Indistinguishable Proofs of Work or Knowledge

Foteini Baldimtsi *, Aggelos Kiayias *, Thomas Zacharias *, and Bingsheng Zhang *

[1] George Mason University, USA
foteini@gmu.edu
[2] University of Edinburgh, UK
akiayias@inf.ed.ac.uk
[3] University of Edinburgh, UK
tzachari@inf.ed.ac.uk
[4] Security Lancaster Research Centre, Lancaster University, UK
b.zhang2@lancaster.ac.uk

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 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 a number of 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 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, shows how PoWorK can be used to *compose* cryptocurrencies that are based on proofs of work ("Bitcoin-like") with cryptocurrencies that are based on knowledge relations (these include cryptocurrencies that are based on "proof of stake", and others). The resulting PoWorK-based cryptocurrency inherits the robustness properties of the underlying two systems while PoWorK-indistinguishability ensures a uniform population of miners. Finally, we show that PoWorK protocols imply straight-line quasi-polynomial simulatable arguments of knowledge and based on our construction we obtain an efficient straight-line concurrent 3-move statistically quasi-polynomial simulatable argument of knowledge.

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

* Work performed while at the National and Kapodistrian University of Athens. Research supported by ERC project CODAMODA, #259152 and H2020 Project Panoramix #653497. Baldimtsi also did part of this work while at Boston University supported by NSF #1012910.

1 Introduction

We introduce a new class of prover verifier protocols where the prover wishes to convince the verifier that 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., [CDS94, 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 languages in $\mathcal{NP}$ can be shown with a (computational) HVZK protocol via reduction to e.g., the Hamilton cycle protocol [Blu87]. The class of Σ-protocols possesses very useful properties including being closed under conjunction and disjunction operations [CDS94].

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 define and construct efficient *three-move* PoWorK protocols as well as relevant cryptographic puzzles. Morerover, we demonstrate how PoWorK can instantiate systems that reduce email spam while preserving user privacy, how they are useful in composition of cryptocurrency systems and how they can give rise to concurrent simulatable protocols. In more details:

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.

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 RO model).

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 3).

Our puzzle definitions are close in spirit to previous formalizations [RSW96, WJHF04, CMSW09, MMV11, BGJ+16] with the following distinctions. In [CMSW09] the hardness of a puzzle is defined 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 to cryptocurrencies. The requirement there is that the puzzle solver should spend a minimum of computational resources to find a solution to the puzzle (and may or may not choose to parallelize).

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 is related to cryptocurrencies based on blockchains to maintain the ledger of transactions. These systems can be naturally divided by the mechanism they use to produce the next block in the blockchain as follows: first there are "puzzle-based" ones, (e.g., Bitcoin [Nak08] and many others that followed[5]), and then there are "knowledge-based" ones, that include those[6] that use "proof-of-stake", "proof-of-activity" or other type of consensus mechanism that relies e.g., on a public-key infrastructure, e.g., [BLMR14, DM16, Maz15]). We demonstrate how given two cryp-

[5] E.g., Litecoin, Dogecoin, Ethereum, Dashcoin, etc.

[6] E.g., Peercoin, NXT, Nushares, Faircoin etc.

tocurrencies $\mathtt{C}_1, \mathtt{C}_2$ of each type, one can use PoWorK to fuse them into a single cryptocurrency $\mathtt{C}$ with the following properties: (i) in $\mathtt{C}$, the miners that perform $\mathtt{C}_1$-type of mining are indistinguishable from those that perform $\mathtt{C}_2$-type of mining, (ii) $\mathtt{C}$ would reach consensus in the sense of persistence of transactions in the ledger under the conjunction of the conditions that systems $\mathtt{C}_1, \mathtt{C}_2$ would do, (iii) $\mathtt{C}$ would satisfy liveness under the disjunction of the conditions that systems $\mathtt{C}_1, \mathtt{C}_2$ would do.[7] PoWorK-based cryptocurrencies that fuse the knowledge-based and the puzzle-based approach have novel features in the context of cryptocurrencies: for instance, by composing a regular Bitcoin-like cryptocurrency $\mathtt{C}_1$ with a centralized cryptocurrency $\mathtt{C}_2$ supported by a single authority, we get a cryptocurrency $\mathtt{C}$ that resembles Bitcoin but has a trusted authority with a trapdoor that enables it to regulate and normalize the block production rate. Such systems may offer a more attractive solution for nation-states or central banks that wish to issue centralized cryptocurrencies, however they do not want to be constantly involved with block production and they prefer to leave ledger maintenance to the public, while retaining the ability to issue blocks in case of an emergency situation (e.g., many miners go offline due to a software problem). The PoWorK indistinguishability property is critically useful in this setting, since it enables the regulation of the block production rate made by the trusted party to be indistinguishable to everyone, thus ensuring that the trusted party's involvement will be unnoticed and hence will have no impact to the economy that the cryptocurrency supports.

Our third 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. Given our 3-move PoWorK construction, this immediately yields a 3-round protocol in this setting which is optimal in terms of efficiency (round complexity is optimal and computational overhead is just two exponentiations for prover and verifier in total when using the elliptic curves from [BHKL13]); we note that a similar result in terms of rounds can be obtained via a different route, specifically, via the efficient OR composition with an input-delayed Σ-protocol as recently observed in [CPS+16], however the resulting complexity overhead would be at least 5 exponentiations for prover and verifier in total when instantiated using discrete logarithms.

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

[7] For definitions of properties like liveness and persistence of the ledger we refer to e.g., [GKL15, BMC+15].

section, we present two dense puzzle instantiations. Finally, in Section 4, we describe the applications of PoWorKs. Namely, (i) a method to reduce the amount of spam email while preserving the privacy of the receiver, (ii) the composition of knowledge-based and puzzle-based cryptocurrencies that gives rise to PoWorK-based cryptocurrencies, (iii) an efficient 3-move straight-line concurrent statistically $\lambda^{\text{poly}(\log \lambda)}$-simulatable argument of knowledge as defined in [Pas03, Pas04].

Alternative PoWorK constructions. In the full version of this work [BKZZ15] 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. A third way to construct PoWorK's can be derived from the recent efficient OR composition technique that was introduced in [CPS+16] that can be used with "input-delayed" Σ-protocols, where the statement need not be determined ahead of time. It is easy to see that in the case a puzzle accepts an "input-delayed" Σ proof of knowledge of the puzzle solution (e.g., a puzzle based on discrete-logarithms), a third possible construction method for PoWorK's is facilitated. We stress however that these alternative methods for constructing PoWorK's do not combine well with puzzles based on hash functions and thus may be of only theoretical interest in the context of our primitive.

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 $z \xleftarrow{\$} \mathcal{Z}$ 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 denoted 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[8]. 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 $|w| \leq \text{poly}(|x|)$), 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

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

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 $h \in \mathcal{HS}_\lambda$:

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

Note that the number of steps that Solve takes to run is monotonically decreasing in the hardness factor h and may exponentially depend on λ, while Verify should run in polynomial time in λ.

g-Hardness: We say that a puzzle system PuzSys is *g-hard* for some function g, 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 $(\mathsf{puz}, \mathsf{soln}) \in \mathcal{PS}_\lambda \times \mathcal{SS}_\lambda$.

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

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

Efficiency of Sampling: 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) $\Pr[out_{\mathcal{V}} \leftarrow \langle \mathcal{P}(w) \leftrightarrow \mathcal{V} \rangle(x,z,h) : out_{\mathcal{V}} = \text{accept}] > 1 - 1/\mathrm{poly}(\lambda)$ *and*

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

(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* $\pi_{x,y,z,h,\lambda}$ *the probability*

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

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 for any $\lambda \in \mathbb{N}, x \in \{0,1\}^{\mathrm{poly}(\lambda)}, y, z \in \{0,1\}^*, h \in \mathcal{HS}_\lambda$, if $\pi_{x,y,z,h,\lambda} \geq s(\lambda)$ (representing the knowledge error), then

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

(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:*

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

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 steps) 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 less than $f(T)$ steps 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) into a 3-move public-coin PoWorK. Our construction is lightweight and requires dense samplable puzzle systems that we formalized in Section 1. In our full version [BKZZ15] we provide a second construction 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). 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 require correctness, efficient samplability, and statistical indistinguishability.

3.1 Preliminaries

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 $\mathsf{ChSampler}$ 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,

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

3.2 The Dense Puzzle Based Compiler

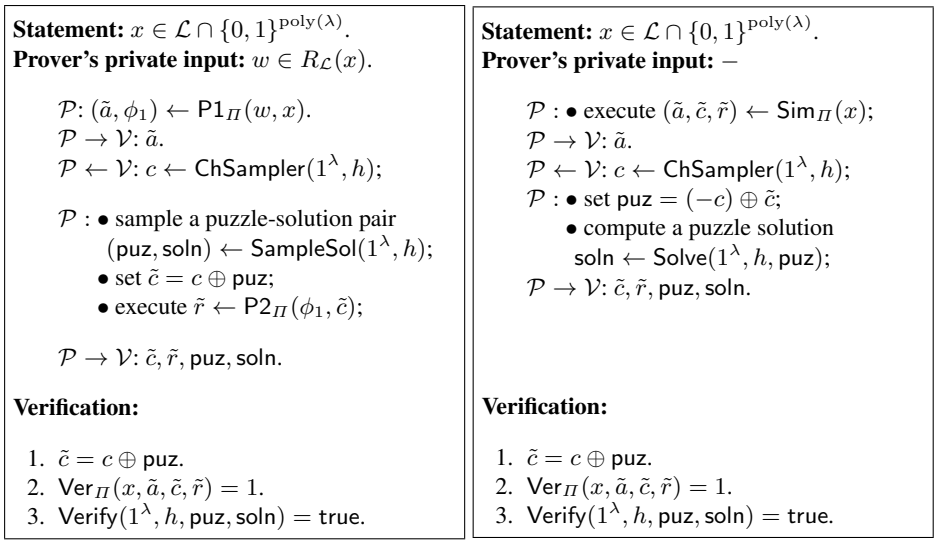
We now provide a detailed description of our protocol $(\mathcal{P}, \mathcal{V})$, which can be viewed as a compiler that can transform a SS-HVZK protocol $\mathit{\Pi} = (\mathsf{P1}_{\mathit{\Pi}}, \mathsf{P2}_{\mathit{\Pi}}, \mathsf{Ver}_{\mathit{\Pi}})$ for $\mathcal{L} \in \mathcal{NP}$ and a g-hard 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 $\mathcal{CS}_{\mathit{\Pi}}$. We denote by $\mathsf{Sim}_{\mathit{\Pi}}$ the HVZK simulator of $\mathit{\Pi}$.

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 $\mathsf{P1}_{\mathit{\Pi}}$ and $\mathsf{P2}_{\mathit{\Pi}}$ as described by the original SS-HVZK protocol, with the difference that instead of providing $\mathsf{P2}_{\mathit{\Pi}}$ 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 $\mathsf{P2}_{\mathit{\Pi}}$ 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 $\mathsf{puz} = (-c) \oplus \tilde{c}$. 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.



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

Fig. 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 $\mathcal{PS}_\lambda \subseteq \mathcal{CS}_\lambda = \mathcal{CS}_\Pi$; ChSampler is the challenge sampling algorithm over $\mathcal{CS}_\lambda$.

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.
(**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 $\mathcal{CS}_\lambda$.

(**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 $\mathcal{CS}_\lambda$.

Fig. 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.

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 solving a puzzle is believed to be 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**) and (**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:

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

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

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, h) and given the code of 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

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

We construct an algorithm $\mathcal{W}$ that makes 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 $(-\mathsf{puz})$.
4. It computes $c_2 = \tilde{c}_1 \oplus (-\mathsf{puz})$.
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. By Equal we denote the event that this happens and $\tilde{c}_1 = \tilde{c}_2$ holds. Obviously, either Equal, or $\neg$Equal will occur with probability at least $(s(\lambda)/2)^2/2 = s(\lambda)^2/8$.

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

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

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 $c_2 = \tilde{c}_1 \oplus (-\mathsf{puz})$ 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

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

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

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

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

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

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 $\Pr[\text{Equal}] \geq s(\lambda)^2/8$, we have that for auxiliary tape (x,y,z) and hardness factor h:

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

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

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

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), when $\mathcal{K}$ is given oracle access to $\mathcal{P}^*$ it holds that

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

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 that $\mathcal{B}$ can distinguish between:

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

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 $\mathsf{puz} = (-c) \oplus \tilde{c}$ 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 $\mathsf{puz} = (-c) \oplus \tilde{c}$ 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 $\mathsf{puz} = (-c) \oplus \tilde{c}$, 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

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

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:

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

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.

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 samplable, statistically indistinguishable, and g-hard, where $g(T) = T^{1/c}$, for any constant $c > 2$. In addition, for any k that is $O(2^{\lambda/8})$,* PuzSys *is $(\mathsf{id}(\cdot), k)$-amortization resistant, where $\mathsf{id}(\cdot)$ is the identity function.*

Proof. Please see the full version [BKZZ15].

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.

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 $\mathsf{LSB}_h(z) = \mathsf{LSB}_h(H(\mathsf{soln}, y))$.

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

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 $\psi^U : \mathcal{X} \times \{0,1\}^d \mapsto \{0,1\}^m \times \{0,1\}^d$ 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 construct TCR strong extractors from *regular universal oneway hash functions* (UOWHFs), initially proposed by Naor and Yung [NY89]. We first formally define the TCR property for a 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]. 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].[9]

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]).

Dense Oneway Functions and Dense Puzzles from Complexity Assumptions.
We apply a TCR strong extractor for 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 3. *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 $(H_{\lambda_1}, \epsilon)$-TCR strong extractor, then

$$\psi^U_{\lambda_1,\lambda_2}(x, s_1, s_2) = (\mathsf{Ext}_{\lambda_2}(\zeta_{\lambda_1}(\psi_{\lambda_1}(x)), (s_1, s_2)), s_2)$$

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. Please see the full version [BKZZ15].

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

[9] 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.

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).

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 a similar assumption was used by Gennaro to construct a more efficient pseudo-random generator [Gen00]. It is easy to see that this assumption 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].

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 4. *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*

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 $z = \mathsf{Ext}_\lambda(\psi_G(\mathsf{soln} + 2^h \cdot y), (s_1, s_2))$.

Fig. 4: The Dense Puzzle System From DLP.

$O(2^{\log^3\lambda})$, PuzSys *is* $(\mathsf{id}(\cdot), k)$*-amortization resistant, where* $\mathsf{id}(\cdot)$ *is the identity function.*

Proof. Please see the full version [BKZZ15].

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 instantiate our PoWorK protocol as described in Figure 1 by building it upon the Schnorr identification scheme [Sch89] and the dense puzzle system instantiation in the RO model[10] (see Section 3.4). The description of our instantiation is presented in the full version of this work [BKZZ15].

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

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

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 PoWorK in practice, the time that takes a prover to respond to a challenge is not a distinguishing factor: the prover might have as well solved the puzzle in constant time by fully parallelizing its computation or alternatively, for the case of non-interactive PoWorK's the receiver may not know when the prover started proof computation. Finally note that in any case, 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:

> "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[11] 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

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

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 [12].

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). 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. $w = \perp$) 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[13], and H is a hash. 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 $\mathcal{R}$.

[12] 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.

[13] 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

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). In the full version of this work [BKZZ15] we discuss some interesting extensions of our protocol that address revocation, prevention of witness sharing and solving "useful" puzzles.

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.

Definition 5. *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* $\alpha_1(\cdot), \alpha_2(\cdot)$*:*

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

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:*

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

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

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

Theorem 5. *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 super-polynomial in* λ*. Then, the email system described above is* private *and* $(\sqrt{\tau \circ g}, k)$-spam resistant.

Proof. Please see the full version [BKZZ15].

Intuitively, the privacy holds because of the indistinguishability of PoWorK . The $(\sqrt{\tau \circ g}, k)$-*spam resistance* property holds because of the soundness of PoWorK and the amortization resistance of the underlying PuzSys.

4.2 PoWorK-based Cryptocurrencies

Proofs of work is the basic primitive used in achieving the type of distributed consensus required in cryptocurrencies, notably Bitcoin [Nak08] and many others that use the same approach. The main idea is that a proof of work operation can be used to calibrate the ability of parties to build a hash chain that contains transaction records, commonly referred to as the blockchain.

An important feature of a blockchain is its decentralized nature. Given the view of a participant (commonly referred to as a miner) that includes its view of the blockchain, a fresh instance of a puzzle of a specified difficulty is created (which itself may depend on the blockchain) and has to be solved in order to add another block in the chain. Formally, the operation of a PoW-based miner as used in Bitcoin and numerous other cryptocurrencies (such as Litecoin, Namecoin, Dogecoin) is as shown in Figure 5.

Let $\langle B_1, \ldots, B_n \rangle$ be the current blockchain where B_i is a tuple (t_i, T_i, u_i, π_i) with t_i a time-stamp, T_i a set of transactions, $u_i = H(B_{i-1})$ (for a hash function H) and π_i is such that $\mathsf{Verify}(1^\lambda, h_i, H(B_i), \pi_i) = \mathsf{true}$. The hardness h_i is calculated via a function operating on the time-stamps as follows $h_i = \mathsf{HC}(t_1, \ldots, t_{i-1})$. A new block B_{n+1} is created as follows.

1. Collect transactions into a vector T_{n+1}.
2. Calculate $h_{n+1} = \mathsf{HC}(t_1, \ldots, t_n)$.
3. Set $\mathsf{puz} = H(t_{n+1}, T_{n+1})$ where t_{n+1} is a current timestamp and run $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$ to produce a $\mathsf{soln} = \pi_{n+1}$.
4. If the above step is successful, broadcast $B_{n+1} = (t_{n+1}, T_{n+1}, u_{n+1}, \pi_{n+1})$.

Fig. 5: Miner operation in a puzzle-based cryptocurrency (using a puzzle $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$ that is dense). $\mathsf{HC}(\cdot)$ is the puzzle hardness calculation function which depends on the timestamps of the blocks of the current blockchain.

Under certain assumptions about the network synchronicity and the hardness of the proof, the above mechanism has been shown to be robust in the sense of satisfying two properties, *persistence* (transactions remain stable in the "ledger") and *liveness* (all transactions are eventually inserted in the ledger) assuming that the honest parties are above majority [GKL15]. Puzzle-based cryptocurrencies have also drawn a lot of criticism due to the fact that they require a lot of natural resources (e.g., in [OM14] it is reported that Bitcoin mining in 2014 already consumed as much energy as the needs of the country of Ireland for electricity).

This lead to the development of a number of systems that circumvent puzzles (including, [DM16, BLMR14, Maz15] as well as Peercoin, DasHCoin, NXT, Nushares, ACHCoin, Faircoin and others). These systems maintain a blockchain as well, however they rely on different mechanisms for producing blocks. We call them, generically, "knowledge-based cryptocurrencies" since the production of a block is associated with the production of a witness for a public-relation relation $\mathcal{R}$ which parameterizes the system. Formally, we present the miner[14] operation in Figure 6.

Let $\langle B_1, \ldots, B_n \rangle$ be the current blockchain where B_i is a tuple (t_i, T_i, u_i, π_i), for t_i, T_i, u_i defined as in Figure 5 and π_i being a NIZK that shows $x_i \in \{x \mid \exists w : (x, w) \in \mathcal{R}\}$, where $x_i = V(B_1, \ldots, B_{i-1}, t_i, T_i)$ for $i = 1, \ldots, n$. The miner, equipped with secret-key sk, produces the next block as follows.

1. Collect transactions into a vector T_{n+1}.
2. Calculate the pair $(x_{n+1}, \mathsf{aux}) \leftarrow V(B_1, \ldots, B_n, t_{n+1}, T_{n+1})$ where t_{n+1} is the current time. Then calculate $W_{sk}(x_{n+1}, \mathsf{aux}) = w_{n+1}$. If $w_{n+1} \neq \perp$ it holds that $(x_{n+1}, w_{n+1}) \in \mathcal{R}$.
3. If the above step is successful, compute a NIZK proof π_{n+1} for x_{n+1} using witness w_{n+1}.
4. Broadcast $B_{n+1} = (t_{n+1}, T_{n+1}, u_{n+1}, \pi_{n+1})$.

Fig. 6: Miner operation in a knowledge-based cryptocurrency parameterized by relation $\mathcal{R}$. The function $V(\cdot)$, given the blockchain information, the current set of transactions and the time-stamp produces a statement x, while the function $W_{sk}(\cdot)$ given a statement produces a witness w so that $(x, w) \in \mathcal{R}$.

A trivial way to construct a knowledge-based cryptocurrency would be to have a a single trusted authority with a public and secret key pair, (pk, sk), acting as the sole miner.[15] At a time-step $n+1$, the function $V(\cdot)$ would set simply $x_{n+1} = (t_{n+1}, T_{n+1}, u_{n+1})$ and $W_{sk}(x_{n+1})$ would produce a signature on x_{n+1} that would serve as π_{n+1} (there is no need for a NIZK). Another example of a knowledge-based cryptocurrency is NXT. On a high level, in this system each miner (called forger) has a digital signature public and secret key, (pk, sk), associated with her account. The function $V(B_1, \ldots, B_n, t_{n+1}, T_{n+1})$ (run by each miner), operates as follows: it parses T_{n+1} to recover the public pk of the miner (note that it is always present in the transaction collecting the fees). Then, based on the public-key pk and the blockchain $B_1, \ldots, B_n$ it determines how much currency is associated with the account that corresponds to the public-key pk; this results in a time-window $d \in \mathbb{R}^+$ whose expectation is proportionate to the amount of currency in the account (the more currency, the shorter the expectation of d is; we omit the exact dependency in this high level description). The

[14] Note that we use the term "miner" for symmetry. Miners are associated with puzzle based cryptocurrencies and thus different terminology has been introduced in knowledge-based systems including "mintettes", "forgers" and others.

[15] For instance, this would be a single "mintette" instantiation of [DM16].

function $V(\cdot)$ returns (x_{n+1}, aux) with $x_{n+1} = (t_{n+1}, T_{n+1}, u_n)$ and $\mathsf{aux} = d$. The procedure $W_{sk}(x_{n+1}, d)$, will produce a signature w on the message (t_{n+1}, T_{n+1}, u_n) if $t_{n+1} \geq t_n + d$; else, it produces $\perp$. Note that in this system no NIZK is employed, one may just set $\pi_{n+1} = w$; however, the system would operate similarly if a NIZK was employed to establish knowledge of a signature w on the message (t_{n+1}, T_{n+1}, u_n).

We now show how to construct a PoWorK-based cryptocurrency derived from a knowledge-based cryptocurrency $\mathtt{C}_1$ and a puzzle-based cryptocurrency $\mathtt{C}_2$ for a dense puzzle, see Figure 7. The construction is straightforward: a new block can be added to the blockchain by someone who can efficiently compute a proof π_i using some secret key or by someone who is computing a π_i by performing computational work.

The properties of the composition are informally stated in the following (meta)-theorem; the proof of the theorem follows from the properties of PoWorK and is similar in spirit to the proof of Theorem 5. The formal statement and proof of the theorem (that should also include a formalization of all relevant underlying properties of cryptocurrencies, both in the puzzle-based and knowledge-based setting, e.g., in the sense of [GKL15]) is out of scope for the present exposition.

Let $\langle B_1, \ldots, B_n \rangle$ be the current blockchain where B_i is a tuple (t_i, T_i, u_i, π_i), for t_i, T_i, u_i defined as in Figure 5 and π_i being a non-interactive PoWorKthat demonstrates either the solution of the puzzle $\mathsf{puz} = H(t_i, T_i)$ with hardness $h_i = R(t_1, \ldots, t_{i-1})$ or that $x_i \in \{x \mid \exists w : (x, w) \in \mathcal{R}\}$ where $x_i = V(B_1, \ldots, B_{i-1}, t_i, T_i)$.

1. Collect transactions into a vector T_{n+1}.
2. If a secret-key sk is available, perform steps 2-3 of Figure 6 and follow the PoK direction of PoWorK(cf. Figure 1), using the $H(\cdot)$ to compute the challenge of the verifier.
3. Else, perform steps 2-3 of Figure 5 and follow the PoW direction of PoWorK(cf. Figure 1) using the $H(\cdot)$ to compute the challenge of the verifier.
4. Broadcast $B_{n+1} = (t_{n+1}, T_{n+1}, u_{n+1}, \pi_{n+1})$.

Fig. 7: Miner operation in a PoWorK-based cryptocurrency parameterized by relation $\mathcal{R}$ and $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$. The functions $V(\cdot), W_{sk}(\cdot)$ are as in Figure 5 and the function $C(\cdot)$ is as in Figure 6.

Theorem 6. *(informally stated) The cryptocurrency $\mathtt{C}$ of Figure 7 is the composition of a knowledge-based cryptocurrency $\mathtt{C}_1$ and a puzzle-based cryptocurrency $\mathtt{C}_2$ so that (i) the population of miners of $\mathtt{C}_1, \mathtt{C}_2$ becomes a single set that is indistinguishable to any adversary that controls a subset of miners of $\mathtt{C}$, (ii) the persistence property of $\mathtt{C}$ is upheld as long as the conditions for persistence of $\mathtt{C}_1, \mathtt{C}_2$ hold in conjunction. (iii) the liveness property of $\mathtt{C}$ is upheld as long as the conditions for liveness of $\mathtt{C}_1, \mathtt{C}_2$ hold in disjunction.*

4.3 PoWorKs as 3-move Straight-line Concurrent Simulatable Arguments of Knowledge

In this section, 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. Our application is described at length in our full version [BKZZ15]. Here, we provide the statement of our main result.

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 statistically $\lambda^{\mathrm{poly}(\log \lambda)}$-simulatable argument of knowledge.

Remark. In practice, we can instantiate the dense puzzle with a DL function over a dense elliptic curve [BHKL13] (without the need of an extractor). This means that we can transform a 3-move proof/argument of knowledge to a concurrent one with minimal computational overhead – 1 exponentiation for the prover and 1 exponentiation for the verifier. (cf. Fig. 1(a).) Note that a similar result in terms of rounds and with similar assumptions (i.e. DL) can be obtained via the efficient OR composition with an input-delayed Σ-protocol as recently observed in [CPS+16], however the resulting complexity overhead would be at least 3 exponentiations for the prover and 2 exponentiations for the verifier when the underlying Chameleon Σ-protocol is instantiated from Schnorr's protocol.

References

ACJT00. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*. 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 *CRYPTO*, 1999.

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

BHKL13. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *CCS*, 2013.

BKZZ15. Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. Cryptology ePrint Archive, Report 2015/1230, 2015. `http://eprint.iacr.org/2015/1230`.

BLMR14. Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

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.

BMC+15. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy*, 2015.

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.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

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.

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+16. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved or composition of sigma-protocols. TCC-A, 2016.

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.

DM16. George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *NDSS*, 2016.

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

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.

GKL15. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.

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 pseudo-random 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.

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

Maz15. David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. [Online], 2015. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`.

MMV11. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO*, 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.

OM14. Karl J. ODwyer and David Malone. Bitcoin mining and its energy footprint. In *ISSC 2014 / CIICT 2014*, 2014.

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.

WJHF04. Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS*, 2004.