Deterministic-Prover Zero-Knowledge Proofs

Hila Dahari[1] and Yehuda Lindell[2]

[1] Weizmann Institute of Science*
hila.dahari@weizmann.ac.il
[2] Bar-Ilan University
yehuda.lindell@biu.ac.il

Abstract. Zero-knowledge proof systems enable a prover to convince a verifier of the validity of a statement without revealing anything beyond that fact. The role of randomness in interactive proofs in general, and in zero-knowledge in particular, is well known. In particular, zero-knowledge with a deterministic verifier is impossible for non-trivial languages (outside of $\mathcal{BPP}$). Likewise, it was shown by Goldreich and Oren (Journal of Cryptology, 1994) that zero-knowledge with a deterministic prover is also impossible for non-trivial languages. However, their proof holds only for *auxiliary-input zero knowledge* and a *malicious verifier*.
In this paper, we initiate the study of the feasibility of zero-knowledge proof systems with a deterministic prover in settings not covered by the result of Goldreich and Oren. We prove the existence of deterministic-prover auxiliary-input *honest-verifier* zero-knowledge for any $\mathcal{NP}$ language, under standard assumptions. In addition, we show that any language with a hash proof system has a deterministic-prover honest-verifier statistical zero-knowledge proof, with an efficient prover. Finally, we show that in some cases, it is even possible to achieve deterministic-prover *uniform* zero-knowledge for a malicious verifier. Our contribution is primarily conceptual, and sheds light on the necessity of randomness in zero knowledge in settings where either the verifier is honest or there is no auxiliary input.

1 Introduction

Zero-knowledge proofs enable a prover to convince a verifier of the validity of a statement without revealing anything beyond that fact [19]. The theory and practice of zero knowledge has been studied in great depth in the more than 3 decades since its invention, and it is one of the central topics of the foundations of cryptography. In this paper, we revisit a fundamental question that was initially asked in [17]:

> *To what extent is randomness necessary in zero-knowledge proofs? In particular, can zero knowledge with a deterministic prover be achieved?*

This question was asked in [17], and they provided negative answers. First, they proved that zero knowledge with a deterministic verifier exists only for trivial

* This work was done while the author was at Bar-Ilan University.

languages. This is due to the fact that any proof with a deterministic verifier can be made non-interactive (since the prover already knows what the verifier is going to send), and non-interactive zero knowledge (without any setup) is possible only for languages in $\mathcal{BPP}$. Second, they proved that *auxiliary-input* zero knowledge with a deterministic prover exists only for trivial languages. The proof in this case is more involved, and relies inherently on the fact that the verifier may receive auxiliary input, and may not necessarily be honest. Indeed, as they point out, the honest-verifier statistical zero-knowledge proof of graph non-isomorphism in [16] has a deterministic prover. This is actually true for any language in statistical zero knowledge ($\mathcal{SZK}$), since the honest-verifier proof for statistical difference of [21] (which is a complete language for $\mathcal{SZK}$) also has a deterministic prover. We remark, however, that the deterministic prover in these cases is not efficient, and analogous protocols with an efficient prover are probabilistic [20]. The above leaves open the following questions:

1. *Can deterministic-prover honest-verifier zero knowledge be achieved for languages not in $\mathcal{SZK}$? In particular, can it be achieved for all $\mathcal{NP}$ (or beyond)?*
2. *Can deterministic-prover honest-verifier zero knowledge be achieved for non-trivial languages with an efficient prover?*
3. *Can deterministic-prover honest-verifier zero knowledge be achieved for non-trivial languages when the verifier is malicious, in a setting with no auxiliary input?*

Our interest in deterministic-prover zero knowledge is not due to any application per se, but rather due to the view that the role of randomness in computation in general, and in zero-knowledge in particular, is fundamental. Our work reopens this question that was thought to be closed. We provide answers to some of the open questions, but our work also leaves questions unanswered and highlights the fact that our understanding of this issue is far from complete.

Our results. We first consider the question of honest-verifier zero-knowledge, and prove that indeed it is possible to construct deterministic-prover zero knowledge proofs for all $\mathcal{NP}$. The idea behind our construction is for the verifier to send the randomness needed by the prover in the proof. Of course, this would not work naively since zero knowledge is only preserved if the prover's randomness is not known to the verifier. This can be solved by having the verifier send random strings, and then the prover inverts them relative to a one-way function and takes the hard-core bits of the preimages as its random tape. In order for this to work, it is necessary for the verifier to be able to sample images of the one-way function that are hard to invert, even given the randomness of the sampling algorithm. This is due to the fact that the verifier knows this randomness. Trapdoor permutations with this property are called *enhanced*, and this property is needed to prove that the oblivious transfer protocol of [6] is secure [13]. We actually need an additional enhancement, that was used in the context of achieving non-interactive zero knowledge [18]; however, we only need

1–1 one-way functions and not trapdoor permutations. In Section 2, we prove the following theorem:

Theorem 1.1. *Assuming the existence of doubly-enhanced 1–1 one-way functions, every language in $\mathcal{NP}$ has an honest-verifier auxiliary-input zero-knowledge proof with a deterministic prover.*

One of the challenges that arises when trying to instantiate Theorem 1.1 under standard assumptions (like factoring and discrete log) is that *collections* of one-way functions are needed. In particular, the prover cannot choose the function since it is deterministic, and the verifier must be able to choose the function without being able to invert it. This can be achieved assuming that the function description is *dense*, meaning that a random string is a valid function description with noticeable probability. We show that in our setting, even though the verifier cannot necessarily check if the function is valid, it is possible for the prover to do so, without sacrificing soundness. See more details in Section 2.2.

The prover in the proof of Theorem 1.1, as in the two-message proof for graph non-isomorphism of [16] and the statistical zero-knowledge proof of [21] for statistical distance, is not efficient. This seems inherent in some sense, since these $\mathcal{SZK}$ proofs all work by the verifier sending a challenge that can be solved only if the statement is correct. However, the solution to the challenge cannot be found efficiently (e.g., it requires computing graph isomorphisms and the like). We prove that for languages that have hash proof systems [3,4], it *is* possible to make the prover efficient. Our proof is also two message; however, the structure of the hash proof systems means that it is possible to solve the challenge efficiently given the witness. The following is proven in Section 3:

Theorem 1.2. *Let L be an NP-language. If there exists a hash-proof system for L, then there exists an auxiliary-input honest-verifier statistical zero-knowledge where P is an efficient deterministic algorithm.*

Finally, we consider the feasibility of achieving deterministic-prover zero knowledge for *malicious verifiers*. As described above, by the results of [17] this can only be achieved for *plain* zero knowledge (with no auxiliary input). We show that there *exist* non-trivial languages for which this can be achieved. In particular, we construct deterministic zero-knowledge proofs for languages with more than one witness, where it is hard to compute the second witness from the first. This is reminiscent of the construction of witness hiding proofs from witness indistinguishability by [9], and indeed we show that languages of the form used by them have deterministic-prover zero knowledge. In addition, we describe some natural languages for which deterministic-prover zero knowledge can be achieved (e.g., proving that a tuple is a Diffie-Hellman tuple, or proving that a ciphertext encrypts a given message). Informally stated, in Section 4 we prove:

Theorem 1.3 (informal). *Assuming the existence of one-way functions (hard to invert for non-uniform adversaries), there exist non-trivial languages which have deterministic-prover computational* plain *zero-knowledge proof systems.*

Open questions. Our work leaves open some natural open questions. First and foremost, can we categorize the languages for which deterministic-prover plain zero knowledge is feasible with malicious verifiers? Is it possible for all $\mathcal{NP}$, or at least for all "hard" languages? Providing positive or negative answers to these questions goes to core of the role of randomness in (plain) zero knowledge. Another question that arises from our work is whether or not efficient deterministic-prover zero knowledge be achieved for $\mathcal{SZK}$ or for $\mathcal{NP}$.

Related work. As we have mentioned, the question of the necessity of randomness in zero knowledge was studied by [17], who proved negative results. Another line of work aimed at understanding the role of randomness in zero knowledge was initiated by [2], who introduced the notion of *resettable* zero knowledge. In this setting, the prover receives a single random tape, and reuses it in every proof. It was shown in [2] and considerable follow-up work that resettable zero knowledge can be achieved. Clearly, our setting is far more strict, since the prover has no randomness whatsoever.

Organisation. The full definitions that we use throughout appear in Appendix A.

2 Auxiliary-Input Honest-Verifier Zero Knowledge

The proof of impossibility for deterministic-prover zero knowledge of [17] utilizes a *malicious* verifier (who forwards its auxiliary input as verifier-messages). In this section, we consider the case of an honest verifier, which is not covered by that proof. We prove that it is indeed possible to achieve auxiliary-input deterministic-prover zero-knowledge for all languages in $\mathcal{NP}$ (and even all languages in $\mathcal{IP}$). Our proof system requires an unbounded prover.

The idea behind our proof system is very simple; we use the verifier to generate the randomness that the prover will use in the proof. Clearly, this cannot work in a naive way, since the zero-knowledge property requires the prover's randomness to be hidden from the verifier. However, this can be achieved by having the verifier send random strings, and the prover derive pseudorandom coins by inverting each string under a one-way function and taking the hard-core bit. This is reminiscent of the hidden bits paradigm for non-interactive zero knowledge [8] under the assumption of a one-way permutation $f : \{0,1\}^* \to \{0,1\}^*$. Our construction assumes the existence of doubly enhanced 1–1 one-way functions which include oblivious sampling properties, as in non-interactive zero knowledge [14]. We can transform the protocol of [8] to the plain model since we consider the case of an honest verifier, and so we can let the verifier generate the common reference string by itself. Our construction can be achieved under weaker assumptions than [8,14], that include RSA, factoring and discrete log. We note that we do not require a trapdoor function, since in our model the prover is not required to be efficient.

2.1 The Proof System

We begin by defining the hardness assumption needed for our construction; the existence of doubly enhanced 1–1 one-way functions.

Doubly-enhanced 1–1 one-way functions. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a 1–1 one-way function. Informally speaking, f is enhanced if there exists an algorithm S that samples values in the range that are hard to invert, even given the randomness used by the sampler S. We remark that the "canonical" way of sampling elements in the range – by choosing a random x and computing $y = f(x)$ – is not sufficient. This is due to the fact that the sampler's coins reveal the preimage itself. Goldreich introduce the notion of enhanced trapdoor permutations for oblivious transfer [13]; we use the same notion but require only 1–1 one-way functions. We note that enhanced 1–1 one-way functions can be built from standard assumptions (factoring, discrete log, etc.), as shown in [13].

As in the case of non-interactive zero knowledge, we need an additional enhancement, which is the ability to sample pairs of randomness used by the sampler together with a preimage; more exactly, we need to be able to generate random pairs (x_i, r_i) such that $f(x_i) = S(1^n; r_i)$. This is needed in our proof in order to prove a hybrid argument that many hard-core bits of f cannot be guessed, when given the randomness of the range-sampler. We now formally present the definition:

Definition 2.1. *A 1–1 one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is called a* doubly enhanced 1–1 one-way function *if the following conditions hold:*

1. Easy to sample elements in the range: *There exists a sampling algorithm S such that the output distribution of S on input 1^n is distributed almost identically to $f(U_n)$.*
2. Hard to invert given sampling randomness: *For every non-uniform probabilistic polynomial-time algorithm A, there exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,*

$$\Pr\left[A(S(1^n; U_n), U_n) = f^{-1}(S(1^n; U_n))\right] \leq \mu(n).$$

3. Easy to sample pairs of sampling randomness and preimgage: *There exists a probabilistic polynomial-time algorithm R that outputs a pair (r, x) such that*

$$\{R(1^n)\} \stackrel{s}{\equiv} \{(U_n, f^{-1}(S(1^n; U_n))\}.$$

As with any one-way function, an enhanced one-way function has a hard-core predicate. However, it must be hard to guess the bit even given the sampler randomness. Since it is hard to invert the one-way function in this case, such a hard-core predicate can be constructed as in [15,18].

Given the above enhancements, it is possible to prove that many hard-core bits are hard. We remark that proving that many hard-core bits are hard does not require the double enhancement; however, it is needed in order to prove that they are still hard given the sampler's randomness. The following lemma is proven via a standard hybrid argument.

Lemma 2.2. *Let f be a doubly-enhanced 1–1 one-way function and let B its hardcore predicate. Then, for every polynomial $p(\cdot)$,*

$$\left\{\left(U_n^{(1)}, B(f^{-1}(S(1^n;U_n^{(1)})))\right), \ldots, \left(U_n^{(p(n))}, B(f^{-1}(S(1^n;U_n^{(p(n))})))\right)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{U_{(n+1)\cdot p(n)}\right\}_{n\in\mathbb{N}}.$$

The fact that f must be *doubly* enhanced in order to prove the lemma is due to the fact that in the hybrid argument, the reduction must be able to generate pairs $\left(U_n^{(i)}, B(f^{-1}(S(1^n;U_n)))\right)$. However, computing this requires knowing both the randomness U_n used by S to sample a value in the range, as well as the preimage (since without the preimage, the hard-core bit of the preimage cannot be computed). This is exactly the definition of the double enhancement.

We are now ready to present the theorem:

Theorem 2.3. *Let L be a language with an honest-verifier efficient-prover zero-knowledge proof. If there exist doubly-enhanced 1–1 one-way functions, then there exists an auxiliary-input honest-verifier computational zero-knowledge proof system (P,V) for L, where P is deterministic (but not efficient). Furthermore, if L has a public-coin zero-knowledge proof, then (P,V) is public coin.*

Proof: Let f be a doubly-enhanced 1–1 one-way function and let B be its hardcore predicate (that exists, as shown in [15,18]. By the assumption in the theorem, the language L has an honest-verifier zero-knowledge proof; denote the proof system by (P',V'). Since P' is efficient, let $p(n)$ denote the maximum (polynomial) number of coins used by P' on input of length n. We construct the following proof system (P,V) for L:

PROTOCOL 2.4 (proof system (P,V) for L):

- **Input:** *common input $x \in \{0,1\}^n$ (and witness w for P)*
- **The protocol:**
  1. *V chooses a random string $r \in_R \{0,1\}^{n\cdot p(n)}$, and sends r to P.*
  2. *Denote the message achieved from V by $r = (r_1, \cdots, r_{p(n)})$, where each $r_i \in \{0,1\}^n$.*
  3. *For $i = 1, \cdots, p(n)$, prover P computes $y_i = S(1^n, r_i)$.*
  4. *For $i = 1, \cdots, p(n)$, prover P computes $x_i = f^{-1}(y_i)$ and $b_i = B(x_i)$.*
  5. *P runs P' with input x, witness w, and random tape $\overline{b} = (b_1, b_2, \cdots, b_{p(n)})$.*
  6. *V runs V' with input x and a uniform random tape.*

By inspection, Protocol 2.4 has a deterministic prover, and is public coin if (P',V') is public coin. Furthermore, completeness is immediate. Likewise, soundness holds since otherwise a cheating prover P^* for the proof system (P',V') could choose the string $r \in \{0,1\}^{n\cdot p(n)}$ itself and run P using the derived random coins. Thus, if a cheating prover can cause V to accept for $x \notin L$ with non-negligible probability, then it is also possible for V', in contradiction to the assumed soundness of the original protocol.

It remains to show that Protocol 2.4 is honest-verifier zero-knowledge. Intuitively, this follows from the fact that although V chose the coins $r \in \{0,1\}^{n\cdot p(n)}$

that determine the coins used by P in the proof (P', V'), by the double enhancement the hard-core bits define a string that is pseudorandom to V. Thus, V's is indistinguishable from V''s view in (P', V'), and the simulator for (P', V') can be used. The full proof of this statement appears in Appendix B. □

Since public-coin efficient-prover zero-knowledge proof systems exist for every language in $\mathcal{NP}$ as long as one-way functions exist, we conclude that:

Corollary 2.5. *If doubly-enhanced 1–1 one-way functions exist, then every language $L \in \mathcal{NP}$ has an honest-verifier public-coin deterministic-prover auxiliary-input zero-knowledge proof.*

2.2 An Extension to Collections of Dense 1–1 One-Way Functions

We proved Theorem 2.3 under the assumption that there exists a single 1–1 one-way function $f : \{0,1\}^* \to \{0,1\}^*$ that is doubly enhanced. However, it is not known how to construct such a one-way function under any standard assumption, to the best of our knowledge. Rather, in such situations (like when using enhanced trapdoor permutations), *collections* of one-way functions are used instead. However, this raises a challenge regarding who should choose the one-way function in the collection. In typical uses, like non-interactive zero knowledge, the *prover* chooses the function and proves to the verifier that it chose a valid function. However, here the prover is deterministic and so cannot do this. If we have the verifier naively choose the function, then it may learn a trapdoor, making it easy to invert (e.g., in the case that the 1–1 one-way function is instantiated via a family of trapdoor permutations). In this section, we show how to overcome this problem using families of *dense* 1–1 one-way functions. As a result, we show that our result can be achieved under the factoring, RSA and discrete log assumptions.

A collection of dense one-way functions (I, D, F) is a collection of one-way functions (see Definition A.5) with the additional property that the domain contains a *polynomial fractions* of the strings of the specific length. Formally, we require that there exists a positive polynomial $\ell(\cdot)$ such that for all large enough n's

$$\left|\overline{I} \cap \{0,1\}^n\right| \geq \frac{2^n}{\ell(n)},$$

where $\overline{I}$ is as in Definition A.5. When the collection is dense, then we can solve the aforementioned problem of how to choose the function without having a trapdoor that the verifier can use to invert, by having the verifier just choose a random string and hope that it defines a valid function. Clearly, this is not sufficient since we need to find a valid function with high probability. Thus, the verifier can choose *many* random strings, and the prover can just use the first valid one (since anyway, the prover is not efficient, it can verify if a string describes a valid function or not). As we will show, this strategy is sound, as a cheating prover cannot gain anything even by choosing an invalid function.

We begin by proving an extension to Theorem 2.3 for collections of dense functions, and then show to instantiate it under standard assumptions later. (To simplify the theorem statement, we present it as an extension of Corollary 2.5.)

Theorem 2.6. *If collections of dense doubly-enhanced 1–1 one-way functions exist, then every language $L \in \mathcal{NP}$ has an honest-verifier public-coin deterministic-prover auxiliary-input zero-knowledge proof.*

Proof: Let $L \in \mathcal{NP}$ be a language, let (I, D, F) be a collection of dense doubly-enhanced 1–1 one-way functions, and let $\ell(\cdot)$ be the polynomial such that $|\overline{I} \cap \{0,1\}^n| > \frac{2^n}{\ell(n)}$ for all large enough n's. The idea behind the protocol has been described above

PROTOCOL 2.7 (proof system (P, V) for L):

- **Input:** *common input $x \in \{0,1\}^n$ (and witness w for P)*
- **The protocol:**
  1. *V chooses random $\alpha_1, \ldots, \alpha_{n \cdot \ell(n)} \in_R \{0,1\}^n$, and sends them to P.*
  2. *P finds the first $1 \leq i \leq n \cdot \ell(n)$ such that $\alpha_i \in \overline{I} \cap \{0,1\}^n$, and aborts if there is no such i. (This can be carried out by P checking if $I(1^n; r) = \alpha_i$, for each random string r of the maximum length used by I.)*
  3. *P sends i to V.*
  4. *P and V run Protocol 2.4 for L, using the function f_{α_i} defined by α_i.*

By inspection, Protocol 2.7 has a deterministic prover, and is public coin if the proof of Protocol 2.4 is public coin (which exists by Corollary 2.5). Regarding completeness, the probability that a random $\alpha_i \in \{0,1\}^n$ is in $\overline{I} \cap \{0,1\}^n$ is at least $\frac{1}{\ell(n)}$. Thus, the probability that *none* of $\alpha_1, \ldots, \alpha_{n \cdot \ell(n)}$ are in $\overline{I} \cap \{0,1\}^n$ is at most $\left(1 - \frac{1}{\ell(n)}\right)^{n \cdot \ell(n)} < e^{-n}$, which is negligible.

Soundness holds for exactly the same reason as for Protocol 2.4, since a cheating prover can use any randomness it likes. We stress that this is the reason that we can have the prover choose i, and even if it chooses an invalid function, this makes no difference.

Finally, zero knowledge is proven exactly as in Protocol 2.4, since the proof there relies on the verifier not being able to distinguish the hard-core bits from random. Since the function used is guaranteed to be a valid one, it follows that it is hard to invert for V, and Lemma 2.2 holds. □

Instantiations under standard assumptions. In Appendix D we show that the result of Theorem 2.6 can be instantiated under the RSA, factoring and discrete log assumptions.

3 Efficient-Prover Auxiliary-Input Honest-Verifier SZK

In Section 2, we showed the existence of deterministic-prover *computational* zero knowledge. This begs the question as to whether deterministic-prover *statistical* zero knowledge exists. As we have discussed in the introduction, the protocol for the $\mathcal{SZK}$ complete language statistical difference (SD) actually has a deterministic prover [21]. Thus, it is already known that every language in $\mathcal{SZK}$ has an honest-verifier deterministic prover zero-knowledge proof system.

In this section, we ask whether or not it is possible to achieve deterministic-prover statistical zero-knowledge with an efficient prover. We already know that

every $L \in \mathcal{SZK} \cap \mathcal{NP}$ has a statistical zero-knowledge proof with an efficient, but not deterministic, prover [20]. Furthermore, we already know that every $L \in NP$ that has a statistically-sound witness encryption scheme is in $\mathcal{SZK}$ with a deterministic, but not efficient, prover [5]. The question that we address here is whether or not we can construct an *efficient and deterministic* prover for $\mathcal{SZK}$. We show that this can be achieved in some cases, and in particular, for every language with a hash proof system [4]. While our proof will be for languages with a hash proof system, the same result extends to every language L that has a witness encryption scheme. We remark that a similar construction was suggested by [7]. This work showed that for every NP-relation $R_L = \{(x, w)\}$ with a sampler S_{R_L} that has an *extractable* hash proof system it is possible to construct *predictable arguments of knowledge* for the related relation $R'_L = \{(x, r) \mid S_{R_L}(1^n; r) = (x, w)\}$ (where r is the randomness used to sample the instance $(x, w) \in R_L$). They used the construction as a sub-protocol to obtain a zero-knowledge predictable arguments of knowledge, but only in the *random oracle model*. We analyze this construction from a completely different point of view, we construct for every relation R that has a hash proof system (and not for R' as was done in [7]), a *deterministic* prover in the *plain model*. Note that for our protocol R is not required to be sampleable. Our construction yields a statistical zero-knowledge proof system in the plain model with an efficient prover for every relation R that has a hash proof system (similarly, witness encryption), without requiring it to be extractable.

Hash proof system. The notion of hash proof systems was introduced in [3,4] in order to construct practical CCA-secure encryption schemes under standard assumptions. The original definition in [4] refers only to hard subset membership problems, problems for which "hard instances" can be efficiently sampled. In our setting, we do not need to consider only hard instances, and so have a slightly relaxed definition.

The core property of a hash proof system is the notion of a projection function that enables the hash function to be computed in one of two ways. Informally speaking, let L be an NP-language. Then, a function from the collection of hash functions is determined by choosing a key k, and the function output $H_k(x)$ can be efficiently computed for *any* x. In addition, a hash function is projective if there exists a projection function $\alpha(\cdot)$ that maps keys k into their projections $s = \alpha(k)$. The interesting point about the projection key s is that it is also possible to compute $H_k(x)$ using the projection key s together with x and a witness w to the fact that $x \in L$. Furthermore, for any $x \notin L$, the value $H_k(x)$ is essentially uniformly distributed (this is called "smoothness"). All of the above together implies that the ability to compute $H_k(x)$ given s is a *proof* that $x \in L$. In particular, if $x \in L$ and a witness is known, then given s one can efficiently compute the proof $H_k(x)$. However, if $x \notin L$, then given s it is possible to guess $H_k(x)$ with only negligible probability. Observe that this proof can only be verified by a verifier who knows k and so can compute $H_k(x)$ for all x (in and not in the language). In the original definition by [4], the smoothness property

was only required for a random $x \in L$. Since we wish to use this for any $x \in L$, we require smoothness for any $x \in L$, as in the definition provided in [10].

Formally, let $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$, $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$ and $\mathcal{K} = \{K_n\}_{n\in\mathbb{N}}$ where each of X_n, G_n, K_n are finite, non-empty sets, and let $\mathcal{H} = \{H_n\}_{n\in\mathbb{N}}$ with $H_n = \{H_k\}_{k\in K_n}$ be a collection of hash functions from $\mathcal{X}$ to $\mathcal{G}$. We call $\mathcal{K}$ the key space of the family. Let L be a non-empty language. We define a key projection function $\alpha : \mathcal{K} \times \mathcal{X} \to \mathcal{S}$, where $\mathcal{S} = \{S_n\}_{n\in\mathbb{N}}$ is the space of key projections.

Definition 3.1. *The family $(\mathcal{H}, \mathcal{K}, \mathcal{X}, L, \mathcal{G}, \mathcal{S}, \alpha)$ is a* projective hash family *if for all $n \in \mathbb{N}$, $k \in K_n$ and $x \in L \cap \{0,1\}^n$, it holds that the value of $H_k(x)$ is uniquely determined by $\alpha(k, x)$ and x.*

Next, we define the notion of smoothness, meaning that $H_k(x)$ is *not* determined by $\alpha(k, x)$ and x, when $x \notin L$. (We do not actually need $H_k(x)$ to be almost uniform in G in this case, and it would suffice that it be hard to predict. However, we use the standard notion in any case.)

Definition 3.2. *Let $(\mathcal{H}, \mathcal{K}, \mathcal{X}, L, \mathcal{G}, \mathcal{S}, \alpha)$ be a projective hash family. Then, for every $x \in X \setminus L$ of length n, let $V(x, \alpha(k, x), H_k(x))$ be the following random variable: choose $k \in_R K_n$ and output $(x, \alpha(k, x), H_k(x))$. Similarly, define $V(x, \alpha(k, x), g)$ as follows: choose $k \in_R K_n$ and $g \in_R G_n$ and output $(x, \alpha(k, x), g)$. The projective hash family $(\mathcal{H}, \mathcal{K}, \mathcal{X}, L, \mathcal{G}, \mathcal{S}, \alpha)$ is* smooth *if for all $x \in X \setminus L$, $\{V(x, \alpha(k), H_k(x))\}_{x\in X\setminus L} \stackrel{s}{\equiv} \{V(x, \alpha(k), g)\}_{x\in X\setminus L}$.*

Finally, a hash proof system is a smooth projecting hash function that can be sampled and computed efficiently.

Definition 3.3. *Let L be an NP-language. Then, L has a* hash proof system *if there exists a smooth projective hash family $(\mathcal{H}, \mathcal{K}, \mathcal{X}, L, \mathcal{G}, \mathcal{S}, \alpha)$ such that the following algorithms exist:*

1. Key sampling: *a probabilistic polynomial-time algorithm that upon input 1^n samples $k \in K_n$ uniformly at random.*
2. Projection computation: *a deterministic polynomial-time algorithm that upon input 1^n, $k \in K_n$ and $x \in X \cap \{0,1\}^n$, outputs $s_x = \alpha(k, x)$.*
3. Efficient hashing from key: *a deterministic polynomial-time algorithm that upon input 1^n, $k \in K_n$ and $x \in X$, outputs $H_k(x)$.*
4. Efficient hashing from projection key and witness: *a deterministic polynomial time algorithm that upon input 1^n, $x \in L$, a witness w such that $(x, w) \in R$, and $s_x = \alpha(k, x)$ (for some $k \in K_n$), computes $H_k(x)$.*

We are now ready to prove our theorem:

Theorem 3.4. *Let L be an NP-language. If there exists a hash-proof system for L, then there exists an auxiliary-input honest-verifier statistical zero-knowledge where P is an efficient deterministic algorithm.*

Proof: Intuitively, a hash proof system can be used to generate an interactive proof by having the verifier sample a key $k \in K_n$ for a hash function, compute its projection $s_x = \alpha(k, x)$, and send s_x to the prover. Then, if the prover can return $H_k(x)$, the verifier is convinced that $x \in L$. This is honest-verifier zero knowledge since a simulator can simply sample k, compute s_x and simulate the prover sending $H_k(x)$. Furthermore, the prover is deterministic (since it just computes the hash function using the projection) and is efficient given the witness (by the definition of a hash proof system). We now prove this formally. $(\mathcal{H}, \mathcal{K}, \mathcal{X}, L, \mathcal{G}, \mathcal{S}, \alpha)$ be a hash proof system for L. We construct a proof system (P, V) for L, as follows:

PROTOCOL 3.5 (auxiliary-input HVSZK with an efficient prover):

- **Inputs:** *common input* $x \in \{0,1\}^n$ *(and witness* w *for* P*)*
- **The protocol:**
  1. *V samples a random $k \in_R K_n$, computes $s_x = \alpha(k, x)$ and sends s_x to P.*
  2. *P computes $H_k(x)$ using x, s_x and w, and sends $H_k(x)$ to V.*
  3. *V computes $H_k(x)$ and accepts iff it received y such that $y = H_k(x)$.*

It is clear that the prover in Protocol 3.5 is deterministic and efficient. It remains to prove that the proof system is auxiliary-input honest-verifier zero knowledge.

Completeness. If $x \in L$ then, there exists an NP witness w such that $(x, w) \in R_L$. Thus, the honest P can run the efficient hashing from projection key and witness with probability 1.

Soundness. For all n, all $x \notin L \cap \{0,1\}^n$, all (possibly unbounded) cheating provers P^*, it holds that: $\{V(x, \alpha(k), H_k(x))\}_{x \in X \setminus L} \stackrel{s}{\equiv} \{V(x, \alpha(k), g)\}_{x \in X \setminus L}$. Thus, for every cheating prover P^*, the probability that it returns $y = H_k(x)$ is at most negligibly greater than $1/|G_n|$.

Zero-knowledge. We construct a simulator $\mathcal{S}$ who upon input $x \in \{0,1\}^n$, chooses $k \in_R K_n$, and computes $\alpha(k, x)$ and $H_k(x)$. Then, $\mathcal{S}$ outputs the verifier's view to be k (or to be more exact, the randomness used to sample k) and $H_k(x)$ as the message received from the prover. Since $\alpha(k, x)$ uniquely determines $H_k(x)$ for any witness w when $x \in L$, the distribution generated by the simulator is identical to that of an honest prover. □

Instantiations. Hash proof systems are known to exist for languages defined by the decisional Diffie-Hellman problem, quadratic residuosity, and Paillier's decisional composite residuosity [4,10]. Thus, this demonstrates that efficient and deterministic-prover statistical zero-knowledge does exist for at least some non-trivial languages.

Remark. We can construct the same result from witness encryption scheme. The idea is that the verifier encrypts it own random tape with respect to the statement in a way that if the statement is true, the prover can decrypt the encryption and reveal the random tape, and if not there is no way for the prover to open the commitment. As we expected, decrypting the encryption and sending the message to the verifier will be the proof that the statement is true.

Corollary 3.6. *Let L be an NP-language. If L has a statistical witness encryption with PPT encryption, then there exists an auxiliary-input honest-verifier statistical zero-knowledge where P is an efficient deterministic algorithm.*

Corollary 3.7. *If L has a computational witness encryption with PPT encryption, then there exists an auxiliary-input honest-verifier zero-knowledge argument where P is an efficient deterministic algorithm.*

4 Plain ZK (Malicious and Honest Verifier)

4.1 Background

In this section, we ask whether it is possible to achieve deterministic-prover zero knowledge for *any* non-trivial language. Clearly, by the result of [17], this is not possible for auxiliary-input zero knowledge. However, it may be possible for *plain* zero knowledge. We show that it is indeed possible, and in particular, when the language is such that statements have two (or more) witnesses that are "computationally independent" of each other (meaning that it is not possible to compute one from the other). Such languages were used in [9] for constructing witness-hiding proofs from witness indistinguishable proofs. The idea behind the [9] construction is that if a proof that is a conjunction of two independent (hard) statements leaks the witness, then the witness that is leaked reveals which witness the prover used. This then contradicts witness indistinguishability. Our use of such languages is different, and is based on the observation that if the two witnesses are (computationally) independent, then one witness can be used to prove the proof and the other can be used as a basis for the prover's randomness. More specifically, if we consider a statement x with two witnesses w_1, w_2 that attest to $x \in L$, then the prover can run a standard zero-knowledge proof using w_1 with randomness that is obtained by applying a randomness extractor to w_2. This works as long as w_2 has high enough entropy to extract a pseudorandom string of the length needed for the proof (using a pseudorandom generator, it suffices to extract $O(n)$ bits). As such, w_1 can be used as the actual witness in the proof, and w_2 can be used to obtain the "randomness" used by the prover strategy in the standard zero-knowledge proof for the language.

We begin by defining the property needed from a relation in order for our proof to work.

Definition 4.1. *Let R_L be a relation, and let $L = \{x \mid \exists w \ s.t \ (x, w) \in R_L\}$ be an NP language. We say that L is a* hard dual-witness language *if the following conditions hold:*

– Easy to sample statements with two witnesses: *There exists a probabilistic polynomial-time sampler S_L that on input 1^n outputs a tuple (x, w_1, w_2) where $x \in \{0,1\}^n$ and $w_1 \neq w_2$, such that $(x, w_1) \in R_L$ and $(x, w_2) \in R_L$.*
– Witness intractability: *For every PPT algorithm $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that:*

$$\left| \Pr[(S_L^x(1^n), \mathcal{A}(S_L^x(1^n), 1^n)) \in R_L] \right| \leq \mu(n),$$

– Second witness intractability: *For every PPT algorithm $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that:*

$$\left| \Pr[\mathcal{A}(S_L^1(1^n), 1^n) = S_L^2(1^n)] \right| \leq \mu(n),$$

where $S_L^x(1^n)$ denotes the first element x of the triple, $S_L^1(1^n)$ denotes the first two elements (x, w_1) of the triple, and $S_L^2(1^n)$ denotes the third element w_2.

Remark. Definition 4.1 rules out some types of trivial relations. For example, for any NP relation R, we can consider a new relation $R' = \{(x, w||z) \mid (x, w) \in R, z \in \{0,1\}^{\ell(n)}\}$ (where $\ell(\cdot)$ is the polynomial bound on the prover's random coin tosses). Consider a random instance $(x, w||z)$ where z is random and independent from w, the prover can use z as its random tape. This clearly trivializes the problem, but, we note that this relation R' *does not* meet Definition 4.1, because R' is not a *hard* relation. This is because any string z can be appended to a valid witness of R to create a valid witness of R', in contradiction to the second witness intractability property. Definition 4.1 does not allow such trivial relations.

Definition 4.1 requires only that it is hard to find w_2 given (x, w_1). However, we actually require that the computational entropy of w_2 given (x, w_1) be high enough to extract a polynomial amount of pseudorandomness. We formalize this in the following definition:

Definition 4.2. *Let L be a hard dual-witness language, let S_L be the sampler, and let $\ell(n)$ be a function. We say that L has $\ell(n)$-*extractable entropy *if there exists an extractor function $\mathsf{Ext} : L \to \{0,1\}^*$ such that*

$$\left\{ \left(S_L^1(1^n), \mathsf{Ext}(S_L^2(1^n)) \right) \right\} \stackrel{c}{\equiv} \left\{ \left(S_L^1(1^n), U_{\ell(n)} \right) \right\}.$$

We say that L has high extractable entropy *if $\ell(n) = \omega(\log n)$.*

Before proceeding, we give some concrete examples of relations that fulfill Definitions 4.1 and 4.2:

1. *OR of two hard languages:* In [9], witness hiding proofs were constructed for languages that are defined as the OR of two hard statements. For example, let f be a one-way function. Then, one can define

$$L = \left\{ (x_1, x_2) \mid \exists w : f(w) = x_1 \ \vee \ f(w) = x_2 \right\}.$$

Since f is a one-way function, a sampler S_L can easily be defined by choosing independent random $w_1, w_2 \in \{0,1\}^n$ and outputting the statement $x = (x_1, x_2)$ and witnesses w_1, w_2. The fact that it is hard to compute the second witness from the first follows easily from the fact that it is hard to invert the one-way function (and that w_1, w_2 are independent). In order to achieve Definition 4.2, L can be repeated $\ell(n)$ times for $\ell(n) = \omega(\log n)$, and hard-core bits can be taken from each instance. This can be generalized to arbitrary hard languages for which it is hard to compute the witness from the statement in the natural way.

2. *Diffie-Hellman tuples:* In some natural cases, it is not necessary to take the OR of two statements; rather, a single statement has two witnesses. Let $\mathbb{G}$ be a group of order q with generator g and assume that the Diffie-Hellman assumption holds in the group (formally, assume a generator algorithm that upon input 1^n outputs a description $(\mathbb{G}, g, q)$ with $|q| = n$, for which the Diffie-Hellman assumption holds). From here on, we omit the description of the group, for conciseness. Consider the language

$$L = \left\{ (h_1, h_2, h_3) \mid \exists a, b \in \mathbb{Z}_q : (h_1, h_2, h_3) = (g^a, g^b, g^{ab}) \right\}$$

and observe that a by itself and b by itself are witnesses that suffice for determining that $(h_1, h_2, h_3) \in L$. This is due to the fact that given a one can verify that $h_1 = g^a$ and $h_3 = {h_2}^a$. Likewise, given b one can verify that $h_2 = g^b$ and $h_3 = {h_1}^b$. Next, consider a sampler who generates a group description $(\mathbb{G}, g, q)$, chooses $a, b \in \mathbb{Z}_q$ at random, and outputs $\big((g^a, g^b, g^{ab}), a, b\big)$. We argue that given (g^a, g^b, g^{ab}) and a, it is hard to compute b. This follows immediately from the discrete log assumption (otherwise, given g^b, compute g^a and g^{ab} and use the triple to obtain b). As in the previous example, this can be used to achieve Definition 4.2 by taking $\ell(n)$ statements and hard-core bits.

   Observe that this type of language arises quite naturally, in proving that the key generated from a Diffie-Hellman key exchange is a certain value, without revealing the private exponents a and b (which may be important if they may be reused over multiple exchanges).

3. *Public-key encryption ciphertext:* Another example of a natural statement that has two witnesses that fulfill Definition 4.1 can be obtained from public-key encryption. In particular, consider the language

$$L = \left\{ (pk, m, c) \mid \exists r, sk : c \leftarrow Enc_{pk}(m; r), m \leftarrow Dec_{sk}(c) \right\}.$$

   Once again, this language has two witnesses, each of which suffice by itself to validate that $(pk, m, c) \in L$. Specifically, given r one can verify that $c = E_{pk}(m; r)$ and given sk one can verify that $Dec_{sk}(c) = m$. Furthermore, it is clear that given r it is hard to compute sk (otherwise, one can break the security of the encryption scheme by encrypting $c = E_{pk}(m; r)$ and using this information to find sk). Regarding Definition 4.2, one can use the same method as above of repetition, or can prove the existence of an extractor function directly.

Observe that this type of language arises quite naturally, in proving that a given ciphertext indeed encrypts a given message, without revealing the public key.

The above demonstrates that the requirements of Definitions 4.1 and 4.2 can be achieved, and even exist in many natural settings.

We remark that it is *not* possible to fulfil Definitions 4.1 and 4.2 by taking the second witness w_2 to be an independent random string. This is because Definition 4.1 mandates that both $(x, w_1) \in R$ and $(x, w_2) \in R$. Thus, a random independent string cannot be taken as the second witness, and the definition cannot be achieved trivially.

4.2 Plain ZK For Dual Hard Languages

We note that even hard languages may have an easy sampler, and deterministic-prover zero knowledge cannot hold for such generation. Therefore, we suggest a new definition is called "plain zero knowledge for dual hard instances". The definition requires zero knowledge to hold only with respect to samplers that fulfill the definition of hardness as in Definition 4.1, instead of every sampler as in Definition A.4, or random samplers as in [11].

More formally, we define $\langle P, V^* \rangle(S_L(1^n))$ to be the process of running $S_L(1^n)$ to receive (x, w_1, w_2) and then outputting the output of V^* after running the zero knowledge proof with P on common input x, where P has auxiliary input (w_1, w_2). Define $\mathcal{S}(S_L^x(1^n))$ to be the result of running $S_L(1^n)$ to get (x, w_1, w_2) and then outputting $\mathcal{S}(x)$. We also define $\langle P, V^* \rangle(S_L^1(1^n))$ to be the process of running $S_L(1^n)$ to receive (x, w_1) and then and then outputting the output of V^* after running with P on common input x, where P has auxiliary input w_1.

Definition 4.3 (plain zero-knowledge for dual hard instances). *Let (P, V) be an interactive proof system for a hard dual-witness language L. We say that (P, V), is statistical/computational* uniform plain zero-knowledge for dual hard instances *if for every probabilistic polynomial-time V^* there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that the following two ensembles are uniform computationally indistinguishable:*

- $\{\langle P, V^* \rangle(S_L(1^n))\}_{n \in \mathbb{N}}$
- $\{\mathcal{S}(S_L^x(1^n))\}_{n \in \mathbb{N}}$

where S_L is a sampler as in Definition 4.1

4.3 The Construction

We are now ready to prove that deterministic-prover plain zero knowledge for dual hard instances exists for such languages.

Theorem 4.4. *Let L be a hard dual-witness language with high extractable entropy with parameter $\ell(n)$, and assume the existence of a pseudorandom generator $G : \{0,1\}^{\ell(n)} \to \{0,1\}^{p(n)}$ for non-uniform distinguishers, where $p(n)$ is*

the number of random bits needed by the prover in the proof of [16] *for* L*. Then there exists a deterministic prover computational plain zero-knowledge for dual hard instances proof system for* L*. Furthermore, given both witnesses, the proof has an efficient prover.*

Proof: Let (P_L, V_L) be the computational zero-knowledge proof system for L known to exist by [16] (the assumptions in the theorem imply the existence of one-way functions that suffice for [16]), and let $p(n)$ be the number of coins used by the prover in (P_L, V_L) on input $x \in \{0,1\}^n$. We construct a deterministic-prover proof (P, V) for L as follows:

PROTOCOL 4.5 (computational plain zero-knowledge for L):

- **Input:** *common input* $x \in \{0,1\}^n$ *(and two witnesses* w_1, w_2 *for* P*)*
- **The protocol:**
  1. P *computes* $s = \mathsf{Ext}(w_2) \in \{0,1\}^{\ell(n)}$.
  2. P *computes* $r = G(s) \in \{0,1\}^{p(n)}$.
  3. P *and* V *run the zero-knowledge proof* (P_L, V_L)*, where* P *runs* P_L *on input* (x, w_1) *with random tape* r*, and* V *runs* V_L *on input* x *and a uniform random tape.*

Intuitively, Protocol 4.5 is zero knowledge since r is indistinguishable from an independently chosen random tape. We formally prove this in Appendix C. □

Plain versus auxiliary-input zero knowledge. The fact that Theorem 4.4 holds only for plain zero knowledge can be seen as follows. Consider the setting of auxiliary input, and where the distinguisher is given the witnesses w_1, w_2. In this case, the distinguisher can run $P(x, w_1, w_2)$ itself and compare that the result is *exactly* what it received. The probability that a simulator can generate this view is negligible, due to the fact that it w_2 has high extractable entropy.

Technically, in the proof of Theorem 4.4, observe that we use the fact that L has high extractable entropy, meaning that

$$\left\{\left(S_L^1(1^n), \mathsf{Ext}(S_L^2(1^n))\right)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\left(S_L^1(1^n), U_{\ell(n)}\right)\right\}_{n\in\mathbb{N}}.$$

However, such indistinguishability can only be assumed for uniform distinguishers. In particular, in the case of non-uniform distinguishers or auxiliary input, a distinguisher who is given w_2 as well as (x, w_1) can always distinguish the distributions with probability 1.

Single witness languages and circularity. An interesting question that arises in this context is whether it is possible to achieve a similar construction when there is only a single witness of high entropy. The problem that arises in this context is that of circularity since the same witness is used to run the prover strategy in the proof and to derive its randomness. It may be possible to therefore construct deterministic-prover zero knowledge under circular-secure assumptions, but it seems unlikely that this is possible under standard assumptions that do not imply security in this type of circular setting.

4.4 A Generic Instantiation using Strong Extractors

Our above construction assumes the existence of an appropriate extractor function Ext without specifying what it should be. For one-way functions it could be a hard-core predicate, for example, as we described above. In this section, we give a more generic instantiation that uses *strong extractors* and works as long as the min-pseudoentropy of the witness is high enough. However, since strong extractors need a random seed, this must be chosen by the verifier, and thus the construction in this section works only for honest verifiers. As such, the results here provide an alternative to those of Section 2 that assume the existence of a doubly-enhanced 1–1 one-way function. Thus, for languages of the type we consider here, one-way functions alone suffice, and we do not need the addition assumption of the double enhancement.

We begin by formally defining pseudoentropy and strong extractors.

Definition 4.6. *A random variable X has* pseudoentropy *at least k if there exists a random variable Y such that: X is computationally indistinguishable from Y, and $H(Y) \geq k$, where $H(\cdot)$ denotes Shannon entropy.*

In our constructions in this section, we assume that the distribution $\left\{S_L^2(1^n)\right\}_{n\in\mathbb{N}}$ has min-pseudoentropy of $\omega(\log(n))$.

Definition 4.7. *A* strong extractor family *(or simply strong extractor) is an infinite family $E = \{E_n\}$, indexed by a parameter n, of the form $E_n : \{0,1\}^{h(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^{\ell(n)}$ where the functions $h(n), m(n), \ell(n)$, are all polynomial in n. The extractor family E is called $(k(n), \epsilon(n))$-*statistical *if for any probability ensemble X with support in $\{0,1\}^{h(n)}$ and min-entropy $k(n)$, it holds that:*

$$\mathsf{StatDiff}(\langle U_{\ell(n)}, E_n(X_n, U_{m(n)})\rangle, U_{m(n)+\ell(n)}) \leq \epsilon(n)$$

The construction. Clearly, we cannot expect to obtain true statistical closeness between the extracted value from w_2 and a random string, since w_2 has no real entropy (in particular, an unbounded distinguisher can find w_2 and compute the extractor on the value). Thus, we have only computational entropy, and so want to achieve computational indistinguishablity. We achieve this by having the verifier choose the seed for the extractor, and the prover applying it to w_2 as the *source*. We therefore use an extractor, according to the following definition:

Definition 4.8. *A family $E = \{E_n\}$ of strong extractors is called $k(n)$-*computational *if E_n is polynomial-time computable, and for all efficiently-samplable probability ensembles X with min-entropy $k(n)$, the joint distribution $\langle U_{\ell(n)}, E_n(X_n, U_{m(n)})\rangle$ is computationally indistinguishable from $U_{m(n)+\ell(n)}$.*

We are now ready to state the theorem:

Theorem 4.9. *Let L be a hard dual-witness language. Assume that there exists a strong extractor $E_n : \{0,1\}^{h(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^{\ell(n)}$ for a polynomial $\ell(n) = \omega(\log(n))$ and a pseudorandom generator $G : \{0,1\}^{\ell(n)} \to \{0,1\}^{p(n)}$ for*

non-uniform distinguishers, where $p(n)$ is the number of random bits needed by the prover in the proof of [16] for L. Then there exists a deterministic prover computational plain zero-knowledge proof system for L. Furthermore, given both witnesses, the proof has an efficient prover.

Proof: Let (P_L, V_L) be the computational zero-knowledge proof system for L known to exist by [16], and let $p(n)$ be the number of coins used by the prover in (P_L, V_L). We construct a deterministic-prover proof (P, V) as follows:

PROTOCOL 4.10 (honest-verifier computational plain zero-knowledge):

- **Input:** *common input $x \in \{0,1\}^n$ (and two witnesses w_1, w_2 for P)*
- **The protocol:**
  1. *V choose a random $s' \in_R \{0,1\}^{m(n)}$ and sends s' to P.*
  2. *P computes $s = E_n(w_2, s') \in \{0,1\}^{\ell(n)}$.*
  3. *P computes $r = G(s) \in \{0,1\}^{p(n)}$.*
  4. *P and V run zero-knowledge proof (P_L, V_L), where P runs P_L on (x, w_1) with random tape r, and V run V_L on input x.*

The proof of this protocol is as in the proof of Theorem 4.4. □

5 Sequential Composition

An interesting question that arises in the setting of deterministic-prover zero knowledge relates to sequential composition. The fact that sequential composition holds for *auxiliary-input* zero knowledge was proved in [17]. Since our results in Sections 2 and 3 are auxiliary-input zero knowledge, it immediately follows that:

> *There exist deterministic-prover zero knowledge proofs (with an honest verifier) that remain zero knowledge under sequential composition.*

In contrast, it is clear that Protocol 4.5 does *not* preserve zero knowledge under sequential composition. This is due to the fact that the verifier can send different queries in each execution and obtain the witness, in the same way that the knowledge extractor for [16] works (using the fact that it is a proof of knowledge).

Acknowledgments

We thank Iftach Haitner for significant helpful discussions.

References

1. B. Barak, Y. Lindell and S. Vadhan. Lower Bounds for Non-Black-Box Zero Knowledge. In *Journal of Computer and System Sciences*, 72(2):321–391, 2006.

2. R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable Zero-Knowledge. In the 32*nd STOC*, pages 235–244, 2000.
3. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO 1998*, Springer (LNCS 1462), pages 13–25, 1998.
4. R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *EUROCRYPT 2002*, Springer (LNCS 2332), pages 45–64, 2002.
5. S. Garg, C. Gentry, A. Sahai and B. Waters. Witness Encryption and its Applications. In *STOC,* pages 467–476, 2013.
6. S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.
7. A. Faonio, J.B. Nielsen and D. Venturi. Predictable Arguments of Knowledge In *Public-Key Cryptography,* Springer (LNCS 10174), pages: 121–150 ,2017.
8. U. Feige, D. Lapidot and A. Shamir. Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract). In the 31*st FOCS*, pages 308–317, 1990.
9. U. Feige and A. Shamir. Witness Indistinguishable and Witness Hiding Protocols. In the 22*nd STOC*, pages 416–426, 1990.
10. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. In *EUROCRYPT 2003*, Springer (LNCS 2656), pages 524–543, 2003.
11. O. Goldreich. A Uniform-Complexity Treatment of Encryption and Zero-Knowledge. In *Journal of Cryptology*, 6(1):21–53, 1993.
12. O. Goldreich. *Foundations of Cryptography: Volume 1 - Basic Tools.* Cambridge University Press, 2001.
13. O. Goldreich. *Foundations of cryptography: Volume 2 - Basic Applications.* Cambridge University Press, 2004.
14. O. Goldreich. Basing Non-Interactive Zero-Knowledge on (Enhanced) Trapdoor Permutations: The State of the art. *Studies in Complexity and Cryptography*, Springer, pages 406–421, 2011.
15. O. Goldreich and L. Levin. A Hard-Core Predicate for All One-Way Functions. In the 21*st STOC*, pages 25–32, 1989.
16. O. Goldreich S. Micali and A. Wigderson. Proofs That Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. In the *Journal of the ACM*, 38(1):691–729, 1991. (Extended abstract in *FOCS 1986.*)
17. O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. In *Journal of Cryptology*, 7(1):1–32, 1994.
18. O. Goldreich and R. Rothblum. Enhancements of Trapdoor Permutations. In *Journal of Cryptology*, 26(3):484–512, 2013.
19. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In *SIAM Journal on Computing*, 18(1):186–208, 1989. (Extended abstract in *STOC 1985.*)
20. M.H. Nguyen and S. Vadhan. Zero Knowledge with Efficient Provers. In 38*th STOC*, pages 287–295, 2006.
21. S. Vadhan. *A Study of Statistical Zero-Knowledge Proofs.* PhD thesis, Massachusetts Institute of Technology, August 1999.

A Definitions and Preliminaries

A.1 Preliminaries

A function $\mu(\cdot)$ is negligible in n, or just negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n's it holds that $\mu(n) < 1/p(n)$. We denote the length of the statement x to the zero-knowledge proof by n. All parties are assumed to run in time that is polynomial in the length of the statement (in particular, if they also receive auxiliary input, then their running time still depends only on the length of the statement, and not on the length of the auxiliary input). We denote by $A(x;r)$ the deterministic output of algorithm A on input x and the random string r, and denote by $t_M(\cdot)$ the running time of machine M. We sometimes use PPT as shorthand for probabilistic polynomial time. A non-uniform probabilistic polynomial time machine is a pair $(M, \overline{a})$, where M is a two-input polynomial-time Turing machine and $\overline{a} = a_1, a_2, \cdots$ is an infinite sequence of strings. For every x, we the computation of machine M is with the input pair $(x, a_{|x|})$; as stated above, the running time of the machine depends only on the length of the first input x. An interactive protocol (M_1, M_2) consists of two machines that compute the next-message function of the machines in the protocol. We denote by $M_1(x, a, \alpha_1, \cdots, \alpha_k; r)$ the next message α_{k+1} sent by machine M_1 on common input x, auxiliary-input a, random tape r, and messages received $\alpha_1, \cdots \alpha_k$.

In this work, we consider uniform and non-uniform computational indistinguishability, defined as follows:

Definition A.1. *Two distribution ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* non-uniform computationally indistinguishable *if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,*

$$\Big|\Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1]\Big| \leq \mu(n)$$

If the above holds for every uniform probabilistic polynomial-time algorithm D, then X and Y are said to be uniform computationally indistinguishable.

We denote computational indistinguishability by $X \stackrel{\text{c}}{\equiv} Y$, with the distinction between uniform and non-uniform being by context. We also use statistical closeness:

Definition A.2. *Two distribution ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* statistically close*, denoted $X \stackrel{s}{\equiv} Y$, if there exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,*

$$\mathsf{StatDiff}(X, Y) \stackrel{\text{def}}{=} \max_{S\subseteq\{0,1\}^*} \Big|\Pr[X \in S] - \Pr[Y \in S]\Big| \leq \mu(n)$$

A.2 Zero Knowledge Proof Systems – Definitions

Let L be an NP language and let R_L be its associated relation. For any x, define the witness set $W_L(x)$ to be the set of all w such that $(x, w) \in R_L$. We denote by $\langle P(w, r_P), V(z, r_V)\rangle(x)$ the view of a PPT V after the interaction with P on joint input x, where P has additional input w and random tape r_P, and V has additional input z and random tape r_V. We denote by $\langle P(w), V(z)\rangle(x)$ the associated random variable when r_P and r_V are chosen uniformly.

Definition A.3 (auxiliary-input zero knowledge). *Let (P, V) be an interactive proof system for a language L. We say that (P, V), is statistical/computational* auxiliary-input zero knowledge *if for every probabilistic polynomial-time V^* there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that the following two ensembles are statistically/computationally indistinguishable:*

- $\{\langle P(w), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ *for arbitrary* $w \in W_L(x)$
- $\{\mathcal{S}(x, z)\}_{x\in L, z\in\{0,1\}^*}$

We say that the proof system is honest verifier auxiliary-input zero-knowledge *if the above holds for the specified V.*

We remark that in the case of an honest verifier, it suffices to give the auxiliary input to the distinguisher, since the verifier does not use it in any case.

We also note that in the case of auxiliary-input zero-knowledge, it suffices to consider *uniform computational indistinguishability* (with uniform distinguishers), as this is equivalent to non-uniform distinguishers [12].

Plain zero-knowledge. The original definition of zero knowledge did not consider auxiliary inputs. Although auxiliary inputs are crucial in many applications (and to obtain sequential composition), questions of feasibility regarding zero knowledge with no auxiliary input are of theoretical interest. We call this original definition *plain zero knowledge*. There are two ways to define plain zero knowledge; in the first variant the distinguisher is given the witness [12], while in the second variant the distinguisher is not given the witness [1]. Although the latter notion is weaker, we argue that it is a natural definition, since the simulation paradigm guarantees formalizes the notion that V^* could generate an indistinguishable view itself. Since V^* does not have the witness, indistinguishability relative to distinguishers who don't have the witness is natural. In addition, our result in Section 4 relies on this fact crucially. We define two variants of plain zero-knowledge, one with uniform verifiers and distinguishers, and the other with non-uniform machines.

Definition A.4 (plain zero-knowledge). *Let (P, V) be an interactive proof system for a language L. We say that (P, V), is statistical/computational* uniform plain zero-knowledge *if for every probabilistic polynomial-time V^* there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that the following two ensembles are uniform computationally indistinguishable:*

– $\{\langle P, V^* \rangle(x)\}_{x \in L}$
– $\{\mathcal{S}(x)\}_{x \in L}$

We say that the proof system is non-uniform plain zero-knowledge *if the above holds for every non-uniform probabilistic polynomial-time V^* and with non-uniform computational indistinguishability. If the above holds with statistical closeness then it is* plain statistical zero-knowledge*, and if only for the specified verifier then it is* honest-verifier plain zero-knowledge*.*

We remark that our definition is different to the standard definition of uniform zero-knowledge by Goldreich [11], since we quantify over every $x \in L$ and do not require uniform generation of the inputs. The reason for this is that the aim of [11] was to consider a fully uniform treatment of zero knowledge, including the use of uniform hardness assumptions only. In contrast, our aim in this relaxation is to achieve deterministic-prover zero knowledge, and we do not mind using non-uniform hardness assumptions. Thus, it suffices for us to use this simpler formulation.

A.3 Collections of One-Way Functions

For the sake of completeness, we recall the definition of families (or collections) of functions.

Definition A.5. *A collection of functions consists of an infinite set of indices $\overline{I}$, a corresponding set of functions $\{f_i\}_{i \in \overline{I}}$, and a set of finite domains $\{D_i\}_{i \in \overline{I}}$, where the domain of f_i is D_i. A collection of functions $(\overline{I}, \{f_i\}, \{D_i\})$ is called* one-way *if there exist three probabilistic polynomial-time algorithms I, D and F such that the following conditions hold:*

1. Easy to sample and compute: *The output distribution of algorithm I on input 1^n is a random variable assigned values in the set $\overline{I} \cap \{0,1\}^n$. The output distribution of algorithm D on input $i \in I$ is a random variable assigned values in the set D_i. On input $i \in I$ and $x \in D_i$, algorithm F always outputs $f_i(x)$; i.e., $F(i,x) = f_i(x)$.*
2. Hard to invert: *For every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Pr[A'(I_n, f_{I_n}(X_n)) \in f_{I_n}^{-1}(f_{I_n}(X_n))] < \frac{1}{p(n)}$$

where I_n is a random variable denoting the output distribution of $I(1^n)$ and X_n is a random variable denoting the output distribution of D on input (random variable) I_n. We denote a collection of one-way functions by its algorithms (I, D, F).

B Full Proof of Zero-Knowledge for Protocol 2.4

Theorem B.1. *Protocol 2.4, denoted (P,V), is honest-verifier zero knowledge.*

Proof: We begin by constructing a "hybrid" proof system (P_1,V_1) that is the same as (P,V) except that the random tape $\overline{b}$ used to run P is chosen independently of the string r sent by V.

Proof system (P_1,V_1) for L:

- **Input:** common input $x \in \{0,1\}^n$ (and witness w for P)
- **The protocol:**
  1. V chooses a random string $r \in_R \{0,1\}^{n\cdot p(n)}$, and sends r to P.
  2. Denote the message achieved from V by $r = (r_1, \cdots, r_{p(n)})$, where each $r_i \in \{0,1\}^n$.
  3. P chooses a random string $\overline{b} \in \{0,1\}^{p(n)}$.
  4. P runs P' with input x, witness w, and random tape $\overline{b}$.
  5. V runs V' with input x and a uniform random tape.

We claim that

$$\left\{\langle P_1,V_1\rangle(x,z)\right\}_{x\in L; z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\langle P,V\rangle(x,z)\right\}_{x\in L; z\in\{0,1\}^*}. \qquad (1)$$

This follows from the indistinguishability of hard-core bits from random, as in Lemma 2.2. Formally, assume by contradiction that there exists a probabilistic polynomial-time distinguisher D and a polynomial $q(\cdot)$ such that for infinitely many x's,

$$\left|\Pr\left[D(\langle P_1,V_1\rangle(x,z),x,z)=1\right] - \Pr\left[D(\langle P,V\rangle(x,z),x,z)=1\right]\right| > \frac{1}{q(|x|)}.$$

We construct a non-uniform polynomial-time distinguisher D' that distinguishes $\left(U_n^{(1)}, B(f^{-1}(S(1^n;U_n^{(1)})))\right), \ldots, \left(U_n^{(p(n))}, B(f^{-1}(S(1^n;U_n^{(p(n))})))\right)$ from $U_{(n+1)\cdot p(n)}$ with at least the same probability, as follows. Upon receiving an input string r of length $(n+1)\cdot p(n)$ and advice (x,w) with $|x|=n$, D' works as follows:

1. D' parses r into $(r_1,b_1),\ldots,(r_n,b_{p(n)})$ and sets $\overline{b}=(b_1,\ldots,b_{p(n)})$.
2. D' computes $\langle P'((x_n,w_n);\overline{b}),V'\rangle(x_n)$ and obtains the resulting verifier's view. (We stress that P' is run using randomness $\overline{b}$, and V' is run with a uniform random tape.)
3. D' invokes D on the "view" including $r_1,\ldots,r_{p(n)}$:

$$\left((r_1,\ldots,r_{p(n)}),\langle P'((x_n,w_n);\overline{b}),V'\rangle(x_n)\right)$$

and returns whatever D outputs.

Clearly, when D' receives input

$$\left(U_n^{(1)}, B(f^{-1}(S(1^n; U_n^{(1)})))\right), \ldots, \left(U_n^{(p(n))}, B(f^{-1}(S(1^n; U_n^{(p(n))})))\right),$$

the view generated is *exactly* that of (P, V), since P''s coins $\overline{b}$ are the hard-core bits from the preimages of $r_1, \ldots, r_n$. In contrast, when D' receives input $U_{(n+1)\cdot p(n)}$, the view generated is *exactly* that of (P_1, V_1), since P''s coins $\overline{b}$ are random and independent of $r_1, \ldots, r_n$. Thus, for infinitely many n's, D' distinguishes a series of hard-core predicates from random with probability at least $1/q(n)$, in contradiction to Lemma 2.2.

Next, we prove that (P_1, V_1) is auxiliary-input honest-verifier zero-knowledge. This follows from the fact that (P', V') is auxiliary-input honest-verifier zero-knowledge, and so has a simulator $\mathcal{S}'$. In particular, the only difference between (P_1, V_1) and (P', V') is that V_1 sends a uniform random string r at the onset of the protocol. However, this string is not used for anything in the sequel. Thus, we can construct a simulator $\mathcal{S}_1$ for (P_1, V_1) who chooses a random $r \in \{0,1\}^{n \cdot p(n)}$ and outputs r followed by $\mathcal{S}'(x)$. By the assumption that $\mathcal{S}'$ is a good simulator for (P', V') we have that

$$\left\{\mathcal{S}'(x, z)\right\}_{x \in L; z \in \{0,1\}^*} \stackrel{c}{\equiv} \left\{\langle P'(w), V'(z)\rangle(x)\right\}_{x \in L; z \in \{0,1\}^*}$$

and so

$$\left\{\mathcal{S}_1(x, z)\right\} \equiv \left\{U_{n \cdot p(n)}, \mathcal{S}'(x, z)\right\} \stackrel{c}{\equiv} \left\{U_{n \cdot p(n)}, \langle P'(w), V'(z)\rangle(x)\right\} \equiv \left\{\langle P_1(w), V_1(z)\rangle(x)\right\}.$$

Combining the above with Eq. (1), we conclude that

$$\left\{\mathcal{S}_1(x, z)\right\}_{x \in L; z \in \{0,1\}^*} \stackrel{c}{\equiv} \left\{\langle P(w), V(z)\rangle(x)\right\}_{x \in L; z \in \{0,1\}^*},$$

completing the proof. □

C Full Proof of Zero-Knowledge for Protocol 4.5

Theorem C.1. *Protocol 4.5, denoted (P, V), is honest-verifier zero knowledge.*

Proof: We define two "hybrid" protocols (P', V') and (P'', V''):

Protocol (P', V'):

- **Input:** common input $x \in \{0,1\}^n$ (and witness w_1 for P)
- **The protocol:**
  1. P' chooses a random $s \in \{0,1\}^{\ell(n)}$.
  2. P' computes $r = G(s) \in \{0,1\}^{p(n)}$.
  3. P' and V' run the zero-knowledge proof (P_L, V_L), where P' runs P_L on input (x, w_1) with random tape r, and V' runs V_L on input x and a uniform random tape.

Protocol (P'', V'')**:**

- **Input:** common input $x \in \{0,1\}^n$ (and witness w_1 for P)
- **The protocol:**
  1. P'' chooses a random $r \in \{0,1\}^{p(n)}$.
  2. P'' and V'' run the zero-knowledge proof (P_L, V_L), where P'' runs P_L on input (x, w_1) with random tape r, and V'' runs V_L on input x and a uniform random tape.

We begin by proving that the output distributions of (P, V) and (P', V') are computationally indistinguishable; formally,

$$\left\{\langle P, V\rangle(S_L(1^n))\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\langle P', V'\rangle(S_L^1(1^n))\right\}_{n\in\mathbb{N}}.$$

Assume by contradiction that there exists a probabilistic polynomial-time algorithm D and a polynomial $p(\cdot)$ such that for infinitely many n's, algorithm D distinguishes $\langle P, V\rangle(S_L(1^n))$ from $\langle P', V'\rangle(S_L^1(1^n))$ with probability at least $\frac{1}{p(n)}$. We use D to construct D' that distinguishes $(S_L^1(1^n), \mathsf{Ext}(S_L^2(1^n)))$ from $(S_L^1(1^n), U_{\ell(n)})$ with at least the same probability.

Upon receiving inputs $(S_L^1(1^n), s)$, D' computes $r = G(s)$ and generates the output of the interaction between prover $P_L(S_L^1(1^n))$ with random-tape r and verifier $V_L(S_L(1^n))$. Next D' runs D on the output of the interaction and outputs whatever D outputs. Clearly, if $s = \mathsf{Ext}(S_L^2(1^n))$ then D' outputs $D(\langle P, V\rangle(S_L(1^n)))$, whereas if s is random then D' outputs $D(\langle P', V'\rangle(S_L^1(1^n)))$. Thus, D' distinguishes on infinitely many n's with probability at least $\frac{1}{p(n)}$. This contradicts the assumption that L has high extractable entropy.

Next, we prove that

$$\left\{\langle P', V'\rangle(S_L^1(1^n))\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\langle P'', V''\rangle(S_L^1(1^n))\right\}_{n\in\mathbb{N}}.$$

This follows from a straightforward reduction to the assumption that G is a pseudorandom generator that is secure for non-uniform distinguishers (the non-uniformity is needed for the infinite series of n's for which indistinguishability does not hold).

Noting now that (P'', V'') is identical to the original (P_L, V_L) that is zero knowledge, we conclude that (P, V) is zero-knowledge. Since P is deterministic, this concludes the proof.

□

D Instantiations of Collections of Dense 1–1 One-Way Functions

In this section, we show that collections of dense doubly-enhanced 1–1 one-way functions can be instantiated under standard assumptions. We remark that this includes the discrete log assumption (an assumption under which we do not know how to construct trapdoor permutations).

The RSA assumption. The RSA collection of functions has an index set consisting of pairs (N, e), where N is a product of two primes (p and q) and e is relatively prime to $\phi(N) = (p-1)(q-1)$. The fact that the RSA collection fulfills the doubly-enhanced requirement was shown in [18]. The fact that it is dense follows from the fact from the density of prime numbers. In particular, by Bertrand's postulate, the number of $n/2$-bit prime numbers is at least $\alpha \stackrel{\text{def}}{=} \frac{2^{n/2}}{3n/2} = \frac{2^{n/2-1}}{3n}$. Thus, the number of n-bit numbers that are the product of two distinct primes is at least $\binom{\alpha}{2} \approx \frac{2^{n-2}}{18n^2} > \frac{2^n}{n^3}$, where the last inequality holds for large enough n. Thus, one can take $\ell(n) = n^3$ and a random n-bit number is a valid RSA modulus with probability at least $1/\ell(n)$. Observe that for RSA, it is also necessary to find e that is relatively prime to $\phi(N)$. However, the prover can do this deterministically, and can send back to the verifier the index of the modulus and the public exponent e. (Recall that a cheating prover cannot gain anything by cheating, so this is valid.)

The Factoring assumption. The same idea as above can be used to show that (a variant of) the Rabin function meets all requirements (see [18] for the fact that such a variant can be shown to be doubly enhanced).

The Discrete-Logarithm assumption. The discrete log assumption (over a finite field) states that the collection of functions $DLP_{p,g}(x) = g^x \mod p$, for a random safe prime p and generator g is one way. The standard sampler I for this collection samples random q until q and $p = 2q + 1$ are prime, and takes g to be any square. Regarding the enhancements, we define S for the function $DLP_{p,g}$ who outputs a value $y \in \mathbb{Z}_p^*$ that is statistically close to uniform by repeatedly choosing y of length p and outputting it if $y < p$ (and aborting if n repetitions fail). Since each attempt succeeds with probability at least $1/2$, this fulfills the condition. Since the value y is almost uniform, and the randomness used by S defines y directly, it is hard to invert $DLP_{p,g}$ on y under the standard discrete log assumption. Regarding the requirement to generate pairs of random coins for S and its inverse, this can be easily achieved by choosing a random $x \in \mathbb{Z}_q$ and computing $y = g^x$. Then, S as above is run until it succeeds. Finally, all of the random coins used by S in the failed attempt are output, and y is taken as the random coins used by S in the last successful attempt. Algorithm R then outputs these coins, together with x. Finally, regarding the dense requirement, this is the case for the version of the discrete log assumption where a random prime p is taken, and not a safe prime. However, the density of safe primes is not known. Nevertheless, our protocol can easily be instantiated in this case by having (the honest) V generate p, g, and sending the function description to the prover. This works since unlike in the RSA and factoring assumptions, the randomness used to sample p reveals nothing.

Summary. We conclude that deterministic-prover honest-verifier zero-knowledge can be achieved under standard assumptions. That is:

Corollary D.1. *If the RSA, factoring or discrete log assumptions hold, then every language $L \in \mathcal{NP}$ has an honest-verifier public-coin deterministic-prover auxiliary-input zero-knowledge proof.*