (**A**),(**B**) in Figure 2, we have that there are negligible functions $\epsilon_1(\lambda), \epsilon_2(\lambda)$ s.t. for any $\tilde{c}_1$ that $\mathcal{P}^*$ returns,

$$\Delta[\tilde{c}_1 \oplus \mathbf{C}^{\mathsf{Inv}}_{\lambda,h}, \tilde{c}_1 \oplus \mathbf{P}^{\mathsf{Inv}}_{\lambda,h}] < 2\epsilon_1(\lambda) \quad \text{and} \quad \Delta[\mathbf{C}_{\lambda,h}, \tilde{c}_1 \oplus \mathbf{C}^{\mathsf{Inv}}_{\lambda,h}] < 2\epsilon_2(\lambda),$$

where in the first and second inequality, we applied assumptions (**A**) and (**B**) respectively two times (one for inversion and one for $\oplus$ operation). Therefore, by the triangular inequality we have that

$$\Delta[\mathbf{C}_{\lambda,h}, \tilde{c}_1 \oplus \mathbf{P}^{\mathsf{Inv}}_{\lambda,h}] < 2\epsilon_1(\lambda) + 2\epsilon_2(\lambda). \tag{2}$$

Eq. (2) implies that the probability distribution of $c_2 = \tilde{c}_1 \oplus \mathsf{puz}^{-1}$ that $\mathcal{W}$ computes is $[2\epsilon_1(\cdot) + 2\epsilon_2(\cdot)]$-statistically close to the challenge sampling distribution of $\mathcal{V}$.

By construction, the running time of $\mathcal{W}$ (in number of steps) is at most

$$2 \cdot \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h)) + \mathsf{Steps}(\mathsf{puz}^{-1}) + \mathsf{Steps}(\tilde{c}_1 \oplus \mathsf{puz}^{-1}) + \mathsf{Steps}_{\mathsf{ChSampler}}(1^\lambda, h).$$

By assumption (**C**) in Figure 2, there is a negligible function $\epsilon_3(\cdot)$ and a constant $\kappa < 1$ s.t.

$$\begin{aligned}\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h) : \kappa \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) < \mathsf{Steps}_{\mathsf{ChSampler}}(1^\lambda, h) + \\ + \mathsf{Steps}(\mathsf{puz}^{-1}) + \mathsf{Steps}(\tilde{c}_1 \oplus \mathsf{puz}^{-1})] \leq \epsilon_3(\lambda).\end{aligned} \tag{3}$$

When Equal occurs, then it holds that

$$\mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h)) \leq \big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})),$$

hence by the assumption for $\mathcal{P}^*$ and Eq. (2), (3), the probability that the running time of $\mathcal{W}$ is bounded by

$$\begin{aligned}\mathsf{Steps}_{\mathcal{W}}(1^\lambda,(x,y,z),h,\mathsf{puz}) &\le 2\cdot \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y)\leftrightarrow \mathcal{V}\rangle(x,z,h)) + \kappa\cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda,h,\mathsf{puz})) \le \\ &\le \big(2\cdot\big((1-\kappa)/2\big)\big)\cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda,h,\mathsf{puz})) + \kappa\cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda,h,\mathsf{puz})) = \\ &= g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda,h,\mathsf{puz})),\end{aligned}$$

is at least $\Pr[\text{Equal}] - \big(2\epsilon_1(\lambda)+2\epsilon_2(\lambda)+\epsilon_3(\lambda)\big)$. By Eq. (1) ,(2), (3),and the assumption $\Pr[\text{Equal}] \ge s(\lambda)^2/8$, we have that for auxiliary tape (x,y,z) and harndess factor h:

$$\Pr\left[\begin{array}{l}\mathsf{puz}\leftarrow \mathsf{Sample}(1^\lambda,h);\\ \mathsf{soln}_*\leftarrow \mathcal{W}(1^\lambda,(x,y,z),h,\mathsf{puz}):\\ \mathsf{Verify}(1^\lambda,h,\mathsf{puz},\mathsf{soln}_*)=\mathsf{true}\quad\wedge\\ \wedge\mathsf{Steps}_{\mathcal{W}}(1^\lambda,(x,y,z),h,\mathsf{puz})\\ \le g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda,h,\mathsf{puz}))\end{array}\right] \ge s(\lambda)^2/8 - \big(2\epsilon_1(\lambda)+2\epsilon_2(\lambda)+\epsilon_3(\lambda)\big),$$

which contradicts to the g-hardness of PuzSys, since $s(\lambda)^2/8 - \big(2\epsilon_1(\lambda)+2\epsilon_2(\lambda)+\epsilon_3(\lambda)\big)$ is a non-begligible function. Therefore, it holds that $\Pr[\text{Equal}] \le s(\lambda)^2/8$ which implies that

$$\Pr[\neg\text{Equal}] \ge s(\lambda)^2/8. \tag{4}$$

By the construction of $\mathcal{K}$ and the special soundness property of Π, we have that $\mathcal{K}$ will return a witness for x whenever $\mathcal{K}_\Pi$ is provided with different $\tilde{c}_1,\tilde{c}_2$. Define $q(\lambda)=s(\lambda)^2/8$. By Eq. (4), we have that when $\mathcal{K}$ is given oracle access to $\mathcal{P}^*$, it holds that

$$\Pr[\mathcal{K}^{\mathcal{P}^*}(x,y,z,h)\in R_{\mathcal{L}}(x)] = \Pr[\neg\text{Equal}] \ge q(\lambda).$$

Thus, we conclude that our protocol is $\big((1-\kappa)/2\big)\cdot g$-sound.

Statistical Indistinguishability. Assume that the protocol described in Figure 1 does not satisfy the PoWorK indistinguishability property in Definition 2. Then, for some (x,z,h) there exists a verifier $\mathcal{V}^*$ that w.l.o.g. outputs a single bit and can distinguish between:

$$\begin{aligned}\mathbf{D}^{\mathcal{V}^*}_{PoK} &= \{view_{\mathcal{V}^*}\leftarrow\langle\mathcal{P}(w)\leftrightarrow\mathcal{V}^*\rangle(x,z,h)\}\quad\text{and}\\ \mathbf{D}^{\mathcal{V}^*}_{PoW} &= \left\{view_{\mathcal{V}^*}\leftarrow\langle\mathcal{P}^{\mathsf{Solve}(1^\lambda,h,\cdot)}\leftrightarrow\mathcal{V}^*\rangle(x,z,h)\right\}.\end{aligned}$$

with non-negligible advantage $\eta(\lambda)$.

In the following, we will show that if such a $\mathcal{V}^*$ exists, then we can construct an adversary $\mathcal{B}$ who breaks the statistical (auxiliary input) HVZK property of the underlying 3-move protocol $\Pi = (\mathsf{P1}_\Pi,\mathsf{P2}_\Pi,\mathsf{Ver}_\Pi)$. This means (see Appendix A.1) that $\mathcal{B}$ can distinguish between:

$$\mathbf{D}_\Pi = \left\{(\tilde{a},\phi_1)\leftarrow\mathsf{P1}_\Pi(w,x);\tilde{c}\overset{\$}{\leftarrow}\mathcal{CS}_\Pi;\tilde{r}\leftarrow\mathsf{P2}_\Pi(\phi_1,\tilde{c}):(\tilde{a},\tilde{c},\tilde{r})\right\}\text{ and}$$
$$\mathbf{D}_{\mathsf{Sim}} = \{(\tilde{a},\tilde{c},\tilde{r})\leftarrow\mathsf{Sim}_\Pi(x,(z,h)):(\tilde{a},\tilde{c},\tilde{r})\}$$

with some non-negligible advantage $\eta'(\lambda)$, where (z,h) is the auxiliary input. Namely, $\mathcal{B}$ takes as input $(x,(z,h),(\tilde{a},\tilde{c},\tilde{r}))$, and works as follows:

1. Invokes $\mathcal{V}^*$ with input x,z,h and first move message $\tilde{a}$.
2. $\mathcal{V}^*$ responds back with his challenge c.
3. $\mathcal{B}$ computes $\mathsf{puz}=c^{-1}\oplus\tilde{c}$ and runs Solve on input $(1^\lambda,h,\mathsf{puz})$ to receive back soln.
4. $\mathcal{B}$ sends $(\tilde{c},\tilde{r},\mathsf{puz},\mathsf{soln})$ to $\mathcal{V}^*$.

5. $\mathcal{B}$ returns $\mathcal{V}^*$'s output b^*.

By construction of $\mathcal{B}$, what is left to argue is that $\mathsf{puz} = c^{-1} \oplus \tilde{c}$ and $\mathsf{soln} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz})$ are indistinguishable from a pair $(\mathsf{puz}', \mathsf{soln}')$ that was picked by $\mathsf{SampleSol}(1^\lambda, h)$. We stusy the following two cases:

1. *$\mathcal{B}$'s input is sampled according to $\mathbf{D}_\Pi$*: By the assumption (**B**) in Figure 2 and for any c returned by $\mathcal{V}^*$, we have that:
$$\Delta[\mathbf{C}_{\lambda,h}, \mathbf{C}^{\mathsf{Inv}}_{\lambda,h} \oplus \tilde{c}] < 2\epsilon_2(\lambda),$$
where we applied (**B**) two times (one for inversion and one for $\oplus$ operation). By assumption (**A**), we have that
$$\Delta[\mathbf{C}_{\lambda,h}, \mathbf{P}_{\lambda,h}] < \epsilon_1(\lambda).$$
By the triangular inequality, we have that for the distribution of $\mathsf{puz} = c^{-1} \oplus \tilde{c}$, it holds that
$$\Delta[\mathbf{P}_{\lambda,h}, \mathbf{C}^{\mathsf{Inv}}_{\lambda,h} \oplus \tilde{c}] < \epsilon_1(\lambda) + 2\epsilon_2(\lambda).$$
By the statistical indistinguishability property of PuzSys (Definition 1), we have that the distribution $\{\mathsf{soln} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz}) : \mathsf{soln}\}$ is $\epsilon_4(\lambda)$-statistically close to the distribution $\{(\mathsf{soln}', \mathsf{puz}') \leftarrow \mathsf{SampleSol}(1^\lambda, h) : \mathsf{soln}'\}$, for some negligible function ϵ_4. Consequently, the probability distribution of puz that $\mathcal{B}$ computes is $[\epsilon_1(\lambda) + 2\epsilon_2(\lambda) + \epsilon_4(\lambda)]$-statistically close to the puzzle sampling distribution.

2. *$\mathcal{B}$'s input is sampled according to $\mathbf{D}_{\mathsf{Sim}}$*: in this case, it is straightforward that $\mathcal{B}$ simulates perfectly the PoW mode of the PoWorK protocol.

By the above and given that the probability of success of $\mathcal{V}^*$ is at least $\eta(\lambda)$, we have that

$$\begin{aligned}
&\left|\Pr[(\tilde{a},\tilde{c},\tilde{r}) \leftarrow \mathbf{D}_\Pi : \mathcal{B}(x,(z,h),\tilde{a},\tilde{c},\tilde{r}) = 1] - \Pr[(\tilde{a},\tilde{c},\tilde{r}) \leftarrow \mathbf{D}_{\mathsf{Sim}} : \mathcal{B}(x,(z,h),\tilde{a},\tilde{c},\tilde{r}) = 1]\right| \geq \\
&\geq \Big|\big(\Pr[view_{\mathcal{V}^*} \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoK} : \mathcal{V}^*(view_{\mathcal{V}^*}) = 1] - (\epsilon_1(\lambda) + 2\epsilon_2(\lambda) + \epsilon_4(\lambda))\big) - \\
&\qquad - \Pr[view_{\mathcal{V}^*} \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoW} : \mathcal{V}^*(view_{\mathcal{V}^*}) = 1]\Big| \geq \\
&\geq \Big|\Pr[view_{\mathcal{V}^*} \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoK} : \mathcal{V}^*(view_{\mathcal{V}^*}) = 1] - \Pr[view_{\mathcal{V}^*} \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoW} : \mathcal{V}^*(view_{\mathcal{V}^*}) = 1]\Big| - \\
&\qquad - (\epsilon_1(\lambda) + 2\epsilon_2(\lambda) + \epsilon_4(\lambda))\big) \geq \\
&\geq \eta(\lambda) - \big(\epsilon_1(\lambda) + 2\epsilon_2(\lambda) + \epsilon_4(\lambda)\big).
\end{aligned}$$

Therefore, $\mathcal{B}$ is successful in breaking the statistical HVZK property of the underlying 3-move SS-HVZK protocol with non-negligible advantage $\eta'(\lambda) = \eta(\lambda) - \big(\epsilon_1(\lambda) + 2\epsilon_2(\lambda) + \epsilon_4(\lambda)\big)$. This leads us to the conclusion that the protocol in Figure 1 is a PoWorK with statistical indistinguishability.

□

Remark. Theorem 1 can be extended to encompass the case where the protocol Π to be compiled in the construction described in Figure 1 achieves $T(\lambda)$-*computational HVZK*, i.e. it is HVZK for every verifier $\mathcal{B}$ which runs in $T(\lambda)$ steps. Specifically, in the indistinguishability proof the running time of the HVZK adversary $\mathcal{B}$ is (in number of steps) bounded by:

$$\mathsf{Steps}_{\mathcal{V}^*}(\langle(\mathsf{P1}_\Pi, \mathsf{P2}_\Pi)(w), \mathsf{Ver}_\Pi(\tilde{c})\rangle(x,z,h)) + \mathsf{Steps}_{\mathsf{Inv}}(c) + \mathsf{Steps}_\oplus(c^{-1}, \tilde{c}) + \mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}).$$

Therefore, we can prove that if $T(\lambda)$ is an asymptotically larger function than the time of the puzzle solving algorithm, then our dense puzzle based construction achieves computational indistinguishability.

Define $\mathcal{PS}_\lambda = \{0,1\}^\lambda$, $\mathcal{SS}_\lambda = \{0,1\}^\lambda$, and $\mathcal{HS}_\lambda = [\log^2 \lambda, \lambda/4]$. Let $H(\cdot) := \mathsf{LSB}_{\lambda/2}(\mathcal{O}(\cdot))$, where LSB_k stands for k least significant bits.

- $\mathsf{Sample}(1^\lambda, h)$: Return $\mathsf{puz} \leftarrow \{0,1\}^\lambda$.
- $\mathsf{SampleSol}(1^\lambda, h)$: Pick random $x \leftarrow \{0,1\}^\lambda$ and $y \leftarrow \{0,1\}^{\lambda/2}$. Return $\mathsf{puz} = (H(x,y), y)$ and $\mathsf{soln} = x$.
- $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$:
  - Parse puz to (z, y); set $\mathsf{soln} = \perp$ and initialize an empty set X.
  - For $ctr = \{1, \ldots, 2^{h+2\log\lambda}\}$:

    Randomly pick $x \leftarrow \{0,1\}^\lambda \setminus X$, and add x to X. Set $\mathsf{soln} = x$ if $\mathsf{LSB}_h(z) = \mathsf{LSB}_h(H(x,y))$.
  - Return soln.
- $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln})$: Parse puz to (z, y). Return true if and only if $\mathsf{LSB}_h(z) = \mathsf{LSB}_h(H(\mathsf{soln}, y))$.

Figure 3: The Dense Puzzle System from the Random Oracle $\mathcal{O}$.

3.4 Dense Puzzle Instantiation in the Random Oracle Model

We now instantiate a dense puzzle system in the random oracle model. For a given security parameter λ, let $\mathcal{O} : \{0,1\}^* \mapsto \{0,1\}^m$ be a random oracle, where $m \geq \lambda/2$. Our dense puzzle system is described in Figure 3.

Theorem 2 *Let $\lambda \in \mathbb{Z}^+$ be the security parameter. Define $\mathcal{PS}_\lambda = \{0,1\}^\lambda$, $\mathcal{SS}_\lambda = \{0,1\}^\lambda$, and $\mathcal{HS}_\lambda = [\log^2 \lambda, \lambda/4]$. Let $\mathcal{O}$ be a random oracle mapping from $\{0,1\}^*$ to $\{0,1\}^m$, where $m \geq \lambda/2$. For any $h \in \mathcal{HS}_\lambda$, the puzzle system* PuzSys *described in Figure 3 is correct, complete with* Solve*'s running time $2^{h+2\log\lambda}$, efficiently sampleable, statistically indistinguishable, and g-hard, where $g(T) = T^{1/c}$, for any constant $c > 2$. In addition, for any k that is $O(2^{\lambda/8})$,* PuzSys *is $(\mathsf{id}(\cdot), k)$-amortization resistant, where $\mathsf{id}(\cdot)$ is the identity function.*

Proof: See Appendix B.1. □

3.5 Dense Puzzle Instantiation From Complexity Assumptions

In this section, we show how to construct a puzzle system whose puzzle instance distribution is statistically close to the uniform distribution (over $\{0,1\}^{m(\lambda)}$) without random oracles. The main challenge is, given an arbitrary oneway function $\psi : \mathcal{X} \mapsto \mathcal{Y}$, to build another oneway function with uniform output distribution (on random inputs) while still maintaining its onewayness. As an intuition, we would like to first map the output of the given oneway function from $\mathcal{Y}$ to $\{0,1\}^\ell$ using an efficient injective map (which is usually the bit representation of $y \in \mathcal{Y}$), and then apply a strong extractor on it. Let $\mathsf{Ext} : \{0,1\}^\ell \times \{0,1\}^d \mapsto \{0,1\}^m$ be a strong extractor as defined at Definition 3.

Definition 3 *Function* $\mathsf{Ext} : \{0,1\}^\ell \times \{0,1\}^d \mapsto \{0,1\}^m$ *is (t, ϵ)-strong extractor if for any t-source X (over $\{0,1\}^\ell$), we have $\Delta[(S, \mathsf{Ext}(X,S)), (S, \mathbf{U}_m)] \leq \epsilon$, where $S \leftarrow \{0,1\}^d$ and $\mathbf{U}_m \leftarrow \{0,1\}^m$ are drawn uniformly and independently of X.*

The new oneway function is defined as $\psi^U : \mathcal{X} \times \{0,1\}^d \mapsto \{0,1\}^m \times \{0,1\}^d$ is defined as $\psi^U(x,s) = (\mathsf{Ext}(\psi(x), s), s)$. According to LHL [HILL93], if $H_\infty(x) \geq m + 2\log(1/\epsilon)$, then the output of ψ^U is at most ϵ-far from the uniform distribution over $\{0,1\}^{m+d}$. However, in order to maintain its onewayness, we need an extra property of the strong extractor – *Target Collision Resistance* (TCR), i.e. given x and s, it is computationally infeasible to find x' such that $x \neq x'$ and $\mathsf{Ext}(x,s) = \mathsf{Ext}(x',s)$. We focus on this in the next subsection and then we return to our construction.

3.5.1 TCR Strong Extractors from Regular UOWHFs.

We first formally define the TCR property for a randomly indexed strong extractor in Definition 4.

Definition 4 *Let* $\mathsf{Ext} : \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{d(\lambda)} \mapsto \{0,1\}^{m(\lambda)}$ *be a strong extractor. We say* Ext *is target collision resistant if for all PPT adversary* $\mathcal{A}$*, the following probability:*

$$\Pr\left[\begin{array}{l} x \leftarrow \mathcal{A}(1^\lambda); s \leftarrow \{0,1\}^{d(\lambda)} : x' \leftarrow \mathcal{A}(s) : \\ x, x' \in \{0,1\}^{\ell(\lambda)} \wedge x \neq x' \wedge \mathsf{Ext}(x,s) = \mathsf{Ext}(x',s) \end{array}\right] = \mathsf{negl}(\lambda).$$

A stronger notion, *collision resistant extractors*, was introduced by Dodis [Dod05]. Collision resistant extractors were applied to construct *perfectly oneway probabilistic hash functions* proposed [CMR98] in 2005. The construction of such collision resistant extractors relies on a variant of leftover hash lemma proved by Dodis and Smith [DS05] that we recap, for completeness, in Lemma 1.

Lemma 1 ([DS05]) *Let* $f : \{0,1\}^N \mapsto \{0,1\}^m$ *be an arbitrary function. Let* $\mathcal{H} = \{H_i | i \in \mathcal{I}\}$ *be a pairwise independent hash function family with key space* $\mathcal{I}$*, domain* $\{0,1\}^n$ *and range* $\{0,1\}^N$*. If* X *is a* t*-source over* $\{0,1\}^n$ *with* $t \geq m + 2\log(1/\epsilon) + 1$*, then we have*

$$\Delta[(I, f(H_I(X))), (I, f(\mathbf{U}_N))] \leq \epsilon$$

where $I \leftarrow \mathcal{I}$ *and* $\mathbf{U}_N \leftarrow \{0,1\}^N$ *are drawn uniformly and independently of* X*.*

Our observation is that in the same way that [Dod05] employ regular collision resistant hash functions (CRHF) to derive collision resistant strong extractors, we can use regular universal oneway hash function (UOWHF), to obtain TCR strong extractor. The notion of UOWHF was initially proposed by Naor and Yung [NY89] where they showed that UOWHFs can be constructed by composing oneway permutations with (weakly) pairwise independent hash functions. Since then, many constructions of UOWHFs have been proposed, assuming the existence of regular oneway functions [SY90] or any oneway functions [Rom90, HHR+10].[2] We recall the definition of UOWHF as Definition 5.

Definition 5 *A family of functions* $\mathcal{F}_\lambda = \left\{F_i : \{0,1\}^{\ell_1(\lambda)} \mapsto \{0,1\}^{\ell_2(\lambda)} | \forall i \in \{0,1\}^\lambda\right\}$ *is a family of universal oneway hash functions if it satisfies:*

- *Efficiency: Given* $i \in \{0,1\}^\lambda$ *and* $x \in \{0,1\}^{\ell_1(\lambda)}$*,* $F_i(x)$ *can be evaluated in time* $\mathrm{poly}(\ell_1(\lambda), \lambda)$*.*
- *Compressing:* $\ell_2(\lambda) < \ell_1(\lambda)$*.*
- *Target Collision Resistance: For all PPT* $\mathcal{A}$*, the following is negligible in* λ*:*

$$\Pr[x \leftarrow \mathcal{A}(1^\lambda); i \leftarrow \{0,1\}^\lambda; x' \leftarrow \mathcal{A}(i) : x, x' \in \{0,1\}^{\ell_1(\lambda)} \wedge x \neq x' \wedge F_i(x) = F_i(x')].$$

We would like to use $\mathcal{H}_{2n} = \left\{H_{(a,b)}(x) = ax + b | \forall a \neq 0, a, b \in \mathbb{GF}(2^n)\right\}$ as the family of pairwise independent permutations and a regular UOWHF family $\mathcal{F}_\lambda$ to construct our TCR strong extractors. Define $\hat{F}_i(\cdot) := (F_i(\cdot), i)$, where $F_i \in \mathcal{F}_\lambda$. Our TCR strong extractor is constructed as $\mathsf{Ext}(x, (i,s)) = \hat{F}_i \circ H_s(x)$. Note that regularity of the UOWHFs is important to ensure that the output distribution of such strong extractors is close to the uniform distribution, as $F_i(U_{\ell_1(\lambda)}) \equiv U_{\ell_2(\lambda)}$. On the other hand, some UOWHF constructions give regular UOWHFs by default (i.e., the UOWHFs constructed by the oneway permutation based approach [NY89]).

[2]We note that, on the contrary, CR strong extractors cannot be built from arbitrary oneway functions, since Simon [Sim98] gave a black-box separation between CRHFs and oneway functions.

Theorem 3 *Let $\ell(\lambda), m(\lambda)$ be polynomials. Let $\mathcal{H}_{2\cdot\ell(\lambda)} = \left\{H_s : \{0,1\}^{\ell(\lambda)} \mapsto \{0,1\}^{\ell(\lambda)} | \forall s \in \{0,1\}^{2\cdot\ell(\lambda)}\right\}$ be a pairwise independent permutation family. Assume that $\mathcal{F}_\lambda = \left\{F_i : \{0,1\}^{\ell(\lambda)} \mapsto \{0,1\}^{m(\lambda)} | \forall i \in \{0,1\}^{\lambda}\right\}$ is a regular UOWHF family. Then,*

$$\mathsf{Ext}_\lambda(x,(i,s)) = (F_i(H_s(x)), i)$$

is a (t,ϵ)-TCR strong extractor from $\{0,1\}^{\ell(\lambda)} \times \{0,1\}^{\lambda+2\cdot\ell(\lambda)}$ to $\{0,1\}^{\lambda+m(\lambda)}$, for any constant $t \geq m(\lambda) + \lambda + 2\log(1/\epsilon) + 1$.

Proof: See Appendix B.2. □

3.5.2 Dense Oneway Functions and Dense Puzzles from Complexity Assumptions.

Armed with the TCR strong extractor from the previous section we return now to our construction. The key to the construction will be a "dense" oneway function: a oneway function is ϵ-*dense* oneway if its output distribution is at most ϵ-far from $\mathbf{U}_m$ for some $m \in \mathbb{Z}^+$. We now present a transformation of a one-way function to a dense one-way function via the application of a TCR-strong extractor. The TCR property will ensure that any attempt to invert the dense one-way function will result to an inversion of the underlying one-way function. Formally we prove the following.

Theorem 4 *Let $\lambda_1, \lambda_2 \in \mathbb{Z}^+$ be the security parameters. Let $\psi_{\lambda_1} : \mathcal{X}_{\lambda_1} \mapsto \mathcal{Y}_{\lambda_1}$ be an arbitrary oneway function, and define $H_{\lambda_1} = H_\infty(\psi_{\lambda_1}(X))$ for random variable X drawn uniformly from $\mathcal{X}_{\lambda_1}$. Assume there exists an efficient injective map $\zeta_{\lambda_1} : \mathcal{Y}_{\lambda_1} \mapsto \{0,1\}^{\ell(\lambda_2)}$. If*

$$\mathsf{Ext}_{\lambda_2}(x,(s_1,s_2)) : \{0,1\}^{\ell(\lambda_2)} \times \{0,1\}^{\lambda_2+2\cdot\ell(\lambda_2)} \mapsto \{0,1\}^{H_{\lambda_1}-2\log(1/\epsilon)-1}$$

is a (H_{λ_1},ϵ)-TCR strong extractor, $\psi^U_{\lambda_1,\lambda_2}(x,s_1,s_2) = (\mathsf{Ext}_{\lambda_2}(\zeta_{\lambda_1}(\psi_{\lambda_1}(x)),(s_1,s_2)), s_2)$ is an ϵ-dense oneway function with range $\{0,1\}^{2\cdot\ell(\lambda_2)+H_{\lambda_1}-2\log(1/\epsilon)-1}$ and domain $\mathcal{X}_{\lambda_1} \times \{0,1\}^{\lambda_2+2\cdot\ell(\lambda_2)}$.

Proof: See Appendix B.3. □

The above result paves the way for constructing dense puzzles from complexity assumptions. Essentially, given a function with moderately hard characteristics making it suitable for a puzzle, it is possible to transform it to a dense puzzle by applying a suitably hard TCR extractor ("suitable" here means that breaking the TCR property should be harder than solving the puzzle). We now illustrate this methodology by applying it to the discrete logarithm problem. More generally this methodology transforms any puzzle in the sense of Definition 1 to a dense puzzle (assuming again a suitably hard TCR extractor).

3.5.3 The DLP Based Puzzle and Calibrating Its Hardness.

Consider the discrete logarithm problem (DLP) as the candidate oneway function for our puzzle. Let $\mathbb{G} = \langle G \rangle$ be some (multiplicative) cyclic group where the DLP is hard, and G is a generator with order p, which is a λ_1-bit prime. The oneway function $\psi_G : \mathbb{Z}_p \mapsto \mathbb{G}$ is defined as $\psi_G(x) = G^x$. It is shown by Shoup [Sho97] that any probabilistic algorithm takes $\Omega(\sqrt{p})$ steps to solve the DLP over generic groups. Analogously, [GJKY13] shows any probabilistic algorithm must take at least $\sqrt{2p\epsilon}$ steps to solve DLP with probability ϵ in the generic group model. To build a puzzle, we would like to calibrate the hardness of the DLP by revealing the most significant bits of the pre-image. For example, for a puzzle with hardness factor$h \leq \lfloor \frac{\lambda_1-1}{2} \rfloor$, we pick $x \in \{0,1\}^h$ and $y \in \{0,1\}^{\lfloor(\lambda_1-1)/2\rfloor}$ uniformly at random, and set the puzzle as $(\mathsf{Ext}_{\lambda_2}(\psi_G(x+2^h \cdot y),(s_1,s_2)), s_2, y)$. We assume the calibrated DLP is still moderately hard with respect to the min-entropy of x. Note that similar assumption was used by Gennaro to construct a more efficient pseudo-random generator [Gen00]. It is easy to see that this assuption holds for DLP in generic groups, i.e. given $\psi_G(x + 2^h \cdot y)$ and y, the best generic algorithm must take at least $\sqrt{2^{h+1}\epsilon}$ steps to solve DLP with probability ϵ. We note that this problem is closely related to leakage-resilient cryptography [AM11, ADVW13], but due to space limitation we omit the detailed discussion here.

Define $\mathcal{PS}_\lambda = \{0,1\}^{7\lambda/2+\log^4\lambda}$, $\mathcal{SS}_\lambda = \{0,1\}^{\log^4\lambda}$, and $\mathcal{HS}_\lambda = [\log^4\lambda + \log^2\lambda + 1, \log^5\lambda]$. For the given λ, select a pre-defined $\mathsf{Ext}_\lambda : \{0,1\}^\lambda \times \{0,1\}^{3\lambda} \mapsto \{0,1\}^{\lambda+\log^4\lambda}$. Set the DLP $\psi_G : \mathbb{Z}_p \mapsto \mathbb{G}$ over the pre-defined elliptic curve, where p is λ-bit prime such that there exists an efficient injective map $\zeta : \mathbb{G} \mapsto \{0,1\}^\lambda$. (We will omit this map ζ in the rest of the description for notation simplicity.)

- $\mathsf{Sample}(1^\lambda, h)$: Return $\mathsf{puz} \leftarrow \{0,1\}^{7\lambda/2+\log^4\lambda}$.
- $\mathsf{SampleSol}(1^\lambda, h)$:
  - Pick random $s_1 \leftarrow \{0,1\}^\lambda$, $s_2 \leftarrow \{0,1\}^{2\lambda}$, $x \leftarrow \{0,1\}^h$ and $y \leftarrow \{0,1\}^{\lambda/2}$.
  - Return $\mathsf{puz} = (\mathsf{Ext}_\lambda(\psi_G(x + 2^h \cdot y), (s_1, s_2)), s_2, y)$ and $\mathsf{soln} = x$.
- $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$:
  - Parse puz to (z, s_1, s_2, y); set $\mathsf{soln} = \perp$ and initialize an empty set X.
  - For $ctr = \{1, \ldots, 2^h\}$:
    - ∘ Randomly pick $x \leftarrow \{0,1\}^h \setminus X$, and add x to X.
    - ∘ Set $\mathsf{soln} = x$ if $z = \mathsf{Ext}_\lambda(\psi_G(x + 2^h \cdot y), (s_1, s_2))$.
  - Return soln.
- $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln})$: Parse puz to (z, s_1, s_2, y). Return true if and only if $z = \mathsf{Ext}_\lambda(\psi_G(\mathsf{soln} + 2^h \cdot y), (s_1, s_2))$.

Figure 4: The Dense Puzzle System From DLP.

On the other hand, due to the out-layer extractor, we cannot directly adopt any known (generic) DLP algorithms, such as [GTY07, GPR13]. Instead, our puzzle solver just exhaustively searches for a valid solution. There is a subtle caveat, namely the expected running time of solving a puzzle with hardness factor$h$, i.e. $x \leftarrow \{0,1\}^h$ is designed to be 2^h, whereas the TCR property of UOWHF is only guaranteed against PPT adversaries with respect to λ_2 (the security parameter of the UOWHF). To address this issue, we introduce an additional assumption, that is the expected running time of any adversary $\mathcal{A}$ (in number of steps) can break the TCR property of the underlying UOWHF with non-negligible probability on $x \leftarrow \{0,1\}^h$ is $\omega(2^{h/2})$, (i.e. breaking TCR is expected to happen after the birthday paradox bound). The dense puzzle system from DLP (combining with TCR strong extractors) is depicted in Figure 4.

Theorem 5 *Let $\lambda \in \mathbb{Z}^+$ be the security parameter and $h \in [\log^4\lambda + \log^2\lambda + 1, \log^5\lambda]$ be the hardness factor. Let $\mathsf{Ext}_\lambda : \{0,1\}^\lambda \times \{0,1\}^{3\lambda} \mapsto \{0,1\}^{\lambda+\log^4\lambda}$ be a TCR strong extractor such that the expected running time of any adversary $\mathcal{A}$ that breaks its TCR property with non-negligible probability on $x \leftarrow \{0,1\}^h$ is $\omega(2^{h/2})$. Assume $\psi_G : \mathbb{Z}_p \mapsto \mathbb{G}$ is a hard DLP in generic groups such that the best generic algorithm must take at least $\sqrt{2^{h+1}\varepsilon}$ steps to solve it with probability ε. The puzzle system $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{SampleSol}, \mathsf{Solve}, \mathsf{Verify})$ described in Figure 4 is correct, complete with Solve's running time 2^h, efficiently samplable, statistically indistinguishable, and g-hard, where $g(T) = T^{1/c}$ for any constant $c > 2$. In addition, for any k that is $O(2^{\log^3\lambda})$, PuzSys is $(\mathsf{id}(\cdot), k)$-amortization resistant, where $\mathsf{id}(\cdot)$ is the identity function.*

Proof: See Appendix B.4. □

Remark. For notation simplicity, we let the puzzle space "independent" of the hardness factor h, therefore we have to limit h within a small interval to ensure (i) $\psi_G(x + 2^h \cdot y)$ has enough entropy and (ii) it is infeasible to break the TCR property of the underlying UOWHF within $2^{h/2}$ steps. In practice, for any desired h, we can always pick a suitable $\mathsf{Ext}_\lambda : \{0,1\}^\lambda \times \{0,1\}^{3\lambda} \mapsto \{0,1\}^{\lambda+h-\log^2\lambda-1}$.

3.6 Instantiation of the Dense Puzzle Based PoWorK.

We describe an instantiation of our PoWorK protocol as described in Figure 1 built upon the Schnorr identification scheme [Sch89] and the dense puzzle system instantiation in the RO model[3] (see Section 3.4). For completeness, we provide a description of the Schnorr protocol in Appendix A.2. We denote our instantiation by $\mathbf{\Pi}^*$. We fix a security parameter λ and a hardness factor $h \in [\log^2 \lambda, \lambda/4]$. The challenge and puzzle spaces are all set to $\mathcal{CS}_\lambda = \mathcal{CS}_\Pi = \mathcal{PS}_\lambda = \{0,1\}^\lambda$. We choose a random prime q s.t. $2^\lambda \leq q$. We select the parameters and the statement of the Schnorr protocol to be $(q, g, x = g^w)$. We pick a hash function $H_\lambda : \{0,1\}^* \longrightarrow \{0,1\}^\lambda$. The group operator $\oplus$ is the bitwise XOR operation. The PoWorK protocol $\mathbf{\Pi}^*$ consists of the two following stages:

1. **Protocol execution:** The two modes of $\mathbf{\Pi}^*$ are:
   - **PoK mode**. *1st move*: $\mathcal{P}(w)$ selects a random ρ in $\mathbb{Z}_q$ and sends $\tilde{a}$ to $\mathcal{V}$. *2nd move*: $\mathcal{V}$ sends a challenge c selected uniformly at random to $\mathcal{P}$. *3rd move*: $\mathcal{P}$ chooses random $s \leftarrow \{0,1\}^\lambda$ and $y \leftarrow \{0,1\}^{\lambda/2}$; it computes $t = (\mathsf{LSB}_{\lambda/2}(H(s,y)), y)$ and $\tilde{c} = c \oplus t$. It sends $(\tilde{c}, \tilde{r}, s, t)$ to $\mathcal{V}$, where $\tilde{r} = \rho + \tilde{c}w$.
   - **PoW mode**. *1st move*: $\mathcal{P}$ runs $(\tilde{a}, \tilde{c}, \tilde{r}) \leftarrow \mathsf{Sim}_\Pi(x, \tilde{c})$ and sends $\tilde{a} = g^\rho$ to $\mathcal{V}$. *2nd move*: $\mathcal{V}$ sends a random challenge c to $\mathcal{P}$. *3rd move*: $\mathcal{P}$ computes $t = c^{-1} \oplus \tilde{c}$ and runs $\mathsf{Solve}(1^\lambda, h, t)$; if the puzzle solver outputs a value s, then $\mathcal{P}$ sends $(\tilde{c}, \tilde{r}, s, t)$ to $\mathcal{V}$, otherwise it aborts the protocol.
2. **Verification:** The verifier checks that (1) $\tilde{c} = c \oplus t$; (2) $g^{\tilde{r}} = \tilde{a}x^{\tilde{c}}$; (3) parses t as (t_1, t_2), where $t_1, t_2 \in \{0,1\}^{\lambda/2}$ and checks that $\mathsf{LSB}_h(t_1) = \mathsf{LSB}_h(H(s, t_2))$.

We observe that since (a) the RO puzzle instantiation is correct and complete and (b) all spaces are set to $\{0,1\}^\lambda$, $\mathbf{\Pi}^*$ achieves completeness. Moreover, the puzzle sampling distribution is close to uniform $\{0,1\}^\lambda$, which is also the challenge distribution in $\mathbf{\Pi}^*$. Therefore, assumptions (**A**), (**B**) in Figure 2 hold. In addition, the running time of the puzzle solver is $2^{h+2\log\lambda} \geq 2^{\log^2\lambda+2\log\lambda}$ which strongly dominates the linear time complexity of performing $\oplus$ operations or sampling uniformly at random, i.e. assumption (**C**) in Figure 2 also holds. Thus, by Theorems 2 and 1, we have that $\mathbf{\Pi}^*$ is $\sqrt[\nu]{(\cdot)}$-sound, for any $\nu > 2$. The (statistical) indistinguishability of $\mathbf{\Pi}^*$ is achieved by the perfect ZK simulation of the Schnorr protocol and the assumptions (**A**), (**B**).

4 Applications

Below we present some practical and theoretical applications of our PoWorK. When using PoWorK in practice we must ensure that the verifier cannot distinguish between the two types of provers based on their response time. In Section 2.2 we argued that for our indistinguishability proofs, $\mathcal{P}(w)$ (i.e. the prover who knows the witness) should perform some idle steps so that his running time will be lower bounded by the time that one would need to solve the puzzle. However, enforcing a real user to wait is not ideal. Luckily though, the time needed for a prover who solves a puzzle (i.e., does not know the witness) depends on his total computational power and on whether the puzzle is parallelizable or not. Provers who own specialized hardware (e.g., based on ASICs) or that have access to powerful computer clusters (in case that a puzzle is parallelizable) might be able to solve the puzzle very fast – paying of course the relevant computation cost. Thus, when applying PoWorKin practice, the time that takes a prover to respond a challenge is not a distinguishing factor: the prover might have as well solved the puzzle in constant time by fully parallelizing its computation. However, we do care that the prover has paid the corresponding computational cost and he is not able to amortize a previous solution of a puzzle to solve a new one.

4.1 Email Spam Application

Using proofs of work to reduce the amount of spam email was suggested back in 1992 by Dwork and Naor [DN92]. Their idea can be summarized in the following:

[3] The construction using the DLP based puzzle system is similar. We chose to employ the RO instantiation for simplicity in presentation.

> "If I don't know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message" [DN92].

In their proposal there exists some special software[4] that operates on behalf of the receiver and checks whether the sender has properly computed the proof of work or the sender is an *approved* (by the receiver) *contact*. The reason that this approach helps to reduce spam is mainly economic: in order for spammers to send high volumes of emails they would have to invest in powerful computational resources which makes spamming non cost-effective.

A disadvantage of the method described above is that the list of the approved contacts (i.e. email addresses) of the receiver has to be given to this special software/mail server in order to check whether the sender belongs in this list or not - in which case she will have to perform additional computation. This violates the privacy of the receiver who needs to reveal which of her contacts she considers to be approved and thus allows them to send emails "for free". Adopting our PoWorK protocol would give a *privacy preserving* solution to the spam problem: given the indistinguishability feature of PoWorK, the software/verifier does not need to know the approved list of contacts, in fact it does not even need to know whether the incoming email is from an approved contact or a non-approved user who successfully fulfilled the computational work.

Non-interactive PoWorKs. Sending an email should not require any extra communication between the sender and the mail server. Our 3-move PoWorK is public-coin, thus can be turned into non-interactive by applying the Fiat-Shamir transformation [FS86]. Namely, the prover, instead of receiving a challenge from the verifier, hashes the first move message $\mathbf{a}$ together with the context of the email and the email address of the receiver into $\mathbf{c}$, and provides the verifier with the whole proof, π, which includes $(\mathbf{a}, \mathbf{c}, \mathbf{r})$ and the context of the email, in one round.

Multi-witness hard relation. In order for a user to approve a list of contacts she will have to provide each one of them with a unique witness for the same statement (in order to ensure indistinguishability). Let $R_{\mathcal{L}}$ be a multi-witness hard relation with a trapdoor for a language $\{x \mid \exists w : (x, w) \in R_{\mathcal{L}}\}$. A relation is said to be hard if for $(x, w) \in R_{\mathcal{L}}$, a PPT adversary given x can only output w' s.t. $(x, w') \in R_{\mathcal{L}}$ with negligible probability. A multi-witness hard relation *with a trapdoor* is described by the following algorithms: (a) a trapdoor generation algorithm sets a pair of a statement x and associated trapdoor t: $(x, t) \leftarrow \mathsf{GenT}(R_{\mathcal{L}})$, (b) an efficient algorithm GenW that on input $x \in \mathcal{L}$ and a trapdoor t outputs a witness w such that $(x, w) \in R_{\mathcal{L}}$ and, (c) a verification algorithm $1/0 \leftarrow \mathsf{Ver}(R_{\mathcal{L}}, x, w)$ outputs 1 if $(x, w) \in R_{\mathcal{L}}$ and 0 otherwise [5].

PoWorK based spam reducing system. Consider a PoWorK scheme as presented in Figure 1 for a security parameter λ, a puzzle system PuzSys and a multi-witness hard relation with a trapdoor $R_{\mathcal{L}}$ as described above. A spam reducing system SRS consists of the following algorithms:

- *MailServerSetup*(1^λ): the mail server $\mathcal{S}_{mail}$ on input the security parameter, λ, selects the hardness of the puzzle system $h \in \mathcal{HS}_\lambda$.
- *ReceiverSetup*$(1^\lambda, h)$: user $\mathcal{R}$ (i.e. the receiver) runs $(x, t) \leftarrow \mathsf{GenT}(R_{\mathcal{L}}$ and sends x and her email address $ad_{\mathcal{R}}$ to the mail server (potentially signed together). The trapdoor t is secretly stored by $\mathcal{R}$.
- *ApproveContact* (t, x): in order for $\mathcal{R}$ to approve a sender $\mathcal{S}$, it will run $w \leftarrow \mathsf{GenW}(t, x)$ and will give $w \in R_{\mathcal{L}}(x)$ to the sender (unique witnesses allow for revocation as discussed below). From now on, $\mathcal{S}$ can use w to send emails to $\mathcal{R}$.
- *SendEMail*(w, h, x): a sender $\mathcal{S}$ with input the public parameters v, statement $x \in \mathcal{L}$ and with a private input $w \in R_{\mathcal{L}}(x) \cup \{\perp\}$, prepares a PoWorK proof $\pi = (\mathbf{a}, \mathbf{c}, \mathbf{r})$. If $\mathcal{S}$ is an approved contact of $\mathcal{R}$, then she will use the witness w to perform the PoK side of PoWorK, while if $\mathcal{R}$ is not an approved contact (i.e. $w = \perp$)

[4] This special software could for example run on the receiver's mail server or be an independent program running on the receiver's side.

[5] Examples of multi-witness hard relations with trapdoors are (a) the DL representation problem [Bra94, BF99] over prime order groups, (b) the representation problem in composite modular groups [ACJT00] which has constant size parameters in the number of adversarial parties.

she will have to execute the PoW side. To compute π non-interactively she will fix c to be $H(\mathbf{a}, m)$, where $\mathbf{a}$ is the first message of PoWorK, m stands for the body of the email[6], and H is a hash function. The rest of PoWorK is computed as before.

- *ApproveEMail*(h, x, π): is run by the mail server $\mathcal{S}_{mail}$ who verifies π and outputs $0/1$. If proof is π valid, then $\mathcal{S}_{mail}$ forwards the enclosed email to $\mathcal{R}$.

Note that our proposal, similar to [DN92, DGN03], requires to implement additional protocols between the sender and the recipient (i.e. a change in the internet mail standards would be required).

Security. Although a formal definition and description of properties of an email system is out of the scope of this paper, we do define and prove *spam resistance* and *privacy*. Briefly, spam resistance guarantees that the mail server will allow an email message to reach the recipient if and only if a valid proof (of work or knowledge) has been attached. At the same time for a non-approved contact the number of valid proofs of work prepared should not affect the time required to prepare a new one (similar to puzzle amortization property). Privacy implies that the mail server cannot distinguish whether the sender of a message is an approved contact of the recipient or not. More formally:

Definition 6 *Let* SRS *be a spam reducing system built upon a PoWorK* $(\mathcal{P}, \mathcal{V})$ *for a language* $\mathcal{L} \in \mathcal{NP}$ *and a puzzle system* $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$. *We define spam resistance and privacy of* SRS *as follows:*

(i). (σ, k)***-Spam Resistance:*** *We say that* SRS *is* (σ, k)-spam resistant *if there exists a PPT witness-extraction algorithm* $\mathcal{K}$, *such that for every hardness factor* $h \in \mathcal{HS}_\lambda$, *auxiliary tape* $z \in \{0,1\}^*$ *and every adversary* $\mathcal{A}$, *if for non-negligible functions* $\alpha_1(\cdot), \alpha_2(\cdot)$*:*

$$\Pr\left[\begin{array}{l} (t,x) \leftarrow \textit{ReceiverSetup}(1^\lambda, h); \forall 1 \le i \le k : \mathsf{puz}_i \leftarrow \mathsf{Sample}(1^\lambda, h); \\ \{\pi_i = (\mathbf{a}_i, \mathbf{c}_i, \mathbf{r}_i)\}_{i\in[k]} \leftarrow \mathcal{A}(z, 1^\lambda, h, x) : \\ \big(\forall 1 \le i \le k : \ \textit{ApproveEMail}(h, x, \pi_i) = 1\big) \wedge \\ \wedge (\forall i \neq j \in [k] : \pi_i \neq \pi_j) \wedge \\ \wedge \Big(\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, x) \le \sigma\big(\sum_{i=1}^{k} \mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i)\big)\Big) \end{array}\right] = \alpha_1(\lambda) ,$$

then

$$\Pr[\mathcal{K}^{\mathcal{A}}(z, 1^\lambda, h, x) \in R_{\mathcal{L}}(x)] = \alpha_2(\lambda) .$$

(ii). ***Privacy:*** *We say that* SRS *is* private, *if for every hardness factor* $h \in \mathcal{HS}_\lambda$, *auxiliary tape* $z \in \{0,1\}^*$ *and every adversarial mail server* $\mathcal{A}$, *it holds that:*

$$\left|\Pr\left[\begin{array}{c} (t,x) \leftarrow \textit{ReceiverSetup}(1^\lambda, h); w \leftarrow \textit{ApproveContact}(t, x); \\ \pi \leftarrow \textit{SendEMail}(w, h, x) : \mathcal{A}(z, h, x, \pi) = 1 \end{array}\right] - \right.$$
$$\left. - \Pr\left[\begin{array}{c} (t,x) \leftarrow \textit{ReceiverSetup}(1^\lambda, h); \\ \pi \leftarrow \textit{SendEMail}(\perp, h, x) : \mathcal{A}(z, h, x, \pi) = 1 \end{array}\right]\right| = \mathsf{negl}(\lambda) .$$

We state the following theorem for a *private spam reducing* email system:

Theorem 6 *Let* SRS *be a spam reducing system built upon dense puzzle-based PoWorK* $(\mathcal{P}, \mathcal{V})$ *for a* g*-hard and* (τ, k)*-amortization resistant dense puzzle system* $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$, *where* k *is polynomial in* λ, τ *is an increasing function and* g *is a subadditive function. Let* H *be a hash function with output domain equal to challenge sampling space* $\mathcal{CS}_\lambda$ *modeled as a random oracle. Assume that the worst-case running time of* $\mathsf{Solve}(1^\lambda, \cdot, \cdot)$ *is* $o(|\mathcal{CS}_\lambda|)$ *and that* $(\sqrt{\tau \circ g}(\mathsf{Solve}(1^\lambda, \cdot, \cdot))$ *is superpolynomial in* λ. *Then, the email system described above is* private *and* $(\sqrt{\tau \circ g}, k)$-spam resistant.

[6] We can assume that the email body also contains a time-stamp (or that the time-stamp is added later by the mail server) and also includes $(ad_{\mathcal{S}}, ad_{\mathcal{R}})$ which are the sender/receiver email addresses

Intuitively the privacy holds because of the indistinguishability of PoWorK (we can use the adversary of SRS privacy to build an adversary that breaks the indistinguishability of PoWorK). The $(\sqrt{\tau \circ g}, k)$-*spam resistance* property briefly holds because of the soundness of PoWorK and the amortization resistance of the underlying PuzSys. We provide a proof sketch of Theorem 6 in Appendix D.

Extensions. We finally discuss some interesting extensions of our spam reducing application:

Revocation. We could possibly use standard anonymous revocation schemes [CL02, LPY12] on top of our email construction. The idea is similar to group signatures authorization: whenever a receiver approves a user (i.e. adds the user to the group of approved contacts) she also provides her with a membership credential. The receiver has to periodically update a public list of revoked (or unrevoked) users and, whenever a sender wishes to send an email, she will also have to include a proof of non-revocation together with π (which can be done anonymously to preserve the privacy against the mail server).

Preventing witness sharing (transferability). Another possible extension would be to guarantee that a user/receiver is not sharing her witness with more users. A possible way to address this problem is to use the techniques that were proposed by Kiayias and Tang [KT13] and construct a *leakage-deterring* cryptographic function $\mathcal{F}$ that on input a user's witness it outputs some private information associated with it. Whenever a user obtains a witness, this is associated with some private information of the user (e.g.. a credit card number). $\mathcal{F}$ is constructed in such a way that when it receives w as input, outputs the information associated to it. Thus, when a malicious user shares his unique witness, anyone who receives it can find the user's private information.

Performing useful work. It would be very appealing if the computational power consumed by a PoW user to solve a puzzle, was actually used towards some sort of useful work. A possible idea would be to use a *volunteer computing* service [7] as a work provider, WP, that generates the puzzles to be solved. Then, one could use our Lapidot-Shamir based PoWorK that requests that the PoK prover solves a puzzle selected by the verifier (refer to Appendix C). The verifier can pick a random puzzle from the work provider, WP, and once the prover has the solution can submit it back to WP. Assuming that the verifier and the work provider are not colluding, the privacy of the prover is maintained.

4.2 3-move Straight-line Concurrent Simulatable Arguments of Knowledge

In the following, we present a theoretical application of PoWorKs. Namely, we show that any PoWorK protocol that satisfies a couple of reasonable assumptions, implies straight-line concurrent $(\lambda^{\mathrm{poly}(\log \lambda)})$-simulatable arguments of knowledge. To prove it, we use the results of Pass [Pas03, Pas04] who has shown that protocols satisfying straight-line simulatability are also straight-line *concurrent* simulatable. Given this proof, we conclude that our 3-move dense puzzle based PoWorK construction, when instantiated with an appropriate puzzle system, is a 3-move straight-line concurrent $\lambda^{\mathrm{poly}(\log \lambda)}$- statistically simulatable argument of knowledge.

The concurrent self-composition Lemma in [Pas04] states that protocols which are straight-line strongly $T(\lambda)$-simulatable (resp. perfectly simulatable) are also straight-line *concurrent* strongly $T(\lambda)$-simulatable (resp. perfectly simulatable). In the Lemma below, we also consider the case of statistical $T(\lambda)$-simulatability. For definitions of straight-line simulatability and straight-line concurrency, we refer the reader to Appendix E.

Lemma 2 (Concurrent Self-Composition [Pas04]) *Let $T(\lambda)$ be a class of functions closed under composition with any polynomial, and let $(\mathcal{P}, \mathcal{V})$ be an interactive argument of knowledge with efficient provers[8]. If $(\mathcal{P}, \mathcal{V})$ is straight-line strongly (resp. statistically) (resp. perfectly) $T(\lambda)$-simulatable, then it is also straight-line concurrent strongly (resp. statistically) (resp. perfectly) $T(\lambda)$-simulatable.*

In the following theorem, we apply Lemma 2 to prove that any 3-move PoWorK is straight-line concurrent statistically $\lambda^{\mathrm{poly}(\log \lambda)}$-simulatable argument of knowledge, when two additional time complexity assumptions hold.

[7] Like the the Berkeley BOINC system `http://boinc.berkeley.edu/` that contributes to scientific research.

[8] I.e., PPT provers that satisfy completeness.

These assumptions are plausible and can be easily met by our dense puzzle based construction when built upon both of our puzzle instantiations, for an appropriate choice of hardness factor.

Theorem 7 *Let $\mathcal{L}$ be a language in $\mathcal{NP}$ and let* PuzSys *be a puzzle system. Let $(\mathcal{P},\mathcal{V})$ be a 3-move f-sound PoWorK for $\mathcal{L}$ and* PuzSys *with statistical indistinguishability such that for every hardness factor $h \in \mathcal{HS}_\lambda$, it holds that:*

(i). $\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h) : f(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) \leq \lambda^{\log\lambda}] = \mathsf{negl}(\lambda)$.

(ii). The worst-case running time of $\mathsf{Solve}(1^\lambda, h, \cdot)$ *is $\lambda^{\mathrm{poly}(\log\lambda)}$ and $\mathcal{P}$ is a polynomial time algorithm that makes oracle calls to* $\mathsf{Solve}(1^\lambda, h, \cdot)$.

Then, $(\mathcal{P},\mathcal{V})$ is a 3-move straight-line concurrent statisticallly $\lambda^{\mathrm{poly}(\log\lambda)}$-simulatable argument of knowledge.

Proof:(sketch) First, we show that $(\mathcal{P},\mathcal{V})$ is a 3-move straight-line statistically $\lambda^{\mathrm{poly}(\log\lambda)}$-simulatable argument of knowledge. Namely, that $(\mathcal{P},\mathcal{V})$ satisifies the following properties:
Completeness. Follows directly from the completeness of $(\mathcal{P},\mathcal{V})$.
Argument of Knowledge. Consider the PPT witness-extraction algorithm $\mathcal{K}$ as in the f-soundness of $(\mathcal{P},\mathcal{V})$. Assume that for some $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$, $y \in \{0,1\}^*$, $z \in \{0,1\}^*$ and hardness factor $h \in \mathcal{HS}_\lambda$ there exists a PPT prover $\mathcal{P}^*$ and a non-negligible function $s(\cdot)$ s.t

$$\Pr[output_{\mathcal{V}} \leftarrow \langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h) : output_{\mathcal{V}} = \text{accept}] \geq s(\lambda).$$

Since the PPT prover $\mathcal{P}^*$ runs in $o\big(\lambda^{\log\lambda}\big)$ time and by assumption (i) of the statement of the theorem, we have that for some negiligble function $\delta(\cdot)$

$$\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); output_{\mathcal{V}} \leftarrow \langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h) : (output_{\mathcal{V}} = \text{accept}\big) \\ \wedge\, \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x,z,h)) \leq f(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))] \geq s(\lambda) - \delta(\lambda).$$

Since $s(\lambda) - \delta(\lambda)$ is a non-negligible function, by the f-soundness of $(\mathcal{P},\mathcal{V})$, the algorithm $\mathcal{K}$, given oracle access to $\mathcal{P}^*$, returns a witness for x with some non-negligible probability.

Straight-line $\lambda^{\mathrm{poly}(\log\lambda)}$-statistical simulatability. Let $\mathcal{V}^*$ be an arbitrary verifier. We construct a simulator $\mathcal{S}$ that runs in $\lambda^{\mathrm{poly}(\log\lambda)}$ time, such that the distributions

$$\{view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\}_{x\in\mathcal{L},\ w\in R_{\mathcal{L}}(x), z\in\{0,1\}^*, h\in\mathcal{HS}_\lambda} \text{ and} \\ \{view_{\mathcal{V}^*} \leftarrow \langle S \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\}_{x\in\mathcal{L}, z\in\{0,1\}^*, h\in\mathcal{HS}_\lambda}$$

are statistically indistinguishable. Namely, $\mathcal{S}$ encompasses the prover $\mathcal{P}$ and the puzzle solving algorithm Solve and emulates the PoW mode of $(\mathcal{P},\mathcal{V})$. By assumption (ii) in the statement of the theorem, $\mathcal{P}$ runs in polynomial time and makes oracle calls to Solve with worst case complexity $\lambda^{\mathrm{poly}(\log\lambda)}$. Since the complexity class $\lambda^{\mathrm{poly}(\log\lambda)}$ is closed under polynomial composition, the running time of $\mathcal{S}$ is bounded by $p(\lambda)\cdot\lambda^{\mathrm{poly}(\log\lambda)} = \lambda^{\mathrm{poly}(\log\lambda)}$, where $p(\cdot)$ is some polynomial. By the construction of $\mathcal{S}$, the distributions

$$\{view_{\mathcal{V}^*} \leftarrow \langle S \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\}_{x\in\mathcal{L}, z\in\{0,1\}^*, h\in\mathcal{HS}_\lambda} \text{ and} \\ \Big\{view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}^{\mathsf{Solve}(1^\lambda,h,\cdot)} \leftrightarrow \mathcal{V}^*\rangle(x,z,h)\Big\}_{x\in\mathcal{L}, z\in\{0,1\}^*, h\in\mathcal{HS}_\lambda} \equiv \mathbf{D}^{\mathcal{V}^*}_{PoW}$$

are identical. Thus, the straight-line $\lambda^{\mathrm{poly}(\log\lambda)}$-statistical simulatability follows from the statistical indistiguishability of $(\mathcal{P},\mathcal{V})$.

By applying the concurrent self-composition Lemma 2, we conclude that $(\mathcal{P},\mathcal{V})$ is a 3-move straight-line concurrent statisticallly $\lambda^{\mathrm{poly}(\log\lambda)}$-simulatable argument of knowledge for language $\mathcal{L}$.

□

Remark. As shown in Sec. 3.5, dense puzzles can be constructed by combining UOWHFs and any oneway functions. Moreover, one can build UOWHFs from any oneway functions [Rom90, HHR+10]. Therefore, assume there exists an oneway function against all sub-exponential running time adversaries, we have a 3-move concurrent argument of knowledge. As a result, we can improve the round complexity of the Pass's original concurrent ZK protocol with similar assumptions.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer Berlin Heidelberg, 2000.

[ADVW13] Shweta Agrawal, Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. On continual leakage of discrete log representations. In *ASIACRYPT*, 2013.

[AM11] Divesh Aggarwal and Ueli Maurer. The leakage-resilience limit of a computational problem is equal to its unpredictability entropy. In *ASIACRYPT*, 2011.

[Bac97] Adam Back. Hashcash. http://www.cypherspace.org/hashcash, 1997.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. ITCS, 2016. to appear.

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *In: Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[Bra94] S. Brands. An efficient off-line electronic cash system based on the representation problem. In *CWI Technical Report CS-R9323*, 1994.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, 2000.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer Berlin Heidelberg, 2002.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *STOC*, 1998.

[CMSW09] Liqun Chen, Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. Security notions and generic constructions for client puzzles. In *ASIACRYPT*, 2009.

[CPS+15] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved or composition of sigma-protocols. Cryptology ePrint Archive, Report 2015/810, 2015. `http://eprint.iacr.org/`.

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003)*, pages 426–444. Springer-Verlag, August 2003.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *FOCS*, 2000.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *STOC*, 1998.

[Dod05] Yevgeniy Dodis. On extractors, error-correction and hiding all partial information. In *ITW*, 2005.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *STOC*, 2005.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, 1990.

[Gen00] Rosario Gennaro. An improved pseudo-random generator based on discrete log. In *CRYPTO*, 2000.

[GJKY13] Juan A. Garay, David S. Johnson, Aggelos Kiayias, and Moti Yung. Resource-based corruptions and the combinatorics of hidden diversity. In *ITCS*, 2013.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

[GPR13] Steven D. Galbraith, John M. Pollard, and Raminder S. Ruprai. Computing discrete logarithms in an interval. *Math. Comput.*, 82(282), 2013.

[GTY07] K. Gopalakrishnan, Nicolas Thériault, and ChuiZhi Yao. Solving discrete logarithms from partial knowledge of the key. In *INDOCRYPT 2007*. 2007.

[HHR+10] Iftach Haitner, Thomas Holenstein, Omer Reingold, Salil P. Vadhan, and Hoeteck Wee. Universal one-way hash functions via inaccessible entropy. In *EUROCRYPT*, 2010.

[HILL93] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Construction of a pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1993.

[JB99] Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[KT13] Aggelos Kiayias and Qiang Tang. How to keep a secret: Leakage deterring public-key cryptosystems. In *CCS*, 2013.

[LPY12] Benoît Libert, Thomas Peters, and Moti Yung. Group signatures with almost-for-free revocation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 571–589. Springer Berlin Heidelberg, 2012.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO 2011*, volume 6841 of *LNCS*, page 39. Springer, 2011.

[Nak08] Satoshi Nakamoto. Report: `https://bitcoin.org/bitcoin.pdf`, 2008. Last accessed: Oct. 7, 2015.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, 1989.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, 2003.

[Pas04] Rafael Pass. Alternative variants of zero-knowledge proofs. In *Licentiate (Master's) Thesis, ISBN 91-7283-933-3*, 2004.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, 1990.

[RSW96] Ronald Rivest, Adi Shamir, and David Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *EUROCRYPT*, 1998.

[SY90] Alfredo De Santis and Moti Yung. On the design of provably secure cryptographic hash functions. In *EUROCRYPT 1990*, 1990.

[WJHF04] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS 04*, pages 246–256. ACM, 2004.

[Yun15] Aaram Yun. Generic hardness of the multiple discrete logarithm problem. In *EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 817–836. Springer Berlin Heidelberg, 2015.

A Building Blocks

A.1 3-move Special-sound HVZK (SS-HVZK) protocols

SS-HVZK protocols are a class of interactive proofs between a prover, P, and a verifier, V, who have a common input x and P proves in zero-knowledge that he knows a witness w such that $(x, w) \in R_{\mathcal{L}}$, where $R_{\mathcal{L}}$ is a witness relation for lagnuage $\mathcal{L} \in \mathcal{NP}$. In a 3-move public coin SS-HVZK protocol $\Pi = (\mathsf{P1}_\Pi, \mathsf{P2}_\Pi, \mathsf{Ver}_\Pi)$, (i) the prover first runs $(a, \phi) \leftarrow \mathsf{P1}_\Pi(w, x)$ and sends the first message a to the verifier; (ii) the verifier picks a challenge c uniformly at random from some challenge space $\mathcal{CS}_\Pi$ and sends the challenge c to the prover; (iii) the prover then runs $r \leftarrow \mathsf{P2}_\Pi(\phi, c)$ and sends the second message r to the verifier. The verifier accepts the proof if and only if $\mathsf{Ver}_\Pi(x, a, c, r) = 1$.

We say that the protocol satisfies the *computational (resp. statistical) honest-verifier zero knowledge (HVZK)* property, if there exists a polynomial-time simulator Sim_Π, which on input $x \in \mathcal{L}$ outputs an accepting transcript of the form (a, c, r) which distribution is computationally (resp. statistically) indistinguishable from an actual transcript generated by the interaction of the prover and the honest verifier. A stronger property named *special*

HVZK (sHVZK) requires that there exists a simulator that produces indistinguishable transscripts on input $x \in \mathcal{L}$ and a (possibly maliciously sampled) challenge c. When we allow the simulator to obtain an auxiliary input, we say that the protocol satisifies the *auxiliary input* (s)HVZK property. It is straightforward that statistical (s)HVZK is also statistical auxiliary input (s)HVZK.

Finally, we say that a 3-move public coin HVZK protocol is *special-sound* if there exists a polynomial-time knowledge extractor K_Π that on input x and any pair of accepting transcripts, $(a, c, r),\ (a, c', r')$ for x where $c \neq c'$, can output a witness w such that $(x, w) \in R_{\mathcal{L}}$.

A.2 The Schnorr Identification Scheme

An example of a 3-move SS-sHVZK protocol that we will use in our constructions, is the Schnorr identification scheme [Sch89]. This scheme is essentially a proof of knowledge of a discrete logarithm. Let G be a group of prime order q with generator g, and let $\mathbb{Z}_q$ denote the field of integers modulo q. Schnorr's identification scheme works as follows:

Prover$(q, g, x = g^w)$		Verifier(q, g, x)
$t \xleftarrow{\$} \mathbb{Z}_q,\ a = g^t$	$\xrightarrow{a}$	
	$\xleftarrow{c}$	$c \xleftarrow{\$} \mathbb{Z}_q$
$r = t + cw \bmod q$	$\xrightarrow{r}$	$g^r \stackrel{?}{=} ax^c$

B The Dense Puzzle Based PoWorK construction

B.1 Proof of Theorem 2

Proof: **Correctness and efficient samplability.** The correctness and efficient samplability is straightforward.

Completeness. We now show the completeness, namely the probability that $\Pr[\mathsf{puz} \leftarrow \{0,1\}^\lambda; \bot \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz})]$ is negligible in λ. We can view each $H(\cdot, y)$ oracle query as an independent random variable $A_j \in \{0, 1\}$, with $\mathbb{E}[A_j] = p = 2^{-h}$, where $A_j = 1$ if and only if $\mathsf{LSB}_h(H(x^*_j, y)) = \mathsf{LSB}_h(z)$. Let μ denote the expected value of $A = \sum_{j=1}^{2^{h+2\log\lambda}} A_j$, so we have $\mu = \mathbb{E}[\sum_{j=1}^{2^{h+2\log\lambda}} A_j] = \sum_{j=1}^{2^{h+2\log\lambda}} \mathbb{E}[A_j] = p \cdot 2^{h+2\log\lambda} = 2^{2\log\lambda} = \lambda^2$. Hence, let $\delta = 1 - \frac{1}{\lambda^2}$, by the generalized Chernoff bound, the probability Solve outputs $\bot$ for a given puz is

$$\Pr[A < 1] = \Pr[A < (1-\delta)\mu] \leq e^{\frac{-\delta^2\mu}{2}} = e^{-\frac{(1-1/\lambda^2)^2}{2}\cdot\lambda^2} = \mathsf{negl}(\lambda)\ .$$

Statistically indistinguishability. To show $\Delta[D_{s,\lambda,h}, D_{p,\lambda,h}] = \mathsf{negl}(\lambda)$, we first need to show that for all λ and $h \in \mathcal{HS}_\lambda$, the distribution of sampled puzzle, $P(\lambda, h) = \{\mathsf{puz} | (\mathsf{puz}, \mathsf{soln}) \leftarrow \mathsf{SampleSol}(1^\lambda, h)\}$ is statistically close to a uniform distribution over the $\mathcal{PS}_\lambda = \{0,1\}^\lambda$. Recall that puz consists of $H(x, y)$ and y, where x,y are chosen independently and uniformly at random. Analogous to the leftover hash lemma (LHL) [HILL93], we can show that $\Delta[P(\lambda, h), \mathbf{U}_\lambda] \leq 2^{-\lambda/4+1}$ as follows. We define the collision probability as $CP(H(x, y), y) = \Pr[(H(x, y), y), (H(x', y'), y')]$, where (x', y') is independent of and identically distributed to (x, y), i.e. $\mathbf{U}_\lambda \times \mathbf{U}_{\lambda/2}$. Since $\mathcal{O}$ is a random oracle, we have

$$\begin{aligned} CP(H(x, y), y) &= CP(y) \cdot (CP(x) + \Pr[H(x, y) = H(x', y) | x = x']) \\ &\leq 2^{-\lambda/2} \cdot (2^{-\lambda} + 2^{-\lambda/2}) = (1 + 2^{-\lambda/2}) \cdot 2^{-\lambda}\ . \end{aligned}$$

Meanwhile, we have

$$\begin{aligned} (\|(H(x, y), y) - \mathbf{U}_{\lambda/2} \times \mathbf{U}_{\lambda/2}\|_2)^2 &= CP(H(x, y), y) - CP(\mathbf{U}_{\lambda/2} \times \mathbf{U}_{\lambda/2}) \\ &\leq (1 + 2^{-\lambda/2}) \cdot 2^{-\lambda} - 2^{-\lambda} = 2^{-3\lambda/2}\ . \end{aligned}$$

Therefore,

$$\begin{aligned}\Delta[P(\lambda,h),\mathbf{U}_\lambda] &= \frac{1}{2}\|(H(x,y),y)-\mathbf{U}_{\lambda/2}\times\mathbf{U}_{\lambda/2}\|_1\\ &\leq 2^{\lambda/2-1}\cdot\|(H(x,y),y)-\mathbf{U}_{\lambda/2}\times\mathbf{U}_{\lambda/2}\|_2\\ &\leq 2^{\lambda/2-1}\cdot\sqrt{2^{-3\lambda/2}}=2^{-\lambda/4+1}\ .\end{aligned}$$

Secondly, due to that fact that $\mathcal{O}$ is a random oracle, the distribution of puz and soln are independent. Moreover, the Solve is probabilistic algorithm that tests the uniform randomly selected solution candidates, and thus it is obvious that $\mathsf{Solve}(1^\lambda,h,\mathsf{puz}))$ outputs a random soln from the solution set of puz, which is identically distributed to the solution soln in $(\mathsf{puz},\mathsf{soln})\leftarrow\mathsf{SampleSol}(1^\lambda,h)$. Therefore, we have the distance $\Delta[D_{s,\lambda,h},D_{p,\lambda,h}]=\mathsf{negl}(\lambda)$ as claimed.

g-hardness. First of all, although the adversary's auxiliary input is $z\in\{0,1\}^*$ can be arbitrarily long, the adversary is only able to read $O(g(2^{h+2\log\lambda}))\leq O(2^{\lambda/4})$ content of z under its running time limitation. Since $y\leftarrow\{0,1\}^{\lambda/2}$, the probability that the read content of z contains a $H(*,y)$ oracle query is at most $p_w=\frac{2^{\lambda/4}}{2^{\lambda/2}}=\mathsf{negl}(\lambda)$. In the rest case, we assume that each random oracle query takes 1 unit steps. Due to the property of random oracle, we expect $2^{\lambda-h}$ solutions in the solution space $\{0,1\}^\lambda$ for any given puzzle instance puz. The probability the adversary cannot find a solution within $2^{(h+2\log\lambda)/c}$ trials is

$$p_l=\frac{\binom{2^\lambda-2^{\lambda-h}}{2^{(h+2\log\lambda)/c}}}{\binom{2^\lambda}{2^{(h+2\log\lambda)/c}}}>(1-\frac{1}{2^{h-(h+2\log\lambda)/c}})^{2^{(h+2\log\lambda)/c}}\geq 1-2^{-(1-\frac{2}{c})h+\frac{4}{c}\log\lambda}\ .$$

Since $c>2$ and $h\geq\log^2\lambda$, we have the probability the adversary $\mathcal{A}$ can find a solution is

$$p_w+(1-p_w)(1-p_l)=\mathsf{negl}(\lambda).$$

$(\mathsf{id}(\cdot),k)$-amortization resistance. Let $\mathcal{A}$ be an adversary that runs in $O(k2^{(h+2\log\lambda)/c})$ steps and is given a set of k sampled puzzles $\mathsf{puz}_1,\ldots,\mathsf{puz}_k=(z_1,y_1),\ldots,(z_k,y_k)$. By the construction of the algorithm SampleSol, we have that the probability that all k values $y_1,\ldots,y_k$ are distinct is

$$\begin{aligned}p_d&=1\cdot(1-2^{-\lambda/2})\cdots(1-(k-1)2^{-\lambda/2})>(1-k2^{-\lambda/2})^k\geq 1-k^2 2^{-\lambda/2}\geq\\ &\geq 1-(2^{\lambda/8})^2\cdot 2^{-\lambda/2}=1-2^{-\lambda/4}=1-\mathsf{negl}(\lambda).\end{aligned}$$

Assume that k values $y_1,\ldots,y_k$ are distinct. As in the proof of g-hardness, since $\mathcal{A}$ runs in $O(k2^{(h+2\log\lambda)/c})=O(2^{(\lambda/8+h+2\log\lambda)/c})=O(2^{\lambda/4})$, for every $i\in[k]$, the probability that $\mathcal{A}$ reads an oracle query $H(\cdot,y_i)$ from the auxiliary tape is $p_i\leq 2^{-\lambda/4}$. By the union bound, the probability that $\mathcal{A}$ reads any oracle query $H(\cdot,y_1)\ldots,H(\cdot,y_k)$ from the auxiliary tape is $p_w\leq\prod_{i=1}^k p_i\leq k2^{-\lambda/4}\leq 2^{-\lambda/8}\cdot 2^{-\lambda/4}\leq 2^{-\lambda/8}=\mathsf{negl}(\lambda)$.

Let $q_1,\ldots,q_k$ be the number of oracle queries $H(\cdot,y_1),\ldots,H(\cdot,y_k)$ that $\mathcal{A}$ makes. By the restriction on the running time of $\mathcal{A}$, we have that $\sum_{i=1}^k q_i\leq k2^{(h+2\log\lambda)/c}$. By an averaging argument, there is an $i^*\in[k]$ such that $\mathcal{A}$ makes at most $2^{(h+2\log\lambda)/c}$ oracle queries $H(\cdot,y_{i^*})$. Due to the property of random oracle, we expect $2^{\lambda-h}$ solutions in the solution space $\{0,1\}^\lambda$ for puz_{i^*}. As previously, the probability that $\mathcal{A}$ cannot find a solution of puz_{i^*} within $2^{(h+2\log\lambda)/c}$ trials is more than $1-2^{-(1-\frac{2}{c})h+\frac{4}{c}\log\lambda}$. Since $c>2$ and $h\geq\log^2\lambda$, the probability that $\mathcal{A}$ can find a solution for all $\mathsf{puz}_1,\ldots,\mathsf{puz}_k$ is $\mathsf{negl}(\lambda)$. □

B.2 Proof of Theorem 3

Proof: Let $\hat{F}_i(\cdot):=(F_i(\cdot),i)$. If $H_\infty(x)=t\geq m(\lambda)+\lambda+2\log(1/\epsilon)+1$, by Lemma 1, we have $\Delta[(s,\hat{F}_i\circ H_s(x)),(s,\hat{F}_i(U_{\ell(\lambda)}))]\leq\epsilon$. In addition, i is drawn uniformly from $\{0,1\}^\lambda$, and F_i is a regular function; hence $F_i(\mathbf{U}_{\ell(\lambda)})\equiv\mathbf{U}_{m(\lambda)}$, and thus $\hat{F}_i\circ H_s(x)$ is statistically indistinguishable from $\mathbf{U}_{m(\lambda)}\times\mathbf{U}_\lambda$. Therefore, weconclude that $\mathsf{Ext}_\lambda(x,(i,s)) = (F_i \circ H_s(x), i)$ is a (t,ϵ)-strong extractor. In terms of the TCR property, we show that if there exists an adversary $\mathcal{A}$ who can break the TCR of Ext_λ, then we can build an adversary $\mathcal{B}$ who can break the TCR of $\mathcal{F}_\lambda$ as follows. $\mathcal{B}$ is playing the UOWHF TCR game, meanwhile $\mathcal{B}$ interacts with $\mathcal{A}$ as the challenger in the strong extractor TCR game. Up on $\mathcal{A}$ outputs $x \in \{0,1\}^{\ell(\lambda)}$, then B randomly picks $s \in \{0,1\}^{2\cdot\ell(\lambda)}$ and outputs $\hat{x} := H_s(x) \in \{0,1\}^{\ell(\lambda)}$ to its challenger. Up on receiving $i \in \{0,1\}^\lambda$ from its challenger, $\mathcal{B}$ sends (i,s) to $\mathcal{A}$. Up on $\mathcal{A}$ outputs $x' \in \{0,1\}^{\ell(\lambda)}$, $\mathcal{B}$ outputs $\hat{x}' := H_s(x') \in \{0,1\}^{\ell(\lambda)}$. Since $H_s(\cdot)$ is a permutation, $x \neq x'$ implies $H_s(x) \neq H_s(x')$. Clearly, $\mathcal{B}$'s probability of breaking UOWHF TCR property is exactly equal to $\mathcal{A}$'s probability of breaking strong extractor TCR property. □

B.3 Proof of Theorem 4

Proof: The ϵ-density of $\psi^U_{\lambda_1,\lambda_2}$ follows directly from the underlying (H_{λ_1},ϵ)-strong extractors, and by Theorem 3,

$$\Delta[(\mathsf{Ext}_{\lambda_2}(\zeta_{\lambda_1}(\psi_{\lambda_1}(x)),(s_1,s_2)),s_2),(\mathbf{U}_{H_{\lambda_1}-2\log(1/\epsilon)-1},s_2)] \leq \epsilon \ .$$

We now show $\psi^U_{\lambda_1,\lambda_2}$ is oneway by reduction. Namely, if there exists an adversary $\mathcal{A}$ who can break the onewayness of $\psi^U_{\lambda_1,\lambda_2}$ then we can construct an adversary $\mathcal{B}$ who can either break the onewayness of ψ_{λ_1} or break the TCR of Ext_{λ_2}. During the reduction, $\mathcal{B}$ plays the ψ_{λ_1} onewayness game with the environment C_1 and the Ext_{λ_2} TCR game with the environment C_2 simultaneously. $\mathcal{B}$ receives $y = \psi_{\lambda_1}(x)$ for some $x \in \mathcal{X}_{\lambda_1}$ from C_1, and then $\mathcal{B}$ outputs $\zeta_{\lambda_1}(y)$ to C_2. Upon receiving $(s_1,s_2) \in \{0,1\}^{\lambda_2} \times \{0,1\}^{2\cdot\ell(\lambda_2)}$ from the environment C_2, $\mathcal{B}$ sends $\mathcal{A}$ the image $\psi^U_{\lambda_1,\lambda_2}(x,(s_1,s_2)) = (\mathsf{Ext}_{\lambda_2}(\zeta_{\lambda_1}(y),(s_1,s_2)),s_2)$. $\mathcal{A}$ will then output $(x',(s'_1,s'_2)) \in \mathcal{X}_{\lambda_1} \times \{0,1\}^{2\cdot\ell(\lambda_2)}$, and $\mathcal{B}$ halts if $(s_1,s_2) \neq (s'_1,s'_2)$, as $\mathcal{A}$ fails. Otherwise, if $\psi_{\lambda_1}(x') = y$, $\mathcal{B}$ sends x' to the environment C_1; else $\mathcal{B}$ sends $\zeta_{\lambda_1}(\psi_{\lambda_1}(x'))$ to the environment C_2. Since ζ_{λ_1} is injective, $\psi_{\lambda_1}(x') = y$ implies $\zeta_{\lambda_1}(\psi_{\lambda_1}(x')) = \zeta_{\lambda_1}(y)$; hence, if $\mathcal{A}$ wins, $\mathcal{B}$ can win either one of her games. □

B.4 Proof of Theorem 5

Proof: **Correctness and efficient samplability.** Correctness and efficient samplability is straightforward.

Statistically indistinguishability. We now show that the puzzle system is statistically indistinguishable. Recall that puz consists of $\mathsf{Ext}_\lambda(\psi_G(x+2^h\cdot y),(s_1,s_2)),s_2,y$, where s_1,s_2,y are chosen independently and uniformly at random. Hence (s_1,s_2,y) is identically distributed to $\mathbf{U}_\lambda \times \mathbf{U}_{2\lambda} \times \mathbf{U}_{\lambda/2}$. Since $H_\infty(x) = h \geq \log^4\lambda + \log^2\lambda + 1$ and ψ_G is a bijective function, by Theorem 4, the puz $=$ $(\mathsf{Ext}_\lambda(\psi_G(x+2^h\cdot y),(s_1,s_2)),s_2,y)$ is at most $\epsilon = 2^{-(\log^2\lambda-1)/2} = \mathsf{negl}(\lambda)$ far from $\mathbf{U}_{\lambda+\log^4\lambda} \times \mathbf{U}_{2\lambda} \times \mathbf{U}_{\lambda/2}$, where $(s_1,s_2) \leftarrow \{0,1\}^{3\lambda}$ and $y \leftarrow \{0,1\}^{\lambda/2}$ are drawn uniformly random and independent to x. On the other hand, as shown in the paragraph below, the puzzle system is complete. Notice that the solver is probabilistic, so $\mathsf{Solve}(1^\lambda,h,\mathsf{puz}))$ outputs a random soln from the solution set of puz, which is identically distributed to the solution soln in $(\mathsf{puz},\mathsf{soln}) \leftarrow \mathsf{SampleSol}(1^\lambda,h)$. Therefore, $\Delta[D_{s,\lambda,h},D_{p,\lambda,h}] = \mathsf{negl}(\lambda)$ as claimed.

Completeness. Since the puzzle instance is statistically indistinguishable from uniform random, with probability at most $\epsilon = 2^{-(\log^2\lambda-1)/2} = \mathsf{negl}(\lambda)$ a puzzle puz $\leftarrow \{0,1\}^h$ is unsolvable; otherwise, the Solve can be used to distinguish the puzzle instance from uniform random. It is easy to see that the solver's running time is 2^h.

g**-hardness.** In terms of g-hardness, the adversary is able to read at most $O(2^{\log^5\lambda})$ content of its auxiliary tape z within its running time, whereas $(s_1,s_2) \leftarrow \{0,1\}^{3\lambda}$ and $y \leftarrow \{0,1\}^{\lambda/2}$; therefore, the probability that z contains a $\mathsf{Ext}_\lambda(\psi_G(x^*+2^h\cdot y),(s_1,s_2))$ query for some x^* is negligible in λ. In the rest case, recall that we assume breaking the TCR property of strong extractor is always harder than solving the generic DLP. The best generic algorithm must take at least $\sqrt{2^{h+1}\varepsilon}$ steps to solve a hard generic DLP with probability ε. Therefore, given $2^{h/c}$, $c > 2$, the adversary can successfully solve the generic DLP with probability at most $\varepsilon = 2^{-(1-\frac{2}{c})h-1} = \mathsf{negl}(\lambda)$.

(τ, k)**-amortization resistance.** Define the set of k sampled puzzles as $\mathsf{puz}_1, \ldots, \mathsf{puz}_k = (z_1, y_1), \ldots, (z_k, y_k)$. By the construction of the algorithm SampleSol, we have that the probability that all $y_1, \ldots, y_k$ are distinct is

$$p_d = 1 \cdot (1 - 2^{-\lambda/2}) \cdots (1 - (k-1)2^{-\lambda/2}) > (1 - k2^{-\lambda/2})^k \geq 1 - k^2 \cdot 2^{-\lambda/2} = 1 - \mathsf{negl}(\lambda).$$

Assume that k values $y_1, \ldots, y_k$ are distinct. As shown in [Yun15], the probability that an adversary $\mathcal{A}$ can solve the k puzzles with less than $\Theta(\sqrt{k \cdot 2^h})$ group operations is negligible. Hence, there exists a constant $\alpha > 0$ such that, with $1 - \mathsf{negl}(\lambda)$ probability, we have $\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \{\mathsf{puz}_1\}_{i=1}^k) \geq \alpha \cdot (k \cdot 2^h)^{1/2}$. Let $\tau(x) = x$. When $c > 2$, $k = O(2^{\log^3 \lambda})$ and $h > \log^4 \lambda$, we have for sufficiently large $\lambda \in \mathbb{N}$:

$$\tau\Big(\sum_{i=1}^{k} g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\Big) = k \cdot 2^{h/c} < \alpha \cdot (k \cdot 2^h)^{1/2} \ .$$

Therefore, the probability $\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \{\mathsf{puz}_1\}_{i=1}^k) \leq \tau\big(\sum_{i=1}^k g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)$ is $\mathsf{negl}(\lambda)$. □

C The Lapidot-Shamir Based PoWorK Construction

In this section, we describe our second PoWorK construction which is less efficient than the dense-puzzle based construction but can be constructed from any arbitrary puzzle system[9]. We stress that this construction is not black-box on the puzzle verification algorithm and does not retain the public-coin aspect (since the verifier will be sending an actual puzzle in the second move) without any additional assumption about the puzzle system. In Section C.1, we provide a detailed description of the Lapidot-Shamir (LS) protocol and the properties it satifies. In Section C.2, we present a 3-move protocol that compiles any 3-move special sound and computationally auxiliary input special HVZK (sHVZK) protocol (like the LS protocol) into a PoWorK that, as we prove in Section C.3, is $\Theta(g)$-sound and computationally indistinguishable, where g is the hardness scaling function of the underlying puzzle system.

C.1 The Lapidot-Shamir SS -sHVZK protocol

We recap the 3-move Lapidot-Shamir (LS) special sound computational auxiliary input sHVZK protocol [LS90] in this section. The LS protocol is an SS-sHVZK protocol (see Appendix A.1) for Hamiltonian Cycle, and thus it can support any $\mathcal{NP}$ language. In the LS protocol, the prover only needs to know the size of the statement in order to produce the first move, while the actual statement is only needed for the third move. This property is crucial for our construction. In the following description, we run $\ell(\lambda)$ instances of the original LS protocol in a parallel. Denote $\mathsf{P1}_{LS}, \mathsf{P2}_{LS}, \mathsf{Ver}_{LS}$ as the first move prover, third move prover and the verification algorithm respectively. The common input of the prover and verifier is a graph G with N vertices, represented by its adjacency matrix. In addition, the prover takes a Hamiltonian cycle of G (denoted as C) as its private input.

- $\mathsf{P1}_{LS}(N)$: For $i \in \{1, 2, \ldots, \ell(\lambda)\}$, do:
  - Pick a random cycle R_i with N vertices.
  - Commit to every element of the adjacency matrix of R_i, denoted as $\mathsf{Com}(R_i)$, using a *statistically binding commitment scheme*.
- $\mathsf{P1}_{LS} \rightarrow \mathsf{Ver}_{LS}$: $\mathsf{Com}(R_1), \ldots, \mathsf{Com}(R_{\ell(\lambda)})$
- $\mathsf{P2}_{LS} \leftarrow \mathsf{Ver}_{LS}$: $c = c_1 \cdots c_{\ell(\lambda)} \leftarrow \{0,1\}^{\ell(\lambda)}$
- $\mathsf{P2}_{LS}(G, c)$: For $i \in \{1, 2, \ldots, \ell(\lambda)\}$, do:

[9] The authors are grateful to an anonymous reviewer for suggesting the possibility of using this approach for constructing PoWorKs.

- If $c_i = 0$, then define z_i as the openings of the entire committed adjacency matrix, $\mathsf{Com}(R_i)$.
- IF $c_i = 1$, then define z_i as (π_i, d_i), where π_i is a permutation from the vertices of R_i to the vertices of G and d_i is the openings of all adjacency matrix elements of R_i that correspond to non-edges of G.

- $\mathsf{P2}_{LS} \rightarrow \mathsf{Ver}_{LS}$: $z_1, \ldots, z_{\ell(\lambda)}$.
- $\mathsf{Ver}_{LS}\left(G, \{\mathsf{Com}(R_i)\}_{i\in[\ell(\lambda)]}, c, \{z_i\}_{i\in[\ell(\lambda)]}\right)$: return 1 if and only if for every $i \in \{1, 2, \ldots, \ell(\lambda)\}$:
  - if $c_i = 0$, all the openings of the commitments verify and the openings of $\mathsf{Com}(R_i)$ form indeed a random cycle.
  - if $c_i = 1$, the openings of all adjacency matrix elements of R_i that correspond to non-edges of G are 0 (i.e. R_i is a subgraph of G up to permutation).

Properties of the LS protocol.

- **Special soundness:** Given two accepting transcripts with $c \neq c'$, there exists a knowledge extractor that can output a Hamiltonian cycle of G. Indeed, if $c \neq c'$, then $\exists i \in [\ell(\lambda)]$ s.t. $c_i \neq c'_i$. Therefore, from the i-th instance, we obtain (i) the random cycle R_i when $c_i = 0$ and (ii) the permutation that maps R_i to the actual Hamiltonian cycle C of G when $c'_i = 1$.
- **Auxiliary input sHVZK:** There exists a simulator $\mathsf{Sim}_{LS} = (\mathsf{Sim1}_{LS}, \mathsf{Sim2}_{LS})$ s.t. for any challenge c, Sim_{LS} can simulate a transcript that is computationally indistinguishable from the real one. Observe that the LS protocol achieves this property, for any auxiliary input because in each execution, the prover sends a fresh commitment key in the first move. Therefore, the verifier has negligible probability of gaining significant information about the table of messages and corresponding commitments by reading a polynomial size part of the auxiliary input. Finally, the sHVZK is *computational*, as an unbounded algorithm may break the hiding property of the statistically binding scheme.
- **First move independence:** The selection of R and the commitments to the elements of its adjacency matrix are performed independently of G and C. We emphasize that $\mathsf{Sim1}_{LS}$ can simulate the first move without knowing the statement as well, namely it commits to a random cycle if $c_i = 0$; commits to a zero adjacency matrix if $c_i = 1$.

C.2 The Lapidot-Shamir Based Compiler.

The compiler is designed with black-box access to any 3-move special sound auxiliary input sHVZK protocol Π for some language $\mathcal{L} \in \mathcal{NP}$. W.l.o.g., the challenge sampling distribution of Π is uniform in the challenge space. The properties of the LS protocol imply that there exists such a protocol for every language in $\mathcal{NP}$. Let $\mathsf{P1}_\Pi, \mathsf{P2}_\Pi, \mathsf{Ver}_\Pi$, and Sim_Π be the first move prover, third move prover, verification algorithms, and simulator of Π, respectively. The challenge space of $(\mathcal{P}, \mathcal{V})$, Π and the LS protocol coincide and are set as $\{0,1\}^{\ell(\lambda,h)}$, where $\ell(\cdot,\cdot)$ is function that depends on λ and the hardness factor h, so that the size of the challenge space is superpolynomial in λ.

Let Sim_{LS} be the simulator of the aforementioned LS protocol. Here, we need to exploit the feature that Sim_{LS} can simulate the first move without knowing the statement, i.e. it commits to either a random cycle or a zero matrix depending on the challenge bit. Hence, we denote $\mathsf{Sim}_{LS} = (\mathsf{Sim1}_{LS}, \mathsf{Sim2}_{LS})$ such that $(a, \mathsf{st}) \leftarrow \mathsf{Sim1}_{LS}(c, N)$ and $r \leftarrow \mathsf{Sim2}_{LS}(G, c, \mathsf{st})$, where G is the statement of size N, c is the challenge and st is the simulator's state. For fixed security parameter λ and hardness factor λ, h, we define the language

$$\mathcal{L}_{\lambda,h} = \left\{t \in \mathcal{PS}_\lambda \mid \exists s \in \mathcal{HS}_\lambda : \mathsf{Verify}(1^\lambda, h, t, s) = \mathsf{true}\right\}.$$

We reduce $\mathcal{L}_{\lambda,h}$ to the Hamiltonian Cycle via the generic deterministic algorithms $\mathcal{G}$ and $\mathcal{C}$ that will encode a statement (puzzle) t and a witness (solution) $s \in R_{\mathcal{L}_{\lambda,h}}(t)$ to a graph G_t and a hamiltonian cycle H_s of G_t

respectively. Note that the size of G_t, $N_{\lambda,h}$ depends only on λ, h, which enables the application of the first move of LS protocol before receiving the puzzle statement at the second move of our construction.

The protocol $(\mathcal{P}, \mathcal{V})$ can be executed in either of the two following modes:

1. **Proof of Knowledge (PoK) mode.** $\mathcal{P}$ has a witness $w \in \mathcal{R}_{\mathcal{L}}(x)$ as private input. In order to prove knowledge of w to $\mathcal{V}$, in the first move, $\mathcal{P}$ follows the first move of Π and simulates the first move of the LS protocol by providing $\mathsf{Sim1}_{LS}$ with a random challenge c. The verifier responds with a challenge $\hat{c}$ and a sampled puzzle puz. Then, $\mathcal{P}$ executes the third move of Π by running $\mathsf{P2}_{\Pi}$ with the challenge $\tilde{c} = \hat{c} \oplus c$ and simulates the third move of the LS protocol.

2. **Proof of Work (PoW) mode.** $\mathcal{P}$ has no private input and convinces $\mathcal{V}$ by "working" for at least some expected amount of time. To achieve this, $\mathcal{P}$ simulates an execution of Π with a sampled challenge $\tilde{c}$ and follows the first move of the LS protocol. Then, it receives $(\hat{c}, \mathsf{puz})$ from $\mathcal{V}$ as before and runs the puzzle solver to obtain a solution soln of puz, which encodes as a cycle C_{soln} of the graph G_{puz}. Finally, it proves the knowledge of soln via reduction to the third move of the LS protocol with challenge $c = \hat{c}^{-1} \oplus \tilde{c}$.

As in the previous construction, the verification mechanism, must be the same for both modes. Namely, the verifier computes the encoding G_{puz} of the challenge puzzle puz and checks that: (i) the relation $\tilde{c} = \hat{c} \oplus c$ holds, (ii) the Π-protocol's transcript is accepting and (iii) the LS protocol's transcript for statement G_{puz} is accepting. The protocol $(\mathcal{P}, \mathcal{V})$ is presented in detail in Figure 5.

Statement: $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$.
Prover's private input: $w \in R_{\mathcal{L}}(x)$.

$\mathcal{P}$: • pick $c \xleftarrow{\$} \{0,1\}^{\ell(\lambda,h)}$;
• $(a, \mathsf{st}) \leftarrow \mathsf{Sim1}_{LS}(c, N_{\lambda,h})$;
• $(\tilde{a}, \phi_{\Pi}) \leftarrow \mathsf{P1}_{\Pi}(x, w)$;

$\mathcal{P} \rightarrow \mathcal{V}$: $a, \tilde{a}$.

$\mathcal{V}$: • pick $\hat{c} \xleftarrow{\$} \{0,1\}^{\ell(\lambda,h)}$;
• $\mathsf{puz} \leftarrow \mathsf{Sample}(1^{\lambda}, h)$;

$\mathcal{P} \leftarrow \mathcal{V}$: $\hat{c}, \mathsf{puz}$.

$\mathcal{P}$: • $\tilde{c} = \hat{c} \oplus c$;
• $G_{\mathsf{puz}} \leftarrow \mathcal{G}(1^{\lambda}, h, \mathsf{puz})$;
• $\tilde{r} \leftarrow \mathsf{P2}_{\Pi}(\phi_{\Pi}, \tilde{c}, x, w)$;
• $r \leftarrow \mathsf{Sim2}_{LS}(G_{\mathsf{puz}}, c, \mathsf{st})$;

$\mathcal{P} \rightarrow \mathcal{V}$: $c, \tilde{c}, r, \tilde{r}$.

Verification:
1. $\tilde{c} = \hat{c} \oplus c$.
2. $\mathsf{Ver}_{\Pi}(x, \tilde{a}, \tilde{c}, \tilde{r}) = 1$.
3. $\mathsf{Ver}_{LS}(1^{\lambda}, \mathcal{G}(1^{\lambda}, h, \mathsf{puz}), a, c, r) = 1$.

(a) Knowing the witness (PoK)

Statement: $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$.
Prover's private input: $-$

$\mathcal{P}$: • pick $\tilde{c} \xleftarrow{\$} \{0,1\}^{\ell(\lambda,h)}$;
• $(a, \phi_{LS}) \leftarrow \mathsf{P1}_{LS}(N_{\lambda,h})$;
• $(\tilde{a}, \tilde{c}, \tilde{r}) \leftarrow \mathsf{Sim}_{\Pi}(\tilde{c})$;

$\mathcal{P} \rightarrow \mathcal{V}$: $a, \tilde{a}$.

$\mathcal{V}$: • pick $\hat{c} \xleftarrow{\$} \{0,1\}^{\ell(\lambda,h)}$;
• $\mathsf{puz} \leftarrow \mathsf{Sample}(1^{\lambda}, h)$;

$\mathcal{P} \leftarrow \mathcal{V}$: $\hat{c}, \mathsf{puz}$.

$\mathcal{P}$: • $c = \hat{c}^{-1} \oplus \tilde{c}$;
• $\mathsf{soln} \leftarrow \mathsf{Solve}(1^{\lambda}, h, \mathsf{puz})$;
• $G_{\mathsf{puz}} \leftarrow \mathcal{G}(1^{\lambda}, h, \mathsf{puz})$;
• $C_{\mathsf{soln}} \leftarrow \mathcal{C}(1^{\lambda}, h, \mathsf{soln})$;
• $r \leftarrow \mathsf{P2}_{LS}(\phi_{LS}, c, G_{\mathsf{puz}}, C_{\mathsf{soln}})$;

$\mathcal{P} \rightarrow \mathcal{V}$: $c, \tilde{c}, r, \tilde{r}$.

Verification:
1. $\tilde{c} = \hat{c} \oplus c$.
2. $\mathsf{Ver}_{\Pi}(x, \tilde{a}, \tilde{c}, \tilde{r}) = 1$.
3. $\mathsf{Ver}_{LS}(1^{\lambda}, \mathcal{G}(1^{\lambda}, h, \mathsf{puz}), a, c, r) = 1$.

(b) Doing work (PoW)

Figure 5: The LS PoWorK construction for fixed security parameter λ and hardness factor $h \in \mathcal{HS}_{\lambda}$, given a 3-move-SS-sHVZK protocol Π for language $\mathcal{L}$, an LS protocol and a puzzle system PuzSys; the challenge space of $(\mathcal{P}, \mathcal{V})$, Π and the LS protocol coincide and are set as $\{0,1\}^{\ell(\lambda,h)}$; ϕ_{Π}, ϕ_{LS} and st are states of the prover of Π, the prover of the LS protocol and the simulator of the LS protocol, respectively.

C.3 Security of the Lapidot-Shamir PoWorK Construction

We denote by $\mathcal{C}^{\mathsf{Inv}}$: Hamiltonian Cycle $\longrightarrow \mathcal{SS}_\lambda$ the inverse of the cycle encoding algorithm $\mathcal{C}$ that decodes an encoded witness (solution of a puzzle-statement). The algorithm $\mathcal{C}^{\mathsf{Inv}}$ is deterministic and runs in polynomial time. In addition, we denote by $\mathcal{K}_{LS}$ the PPT witness extractor of the LS protocol. Like in Section 3 (Figure 2 Assumption (**C**)), we assume that the running time of Solve dominates the running time of all algorithms associated with the construction.

Theorem 8 *Let $\mathcal{L}$ be a language in $\mathcal{NP}$ and let $\Pi = (\mathsf{P1}_\Pi, \mathsf{P2}_\Pi, \mathsf{Ver}_\Pi)$ be a special sound 3-move computational auxiliary input sHVZK protocol for $\mathcal{L}$, where the challenge sampling distribution is uniform. Let $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$ be a puzzle system that satisfies g-hardness for some function g. Define $(\mathcal{P}, \mathcal{V})$ as the protocol described in Figure 5 when built upon Π and PuzSys. Assume that there exists a constant $\kappa < 1$ and a negligible function $\epsilon(\cdot)$ s.t. for every hardness factor $h \in \mathcal{HS}_\lambda$:*

$$\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h) : \kappa \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) > \\ > 2 \cdot \mathsf{Steps}_{\mathsf{ChSampler}}(1^\lambda, h) + \mathsf{Steps}_{\mathcal{K}_{LS}}(\mathsf{tr}_{\lambda,h}, \mathsf{tr}'_{\lambda,h}) + \mathsf{Steps}_{\mathcal{C}^{\mathsf{Inv}}}(1^\lambda, h)] \geq 1 - \epsilon(\lambda),$$

*where $\mathcal{C}^{\mathsf{Inv}}$ is the inverse of the cycle encoding algorithm $\mathcal{C}$ and $\mathcal{K}_{LS}$ is the witness extractor for the LS protocol on input two protocol transcripts $\mathsf{tr}_{\lambda,h}, \mathsf{tr}'_{\lambda,h}$. Then, $(\mathcal{P}, \mathcal{V})$ is an $\big((1-\kappa)/2\big) \cdot g$-*sound PoWorK *for $\mathcal{L}$ and* PuzSys *with computational indistinguishability.*

Proof:

• **Completeness.** By the completeness of PuzSys, we have that with overwhelming probabiltiy, soln, as computed in the PoW mode of $(\mathcal{P}, \mathcal{V})$, is a solution of the sampled puz, i.e. $\mathsf{soln} \in R_{\mathcal{L}_{\lambda,h}}(\mathsf{puz})$. This implies that with overwhelming probability, the reduction of $\mathcal{L}_{\lambda,h}$ to the Hamiltonian Cycle maps $(\mathsf{puz}, \mathsf{soln})$ to a graph G_{puz} that has C_{soln} as hamiltonian cycle. Moreover, the completeness of the LS and Π protocols implies that the simulated transcripts in both PoK and PoW mode of $(\mathcal{P}, \mathcal{V})$ must be accepting with overwhelming probability. Therefore, verification will be accepting with overwhelming probability for any honest execution of $(\mathcal{P}, \mathcal{V})$.

• $\big((1-\kappa)/2\big) \cdot g$**-Soundness.** First, we make use of the special soundness PPT extractor K_Π of Π to construct a PPT knowledge extractor $\mathcal{K}$ that on input (x, y, z) and given oracle access to an arbitrary prover $\hat{\mathcal{P}}$, executes the following steps:

1. $\mathcal{K}$ samples (honestly) a puzzle, puz and two challenges, $\hat{c}_1, \hat{c}_2$.
2. Using standard rewinding, $\mathcal{K}(x, y, z)$ interacts with $\hat{\mathcal{P}}(y)$ by submitting the challenges $(\hat{c}_1, \mathsf{puz}), (\hat{c}_2, \mathsf{puz})$. It receives two protocol transcripts from $\hat{\mathcal{P}}$, $\langle(a, \tilde{a}), (\hat{c}_1, \mathsf{puz}), (c_1, \tilde{c}_1, r_1, \tilde{r}_1)\rangle$ and $\langle(a, \tilde{a}), (\hat{c}_2, \mathsf{puz}), (c_2, \tilde{c}_2, r_2, \tilde{r}_2)\rangle$.
3. It runs the witness extractor $\mathcal{K}_\Pi$ of the protocol Π on input $(x, \langle\tilde{a}, \tilde{c}_1, \tilde{r}_1\rangle, \langle\tilde{a}, \tilde{c}_2, \tilde{r}_2\rangle)$.
4. It returns the output of $\mathcal{K}_\Pi$.

Assume that for some $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$, $y \in \{0,1\}^*, z \in \{0,1\}^*$, $h \in \mathcal{HS}_\lambda$, there exists a prover $\mathcal{P}^*$ and a non-negligible function $s(\cdot)$ s.t

$$\Pr[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h); output_\mathcal{V} \leftarrow \langle\mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z) : (output_\mathcal{V} = \text{accept}\big) \\ \wedge \mathsf{Steps}_{\mathcal{P}^*}(\langle\mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z)) \leq \big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))] \geq s(\lambda).$$

We will prove that $\big((1-\kappa)/2\big) \cdot g$-soundness of $(\mathcal{P}, \mathcal{V})$ is satisifed, unless we can use $\mathcal{P}^*$ to construct an algorithm $\mathcal{W}$ that breaks the g-hardness of PuzSys.

Let $Y \subseteq \mathcal{P}_\lambda$ be the set of puzzles, such that when the challenge $(\hat{c}, \mathsf{puz})$ of $\mathcal{V}$ satisfies $\mathsf{puz} \in Y$, then

$$\Pr[output_\mathcal{V} \leftarrow \langle\mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z) : (output_\mathcal{V} = \text{accept}\big) \\ \wedge \mathsf{Steps}_{\mathcal{P}^*}(\langle\mathcal{P}^*(y) \leftrightarrow \mathcal{V}\rangle(x, z)) \leq \big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))] \geq s(\lambda)/2.$$

By the assumption for $\mathcal{P}^*$ and a standard counting argument, we have that $\Pr[\mathsf{puz} \in Y] \geq s(\lambda)/2$.

Suppose that we perform rewinding on $\mathcal{P}^*$, by fixing the same puzzle puz in the verifier's challenge. Let $\langle (a, \tilde{a}), (\hat{c}_1, \mathsf{puz}), (c_1, \tilde{c}_1, r_1, \tilde{r}_1) \rangle$ and $\langle (a, \tilde{a}), (\hat{c}_2, \mathsf{puz}), (c_2, \tilde{c}_2, r_2, \tilde{r}_2) \rangle$ be the two protocol transcripts. If $\mathsf{puz} \in Y$, then by the splitting Lemma, both transcripts are accepting with at least $(s(\lambda)/4)^2 = s(\lambda)^2/16$ probability.

The challenge space of $(\mathcal{P}, \mathcal{V})$ (i.e. the challenge space of Π) has superpolynomial size, so the probability that the two uniformly sampled challenges $\hat{c}_1, \hat{c}_2$ are equal is no more than some negligible function $\delta(\lambda)$. If the verification for both transcripts is accepting and $\hat{c}_1 \neq \hat{c}_2$, then it holds that

$$(\tilde{c}_1 = \hat{c}_1 \oplus c_1) \wedge (\tilde{c}_2 = \hat{c}_2 \oplus c_2) \wedge (\hat{c}_1 \neq \hat{c}_2) \Rightarrow (c_1 \neq c_2) \vee (\tilde{c}_1 \neq \tilde{c}_2). \tag{5}$$

Let D be the event that $\mathcal{P}^*$, when rewinded as above, outputs two accepting transcripts and $\hat{c}_1 \neq \hat{c}_2$, $c_1 \neq c_2$ occur. Let $\tilde{D}$ be the event that $\mathcal{P}^*$, when rewinded as above, outputs two accepting transcripts and $\hat{c}_1 \neq \hat{c}_2, \tilde{c}_1 \neq \tilde{c}_2$ occur. By the assumption for $\mathcal{P}^*$ and eq. (5), we have that if $\mathsf{puz} \in Y$, then one of the probabilities $\Pr[D|\mathsf{puz} \in Y]$, $\Pr[\tilde{D}|\mathsf{puz} \in Y]$ must be at least $s(\lambda)^2/32 - \delta(\lambda)$. We analyze both cases:

I. $\Pr[D|\mathsf{puz} \in Y] \geq s(\lambda)^2/32 - \delta(\lambda)$ **holds.** In this case, we can construct an algorithm $\mathcal{W}$ that breaks the g-hardness of PuzSys. The input that $\mathcal{W}$ receives is $(1^\lambda, (x, y, z), h, \mathsf{puz})$, where (x, y, z) is the auxiliary input and puz is sampled from $\mathsf{Sample}(1^\lambda, h)$. Then, $\mathcal{W}$ works as follows:

1. It invokes $\mathcal{P}^*$ for statement x, private input y and auxiliary input z.
2. Using standard rewinding, $\mathcal{W}$ interacts with $\mathcal{P}^*(y)$ with two challenges $(\hat{c}_1, \mathsf{puz})$, $(\hat{c}_2, \mathsf{puz})$, where $\hat{c}_1, \hat{c}_2$ are uniformly sampled from $\{0,1\}^{\ell(\lambda,h)}$. It receives two transcripts, $\langle (a, \tilde{a}), (\hat{c}_1, \mathsf{puz}), (c_1, \tilde{c}_1, r_1, \tilde{r}_1) \rangle$ and $\langle (a, \tilde{a}), (\hat{c}_2, \mathsf{puz}), (c_2, \tilde{c}_2, r_2, \tilde{r}_2) \rangle$.
3. It runs the witness extractor $\mathcal{K}_{LS}$ of the LS protocol on input $(G_{\mathsf{puz}}, \langle a_1, c_1, r_1 \rangle, \langle a_2, c_2, r_2 \rangle)$. It receives an output C from $\mathcal{K}_{LS}$.
4. It runs the inverse of the cycle encoding algorithm $\mathcal{C}$, $\mathcal{C}^{\mathsf{Inv}}$: Hamiltonian Cycle $\longrightarrow \mathcal{SS}_\lambda$ on input C and receives a value soln.
5. It returns soln.

By definition of Y and D and the special soundness property of the LS protocol, we have that if $\mathsf{puz} \in Y$ and D occurs, then $\mathcal{W}$'s output soln is verified, i.e. $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{true}$. By the previous analysis, the probability that the latter happens is at least

$$\Pr[(\mathsf{puz} \in Y) \wedge D] = \Pr[\mathsf{puz} \in Y] \cdot \Pr[D|\mathsf{puz} \in Y] \geq \big(1/2s(\lambda)\big) \cdot \big(s(\lambda)^2/32 - \delta(\lambda)\big) \geq s(\lambda)^3/64 - \delta(\lambda).$$

By the assumption in the statement of the theorem and the assumption for $\mathcal{P}^*$, there is a constant $\kappa < 1$ s.t. the probability that $\mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathsf{soln}) = \mathsf{true}$ and the running time of $\mathcal{W}$ in number of steps is bounded by

$$\begin{aligned}
&\mathsf{Steps}_{\mathcal{W}}(x, y, z, 1^\lambda, h, \mathsf{puz}) \leq \\
&\quad \leq 2 \cdot \mathsf{Steps}_{\mathcal{P}^*}(\langle \mathcal{P}^*(y) \leftrightarrow \mathcal{V} \rangle (x, z, 1^\lambda, h)) + 2 \cdot (\mathsf{Steps}_{\mathsf{Sample}}(1^\lambda, h, \mathsf{puz})) + \\
&\quad\ + \mathsf{Steps}_{\mathcal{K}_{LS}}((G_{\mathsf{puz}}, a_1, c_1, r_1), (G_{\mathsf{puz}}, a_2, c_2, r_2)) + \mathsf{Steps}_{\mathcal{C}^{\mathsf{Inv}}}(1^\lambda, h, C) \leq \\
&\quad \leq 2\big((1-\kappa)/2\big) \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) + \kappa \cdot g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) = \\
&\quad = g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))
\end{aligned}$$

is at least $s(\lambda)^3/64 - \delta(\lambda) - \epsilon(\lambda)$ which is a non-negligible function. Therefore, for auxiliary tape (x, y, z) and hardness factor h, $\mathcal{W}$ breaks the g-hardness of PuzSys, which contradicts to the security of the said puzzle system.

II. $\Pr[\tilde{D}|\mathsf{puz} \in Y] \geq s(\lambda)^2/32 - \delta(\lambda)$ **holds.** In this case, we have that $\tilde{c}_1 \neq \tilde{c}_2$. By the special soundness property of Π, when the knowledge extractor $\mathcal{K}$ invokes $\mathcal{K}_\Pi$ on two accepting transcripts with two different challenges, it will return a witness for x. Define $q(\lambda) = s(\lambda)^3/64 - \delta(\lambda)$. The probability that $\mathcal{K}$ extracts a witness is at least

$$\Pr[\tilde{D}] = \Pr[\mathsf{puz} \in Y] \cdot \Pr[\tilde{D}|\mathsf{puz} \in Y] \geq q(\lambda).$$

Thus, we conclude that our protocol is $\big(\big((1-\kappa)/2\big) \cdot g\big)$-sound.

• **Computational indistinguishability**. We will show that $(\mathcal{P}, \mathcal{V})$ is computationally indistinguishable, if Π and the LS protocol achieve HVZK for any auxiliary input $z \in \{0,1\}^*$. To do this, we will make use of a "hybrid" protocol $(\tilde{\mathcal{P}}, \mathcal{V})$ where the prover $\tilde{\mathcal{P}}$ follows both underlying protocols, Π and LS, of $(\mathcal{P}, \mathcal{V})$ and the verifier $\mathcal{V}$ behaves as before. For fixed λ, h, the description of $(\tilde{\mathcal{P}}, \mathcal{V})$ is as follows:

Statement: $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$.
Prover's private input: $w \in R_\mathcal{L}(x)$.

First move: $\tilde{\mathcal{P}}$ samples a random challenge c and executes $(\tilde{a}, \phi_\Pi) \leftarrow \mathsf{P1}_\Pi(x, w); (a, \phi_{LS}) \leftarrow \mathsf{P1}_{LS}(N_{\lambda,h})$. It sends $a, \tilde{a}$ to $\mathcal{V}$.

Second move: $\mathcal{V}$ samples a pair c, puz and sends it to $\tilde{\mathcal{P}}$.

Third move: $\tilde{\mathcal{P}}$ computes $\tilde{c} = \hat{c} \oplus c$. It runs $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$ and receives a solution soln. Then, it encodes puz and soln as G_puz and C_soln respectively. Finally, it executes $\tilde{r} \leftarrow \mathsf{P2}_\Pi(\phi_\Pi, \tilde{c}, x, w)$ and $r \leftarrow \mathsf{P2}_{LS}(\phi_{LS}, c, G_\mathsf{puz}, C_\mathsf{soln})$ and sends $r, \tilde{r}$ to $\mathcal{V}$.

Verification: as in the $(\mathcal{P}, \mathcal{V})$ protocol.

Let $\mathcal{V}^*$ be a PPT verifier. W.l.o.g., we assume that $\mathcal{V}^*$ returns a single bit. Let $\tilde{\mathbf{D}}^{\mathcal{V}^*}$ be the distribution determined by the view of $\mathcal{V}^*$ when interacting with $\mathcal{P}$. We will show that the distributions $\mathbf{D}^{\mathcal{V}^*}_{PoK}$, $\mathbf{D}^{\mathcal{V}^*}_{PoW}$ determined by the view of $\mathcal{V}^*$ when interacting with $\mathcal{P}$ in the PoK and PoW mode of $(\mathcal{P}, \mathcal{V}^*)$ are computationally indistinguishable because (I) $\mathbf{D}^{\mathcal{V}^*}_{PoK}, \tilde{\mathbf{D}}$ are computationally indistinguishable and (II) $\mathbf{D}^{\mathcal{V}^*}_{PoW}, \tilde{\mathbf{D}}^{\mathcal{V}^*}$ are computationally indistinguishable.

I. $\mathbf{D}^{\mathcal{V}^*}_{PoK}, \tilde{\mathbf{D}}^{\mathcal{V}^*}$ **are computationally indistinguishable**. We observe that in the PoK mode of $(\mathcal{P}, \mathcal{V}^*)$ and $(\tilde{\mathcal{P}}, \mathcal{V}^*)$ the values $c, \tilde{a}, \hat{c}, \mathsf{puz}, \tilde{c}, \tilde{r}$ are identically distributed. Therefore, for every statement $x \in \mathcal{L}$ and auxiliary input $z \in \{0,1\}^*$,

$$\begin{aligned}
&\Big|\Pr_{(c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoK}}[\mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1] - \Pr_{(c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) \leftarrow \tilde{\mathbf{D}}^{\mathcal{V}^*}}[\mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1]\Big| = \\
&= \sum_{(c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r})} \Pr[c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r}] \cdot \\
&\cdot \Big(\Pr[(a,\mathsf{st}) \leftarrow \mathsf{Sim1}_{LS}(c, N_{\lambda,h}); r \leftarrow \mathsf{Sim2}_{LS}(G_\mathsf{puz}, c, \mathsf{st}); \\
&\qquad \mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1 \mid c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r}] - \\
&- \Pr[(a,\phi_{LS}) \leftarrow \mathsf{P1}_{LS}(N_{\lambda,h}); r \leftarrow \mathsf{P2}_{LS}(\phi_{LS}, c, G_\mathsf{puz}, C_\mathsf{soln}); \\
&\qquad \mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1 \mid c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r}]\Big).
\end{aligned} \tag{6}$$

By the computational auxiliary input sHVZK property of the LS protocol, we have that for any challenge c and auxiliary input $(z, \tilde{a}, \hat{c}, \mathsf{puz}, \tilde{c}, \tilde{r})$, the PPT verifier $\mathcal{V}^*$ cannot distinguish between the actual and the simulated view of the LS protocol. Therefore, by eq. (6), we have that for some negligible function $\delta(\cdot)$,

$$\begin{aligned}
&\Big|\Pr_{(c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) \leftarrow \mathbf{D}^{\mathcal{V}^*}_{PoK}}[\mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1] - \Pr_{(c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) \leftarrow \tilde{\mathbf{D}}^{\mathcal{V}^*}}[\mathcal{V}^*(x,z,c,a,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},r,\tilde{r}) = 1]\Big| \leq \\
&\leq \sum_{(c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r})} \Pr[c,\tilde{a},\hat{c},\mathsf{puz},\tilde{c},\tilde{r}] \cdot \delta(\lambda) = \delta(\lambda).
\end{aligned}$$

II. $\mathbf{D}^{\mathcal{V}^*}_{PoW}, \tilde{\mathbf{D}}^{\mathcal{V}^*}$ **are computationally indistinguishable**. When running, in the PoW mode of $(\mathcal{P}, \mathcal{V}^*)$, the challenge c for the LS protocol is computed by the group operation of a value $\hat{c}$ provided by $\mathcal{V}^*$ and a value c uniformly sampled from $\{0,1\}^{\ell(\lambda,h)}$. Thus, in the PoW mode of $(\mathcal{P}, \mathcal{V}^*)$, c follows the same (uniform) distribution that c follows in $(\tilde{\mathcal{P}}, \mathcal{V}^*)$. This implies that the distribution of $c, a, \hat{c}, \mathsf{puz}, \tilde{c}, r$ in the PoW mode of $(\mathcal{P}, \mathcal{V}^*)$ is identical with the distribution in $(\tilde{\mathcal{P}}, \mathcal{V}^*)$. We continue as in case I in a "symmetric" way, i.e. we now show the computational indistinguishability of $\mathbf{D}^{\mathcal{V}^*}_{PoW}, \tilde{\mathbf{D}}^{\mathcal{V}^*}$ by taking advantage of the computational auxiliary input sHVZK property of Π.

□

D Security of Spam Reducing System

We present a proof sketch of Theorem 6.
Proof:(sketch)
Spam Resistance. We start by constructing a knowledge extractor $\mathcal{K}$ which on input $(z, 1^\lambda, h, x)$ and given access to a prover $\mathcal{A}$, uses the special soundness PPT extractor K_Π of Π to extract a witness. Our $\mathcal{K}$ works similarly to the soundness extractor of PoWorK (see proof of Theorem 1), but can now rewind $\mathcal{A}$ at any point and give it two different challenges $\mathbf{c}_i, \mathbf{c}'_i$ (as it controls the random oracle), to receive tuples $(\mathbf{a}_i, \mathbf{c}_i, \mathbf{r}_i)$, and $(\mathbf{a}'_i, \mathbf{c}'_i, \mathbf{r}'_i)$ on which it runs K_Π. Note that since K_Π is a PPT algorithm, $\mathcal{K}$ also runs in polynomial time.

Now assume that for some $z \in \{0,1\}^*, h \in \mathcal{HS}_\lambda$, there exists an adversary $\mathcal{A}$ and a non-negligible function $\alpha_1(\cdot)$ s.t.

$$\Pr\left[\begin{array}{l} (t,x) \leftarrow \mathit{ReceiverSetup}(1^\lambda, h); \forall 1 \le i \le k : \mathsf{puz}_i \leftarrow \mathsf{Sample}(1^\lambda, h); \\ \{\pi_i = ((\mathbf{a}_i, \mathbf{c}_i, \mathbf{r}_i)\}_{i\in[k]} \leftarrow \mathcal{A}(z, 1^\lambda, h, x) : \\ \big(\forall 1 \le i \le k :\ \mathit{ApproveEMail}(h, x, \pi_i) = 1\big) \wedge (\forall i \ne j \in [k] : \pi_i \ne \pi_j) \\ \wedge \Big(\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, x) \le \sqrt{\tau \circ g}\big(\sum_{i=1}^k \mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i)\big)\Big) \end{array}\right] = \alpha_1(\lambda).$$

By an averaging argument, there exist a statement x and and public parameters v s.t.

$$\Pr\left[\begin{array}{l} \forall 1 \le i \le k : \mathsf{puz}_i \leftarrow \mathsf{Sample}(1^\lambda, h); \\ \{\pi_i = ((\mathbf{a}_i, \mathbf{c}_i, \mathbf{r}_i)\}_{i\in[k]} \leftarrow \mathcal{A}(z, 1^\lambda, h, x) : \\ \big(\forall 1 \le i \le k :\ \mathit{ApproveEMail}(h, x, \pi_i) = 1\big) \wedge (\forall i \ne j \in [k] : \pi_i \ne \pi_j) \\ \wedge \Big(\mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, x) \le \sqrt{\tau \circ g}\big(\sum_{i=1}^k \mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i)\big)\Big) \end{array}\right] \ge \alpha_1(\lambda).$$

Using $\mathcal{A}$ we will construct an algorithm $\mathcal{W}$ to break the (τ, k)-amortization resistance of PuzSys. We recall that in the non-interactive variant of our dense puzzle based PoWorK construction the format of a proof π is $(\mathbf{a}, \mathbf{c}, \mathbf{r}) = (\tilde{a}, c, (\tilde{c}, \tilde{r}, \mathsf{puz}, \mathsf{soln}))$.

$\mathcal{W}$ is given as input $(x, v), 1^\lambda, h, \{\mathsf{puz}_1, \ldots, \mathsf{puz}_k\}$, where $\forall 1 \le i \le k : \mathsf{puz}_i \leftarrow \mathsf{Sample}(1^\lambda, h)$. Then $\mathcal{W}$, who also controls the random oracle, runs as follows:

1. Invoke $\mathcal{A}$ with input $(1^\lambda, h, x)$.
2. For every i-th RO query of $\mathcal{A}$ $((a)_i, m_i)$ respond by a challenge c_i which can be honestly generated by asking H (thus, $c_i \in \mathcal{CS}_\lambda$). $\mathcal{W}$ stores all $c_1, \ldots, c_{k'}$ in a table T along with the corresponding query of $\mathcal{A}$. Note that $k' \ge k$.
3. Receive $\mathcal{A}$'s output $\pi_1, \ldots, \pi_k = (\tilde{a}_1, c_1, (\tilde{c}_1, \tilde{r}_1, \mathsf{puz}_1, \mathsf{soln}_1)), \ldots, (\tilde{a}_k, c_k, (\tilde{c}_k, \tilde{r}_k, \mathsf{puz}_k, \mathsf{soln}_k))$.
4. Look at the first proof π_1 of $\mathcal{A}$, locate the corresponding c_1 in table T (let r be the row in which found), and rewind $\mathcal{A}$ just before the point it made that query, i.e. at its $r-1$ query. With high probability $\mathcal{A}$ will start making the same RO queries. For every query from 1 to $r-1$ return the same c as before. However, when $\mathcal{A}$ makes its r-th query return $c_r = \tilde{c_1} \oplus \mathsf{puz}_1$. For the rest of the queries (from $r+1$ and on) return a random

challenge as in Step 2 and update table T with the fresh values. When $\mathcal{A}$ outputs its second set of proofs $\pi_1^{(2)}, \ldots, \pi_k^{(2)}$[10] check that puz_1 is included in $\pi_1^{(2)}$ and store the corresponding solution.

5. Proceed until all k solutions have been found, i.e. in the i-th rewind the new challenges are $c_1^{(i)}, \ldots, c_{k'}^{(i)}$, where $c_i^{(i)} = \tilde{c}_i \oplus \mathsf{puz}_i^{-1}$, $\forall r < i : c_r^{(i)} = c_r^{(i-1)}$ and all the rest of the challenges $c_{i+1}^{(i)}, \ldots, c_{k'}^{(i)}$ are honestly sampled. When $\mathcal{A}$ outputs its i-th set of new proofs $\pi_1^{(i)}, \ldots, \pi_k^{(i)}$ check that the corresponding puzzle included in the proof $\pi_i^{(i)}$ is equal to puz_i and store its solution soln_i.

6. Output $\mathsf{soln}_1, \ldots, \mathsf{soln}_k$.

We follow the reasoning of the proof of Theorem 1. For each rewinding i of $\mathcal{A}$, we have that when it received honestly selected sequences $c_1^{(i-1)}, \ldots, c_{i-1}^{(i-1)}, c_i^{(i-1)} \ldots, c_{k'}^{(i-1)}$ (in its $i-1$-th run) and $c_1^{(i)}, \ldots, c_{i-1}^{(i)}, c_i^{(i)} \ldots, c_{k'}^{(i)}$ in its i-th run (where $c_1^{(i)}, \ldots, c^{(i)} = c_1^{(i-1)}, \ldots, c_{i-1}^{(i-1)}$), it outputs accepting transcripts in no more than

$$\Big[(\sqrt{\tau \circ g})\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)\Big]$$

steps and with probability $\alpha_1(\lambda)^2/4$. Similar to the PoWorK soundness proof, we denote by Equal_i, the event that this happens and $\tilde{c}_i^{(i-1)} = \tilde{c}_i^{(i)}$ holds (again for each rewinding i). Obviously, either Equal_i, or $\neg\mathrm{Equal}_i$ will occur with at least $\alpha_1(\lambda)^2/8$ probability. We distinguish the following cases:

Case I. $\forall i \in [k] : \Pr[\mathrm{Equal}_i] \geq \alpha_1(\lambda)^2/8$**:** in this case, as in the soundness proof of Theorem 1, with probability $\alpha_1(\lambda)^2/8 - \mathsf{negl}(\lambda)$ it holds that:

1. $\forall i \in [k] : \mathsf{Verify}(1^\lambda, h, \mathsf{puz}_i, \mathsf{soln}_i) = \mathsf{true}$.

2. The running time of $\mathcal{W}$ in number of steps is no more than

$$k \cdot \Big[(\sqrt{\tau \circ g})\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)\Big]$$

steps.

Since k is polynomial we have that w.h.p. $k \leq (\sqrt{\tau \circ g})\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)$. In addition, τ is an increasing function and g is a subadditive function, hence we have that

$$k \cdot \Big[(\sqrt{\tau \circ g})\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)\Big] \leq \Big[(\sqrt{\tau \circ g})\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big)\Big]^2 \leq$$
$$(\tau \circ g)\big(\sum_{i=1}^{k}(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big) \leq \tau\big(\sum_{i=1}^{k} g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}_i))\big).$$

Therefore, $\mathcal{W}$ breaks the (τ, k)-amortization resistance property of PuzSys.

Case II. $\exists i^* \in [k] : \Pr[\neg\mathrm{Equal}_{i^*}] \geq \alpha_1(\lambda)^2/8$**:** in this case, we set the knowledge extractor to guess a priori an $i \in [k]$ to rewind $\mathcal{A}$ expecting to invoke $\mathcal{K}_\Pi$ with two different challenges $\tilde{c}_i^i = \tilde{c}_i$. We stress that this setting is black-box and independent of $\mathcal{A}$, thus consistent with the definition of spam resistance. By the soundness property of Π, if $\mathcal{K}$ guesses i^* correctly, then it will return a witness for x. Therefore, $\mathcal{K}$ is succesfully returns a witness with at least $\alpha_2(\lambda) = \alpha_1(\lambda)^2/(8k)$ probability.

[10]From now on the superscript $x^{(\cdot)}$ denotes in which rewinding of $\mathcal{A}$ we are.

• **Privacy.** Let $h \in \mathcal{HS}_\lambda, z \in \{0,1\}^*$ and an adversary $\mathcal{A}$ that breaks SRC privacy with non-negligible advantage $\alpha(\lambda)$. By an averaging argument, there exist a statement x, a witness $w \in R_\mathcal{L}(x)$ and public parameters v s.t.

$$\big| \Pr[\pi \leftarrow \textit{SendEMail}(w, h, x) : \mathcal{A}(z, h, x, \pi) = 1] - \\ - \Pr[\pi \leftarrow \textit{SendEMail}(\perp, h, x) : \mathcal{A}(z, h, x, \pi) = 1]\big| \geq \alpha(\lambda).$$

Given $\mathcal{A}$ we construct an adversary $\mathcal{B}$ against PoWorK statistical indistinguishability that on input a statement x, auxiliary input z, h and a PoWorK proof π (i.e. the view of $\mathcal{B}$ either in PoK mode on witness w or PoW mode), invokes $\mathcal{A}$ on input (z, h, x, π) and returns $\mathcal{A}$'s output. It is straightforward that $\mathcal{B}$ distinguishes the mode of the PoWorK prover with advantage $\alpha(\lambda)$.

□

E Straight-line Simulation Definitions

Being able to perform straight-line simulation (i.e. simulation without rewinding) is helpful in the concurrent setting. Pass [Pas03, Pas04] has shown that protocols satisfying straight-line strong $T(\lambda)$-simulatability (where $T(\lambda)$ is a class of functions closed under composition with any polynomial) are also *concurrent* $T(\lambda)$-strongly simulatable. We start by defining straight-line $T(\lambda)$-simulatability.

Definition 7 ([Pas04]) *Let $T(\lambda)$ be a class of functions that is closed under composition with any polynomial. We say that an interactive argument $(\mathcal{P}, \mathcal{V})$ for the language $\mathcal{L} \in \mathcal{NP}$ with witness relation $R_\mathcal{L}$, is* straight-line strongly $T(\lambda)$-simulatable*, if for every probabilistic verifier V^* with running time bounded by $T(\lambda)$, there exists a probabilistic simulator S with running time bounded by $T(\lambda)$ such that the following two ensembles are strongly $T(\lambda)$-indistinguishable:*

(i). $\{view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V}^* \rangle(x, z)\}_{x \in \mathcal{L},\, w \in R_\mathcal{L}(x), z \in \{0,1\}^*}$

(ii). $\{\langle S \leftrightarrow \mathcal{V}^* \rangle(x, z)\}_{x \in \mathcal{L}, z \in \{0,1\}^*}$

That is, for every probabilistic algorithm D running in time $T(\cdot)$ in the length of its first input, all sufficiently long $x \in \mathcal{L}$, all $w \in R_\mathcal{L}(x)$ and all auxiliary inputs $z, z' \in \{0,1\}^*$, it holds that

$$|\Pr[D(x, z', view_{\mathcal{V}^*} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V}^* \rangle(x, z)) = 1] - \Pr[D(x, z', S(x, z)) = 1]| < \frac{1}{T(|x|)}.$$

The notion of *perfect* (resp. *statistical*) $T(\lambda)$-simulatability is defined similarly, by requiring that the two ensembles in Definition 7 are identically (resp, statistically close distributed) for every (computationally unbounded) verifier V^*. The notion above could be further restricted to guarantee security under concurrent executions. Pass in [Pas04] provides the following definition.

Definition 8 ([Pas04]) *Let $T(\lambda)$ be a class of functions that is closed under composition with any polynomial. We say that an interactive argument $(\mathcal{P}, \mathcal{V})$ for the language $\mathcal{L} \in \mathcal{NP}$ with witness relation $R_\mathcal{L}$, is* straight-line concurrent $T(\lambda)$-simulatable*, if for every PPT oracle machine A that is not allowed to restart of rewind the oracle it has access to, and every polynomial $p(\lambda)$, there exists a probabilistic simulator $S(i, x)$ with running time bounded by $T(\lambda)$ such that the following two ensembles are computationally indistinguishable:*

(i). $\left\{A^{P(x_1,w_1),\cdots,P(x_{p(\lambda)},w_{p(\lambda)})}(z, x_1, \ldots, x_{p(\lambda)})\right\}_{z \in \{0,1\}^*, x_1,\ldots,x_{p(\lambda)} \in \mathcal{L}, \{w_i \in R_\mathcal{L}(x_i)\}_{[p(\lambda)]}}$

(ii). $\left\{A^{S(1,w_1),\cdots,S(p(\lambda),w_{gp\lambda)})}(z, x_1, \ldots, x_{p(\lambda)})\right\}_{z \in \{0,1\}^*, x_1,\ldots,x_{p(\lambda)} \in \mathcal{L}}$