Watermarkable and Zero-Knowledge Verifiable Delay Functions from any Proof of Exponentiation

Charlotte Hoffmann and Krzysztof Pietrzak

Institute of Science and Technology Austria
{charlotte.hoffmann, pietrzak}@ista.ac.at

Abstract

A verifiable delay function $\texttt{VDF}(x, T) \to (y, \pi)$ maps an input x and time parameter T to an output y together with an efficiently verifiable proof π certifying that y was correctly computed. The function runs in T sequential steps, and it should not be possible to compute y much faster than that. The only known practical VDFs use sequential squaring in groups of unknown order as the sequential function, i.e., $y = x^{2^T}$. There are two constructions for the proof of exponentiation (PoE) certifying that $y = x^{2^T}$, with Wesolowski (Eurocrypt'19) having very short proofs, but they are more expensive to compute and the soundness relies on stronger assumptions than the PoE proposed by Pietrzak (ITCS'19).

A recent application of VDFs by Arun, Bonneau and Clark (Asiacrypt'22) are short-lived proofs and signatures, which are proofs and signatures which are only sound for some time t, but after that can be forged by anyone. For this they rely on "watermarkable VDFs", where the proof embeds a prover chosen watermark. To achieve stronger notions of proofs/signatures with reusable forgeability, they rely on "zero-knowledge VDFs", where instead of the output y, one just proves knowledge of this output. The existing proposals for watermarkable and zero-knowledge VDFs all build on Wesolowski's PoE, for the watermarkable VDFs there's currently no security proof.

In this work we give the first constructions that transform any PoEs in hidden order groups into watermarkable VDFs and into zkVDFs, solving an open question by Arun et al.. Unlike our watermarkable VDF, the zkVDF (required for reusable forgeability) is not very practical as the number of group elements in the proof is a security parameter. To address this, we introduce the notion of zero-knowledge proofs of sequential work (zkPoSW), a notion that relaxes zkVDFs by not requiring that the output is unique. We show that zkPoSW are sufficient to construct proofs or signatures with reusable forgeability, and construct efficient zkPoSW from any PoE, ultimately achieving short lived proofs and signatures that improve upon Arun et al's construction in several dimensions (faster forging times, weaker assumptions).

A key idea underlying our constructions is to not directly construct a (watermarked or zk) proof for $y = x^{2^T}$, but instead give a (watermarked or zk) proof for the more basic statement that x', y' satisfy $x' = x^r, y' = y^r$ for some r, together with a normal PoE for $y' = (x')^{2^T}$.

Contents

1 Introduction

Verifiable delay functions (VDFs), introduced by Boneh et al. [BBBF18], are functions that take a prescribed amount of time T to compute and can be verified in time much less than T. They have found a lot of applications including the design of blockchains [CP18], randomness beacons [Rab83, SJH+21], proofs of data replication [BBBF18] and computational time-stamping [CE12, LSS20].

Time-Based Deniability from VDFs. A recent application of VDFs was presented by Arun, Bonneau and Clark in [ABC22]. The authors construct so called *short-lived proofs* and *short-lived signatures* from VDFs that satisfy some additional properties. Short-lived proofs and signatures are only valid for a prescribed amount of time T. After time T they are easy to forge by anyone and hence validity cannot be verified anymore. The authors of [ABC22] achieve this notion for any relation $\mathcal{R}$ by combining a proof system for $\mathcal{R}$ with a VDF computation using a simple OR statement: A short-lived proof is correct if either the proof for $\mathcal{R}$ is correct or a VDF computation has been performed. This way the proof for $\mathcal{R}$ is only valid before time T has passed since after time T anyone can output a valid proof by proving that they have performed the VDF computation. One useful property that short-lived proofs and signatures can have is *reusable forgeability*, which means that one slow computation enables efficient proof forgery for many statements.

Zero-Knowledge VDFs. Zero-Knowledge VDFs are VDFs that can verify that a prover $\mathcal{P}$ knows the result $y = x^{2^T}$ without revealing any other information about y to the verifier $\mathcal{V}$, i.e., instead of sending the result y to $\mathcal{V}$, $\mathcal{P}$ and $\mathcal{V}$ engage in a *zero-knowledge proof of knowledge* of y. Such VDFs were introduced by Arun, Bonneau and Clark in [ABC22], where they use it as a building block to construct short-lived proofs. The zero-knowledge VDF in [ABC22] is a zero-knowledge version of Wesolowski's VDF. Using this VDF in the OR construction described above to obtain a short-lived proof provides some form of reusable forgeability: After performing one slow computation of the delay function, one can forge proofs for multiple statements of the same sender by providing a re-randomized VDF proof for each statement. Using the zero-knowledge VDF in [ABC22] computing a re-randomized proof takes time roughly $T/\log(T)$ since this is the time it takes compute Wesolowski's VDF proof. Since Pietrzak's VDF proof can be computed in time $T/\sqrt{T}$, a zero-knowledge version of Pietrzak's VDF would enable much faster forging times. The authors of [ABC22] leave a construction of a zero-knowledge version of Pietrzak's VDF as an open problem.

Watermarkable VDFs. Watermarkable VDFs are VDFs in which the proof can be watermarked, i.e., tied to a specific prover. They were informally introduced by Wesolowski in [Wes19], where he claims that his VDF can be watermarked by including a unique identifier in the computation of the random challenge. The authors of [ABC22] point out that security of this scheme is not proven since the proof of Wesolowski's VDF reveals the value x^q for a large q which may speed up the computation of $y = x^{2^T}$ and hence the computation of a proof with a different watermark. They propose to watermark the proof of their zero-knowledge VDF construction by including a unique identifier in the computation of the random challenge of the proof. Since the protocol is zero-knowledge, the watermarked proof does not reveal any information that might help computing a proof with a different watermark. However, this also means that the value $y = x^{2^T}$ cannot be revealed, which might be relevant in other applications of watermarkable VDFs.

1.1 Our Contribution

We give the first constructions that transform *any* PoE in hidden order groups into a watermarkable VDF and into a zero-knowledge VDF. We note that this gives the first practical watermarkable VDF with a security proof, and the zkVDF solves the open problem stated in [ABC22] asking for a zkVDF based on Pietrzak's VDF.

Instantiating the watermarkable VDFs with Pietrzak's PoE and using it in the [ABC22] construction of short-lived proofs (without reusable forgeability) we get proofs with significantly faster forging times and under different (arguably weaker) assumptions than the construction in [ABC22].

Our watermarkable VDF is practically efficient and only slightly increases the proof size of the PoE. However, the zero-knowledge VDF increases the proof size by roughly 4λ group elements, where λ is a statistical security parameter. While the proof size is still independent of the time parameter T, a blow-up by 4λ group elements is undesirable in practice.

To address this, we introduce the notion of zero-knowledge proofs of sequential work (zkPoSW). We show that zkPoSW can replace zkVDFs in the construction of short-lived proofs and signatures with reusable forgeability from [ABC22], and also give a construction that transforms any PoE into a zkPoSW which only increases the proof size of the PoE by 3 group elements. As before, using Pietrzak's PoE we get short lived proofs and signatures with shorter forging times and different assumptions than [ABC22].

Watermarkable VDFs. We construct the first general watermarkable VDF scheme from any PoE in hidden order groups. In this construction the proof of the statement $x^{2^T} = y$ is computed as follows:

1. Sample a random $r \leftarrow \pm[2^\lambda]$.
2. Compute $x' := x^r$ and $y' := y^r$.
3. Compute a PoE proof $\pi_{\texttt{PoE}}$ for the statement $(x')^{2^T} = y'$.
4. Compute a watermarked zero-knowledge proof of knowledge of r, denoted by $\pi_{\texttt{PoK}}$.
5. Publish $x, x', y, y', \pi_{\texttt{PoE}}$ and $\pi_{\texttt{PoK}}$.

Watermarking the proof of knowledge is done by including a unique identifier in the computation of the random challenge. Watermark unforgeability holds by soundness of the proof of knowledge, a decisional variant of the discrete log assumption in hidden order groups and a decisional variant of the iterated squaring assumption: If the proof of knowledge is sound, then the only two ways for an adversary to forge a proof with its own watermark are the following:

- Find r such that $x^r = x'$ and $y^r = y'$, copy $\pi_{\texttt{PoE}}$ and honestly compute $\pi_{\texttt{PoK}}$. By finding r, the adversary finds a small discrete log of x' with base x.
- Compute a PoE for a new statement $(x^{r'})^{2^T} = y^{r'}$ faster than time T. If the adversary is able to do this, then in particular it can also recognize that y is indeed the result of x^{2^T} in time faster than T, which breaks the decisional iterated squaring assumption.

The construction in [ABC22] is only slightly more efficient than ours: It increases the proof size of Wesolowski's proof by one group element, whereas our construction increases the proof size of a PoE by four group elements. However, our construction can watermark *any* PoE, while the construction in [ABC22] cannot be generalized to other PoEs than Wesolowski's. Further, the watermarkable VDF in [ABC22] cannot reveal the output of the iterated squaring instance since it is based on a zero knowledge VDF, which may be undesirable for other applications.

Zero-Knowledge VDFs. We construct the first general zero-knowledge VDF scheme from any PoE in hidden order groups. It is similar to the watermarkable VDF construction but instead of publishing the element y, the prover just proves knowledge of y. In this construction the proof of the statement $x^{2^T} = y$ is computed as follows:

1. Sample a random $r \leftarrow \pm[2^\lambda]$.
2. Compute $x' := x^r$ and $y' := y^r$.
3. Compute a PoE proof $\pi_{\texttt{PoE}}$ for the statement $(x')^{2^T} = y'$.
4. Compute a zero-knowledge proof of knowledge of y and r, denoted by $\pi_{\texttt{PoK}}$.

5. Publish $x, x', y, y', \pi_{\texttt{PoE}}$ and $\pi_{\texttt{PoK}}$.

To prove that this scheme is zero-knowledge, we need to give the simulator a precomputed pair x^*, y^* and a PoE for the statement $(x^*)^{2^T} = y^*$. Then it can output a simulated proof simply by forging the proof of knowledge of y and r. We need to rely on a decisional version of the discrete log assumption in hidden order groups so that the adversary cannot decide if there exists a small discrete log between elements x^* and x or not. The bottleneck of this construction is the zero-knowledge proof of knowledge of y and r. We obtain it by combing Schnorr's protocol with the Guillou-Quisquater protocol for proving knowledge of a s-root in hidden order groups. In our setting we can only proof soundness of this scheme when using challenge space $\{0, 1\}$ and running λ many repetitions, which makes the scheme unpractical.

Zero-Knowledge Proofs of Sequential Work. We salvage the efficiency of the above scheme by dropping the requirement of a proof of knowledge of y. This means that our construction is not a VDF anymore: The prover might not know the unique result $y = x^{2^T}$ since it can also just compute $y' = (x^r)^{2^T}$ and output a valid proof. However, we assume that computing the proof still requires time T, which is sufficient for a proof of sequential work. We call the assumption that computing $(x^r)^{2^T}$ for an adversarially chosen $r \neq 0$ takes T sequential steps *generalized iterated squaring assumption*. In this construction, given x, the proof of sequential work is computed as follows:

1. Compute $y = x^{2^T}$ together with advice string α.
2. Sample a random $r \leftarrow \pm[2^\lambda]$.
3. Compute $x' := x^r$ and $y' := y^r$.
4. Compute a PoE proof $\pi_{\texttt{PoE}}$ for the statement $(x')^{2^T} = y'$ using α.
5. Compute a zero-knowledge proof of knowledge of r, denoted by $\pi_{\texttt{PoK}}$.
6. Publish $x, x', y, y', \pi_{\texttt{PoE}}$ and $\pi_{\texttt{PoK}}$.

New Assumptions. As discussed above, we use three assumptions in this work that are natural modifications of well-known assumptions but, to the best of our knowledge, have not already been defined in previous work.

- The *decisional discrete log assumption with small exponents* (defined in Section 4) states that given group elements a, b it's hard to decide if there exists a discrete logarithm $w, b = a^w$ even if the discrete logarithm w is guaranteed to be bounded by 2^λ for a security parameter λ (rather than uniform as in the standard discrete log assumption).

  We require this assumption to hold in the groups of unknown order over which the corresponding VDFs are defined. In groups of known order a stronger assumption is sometimes made in Diffie-Hellman key exchange where, for efficiency reasons, the exponents are chosen to be random numbers of only, say 275 bits (`https://www.rfc-editor.org/rfc/rfc7919#section-5.2`). In [Can97] Canetti makes an even stronger assumption (DHI Assumption II) in groups of order $2q + 1$ for a large prime q, which implies that discrete log is hard even when the set from which the exponent is chosen is "well spread", which basically means it can be an arbitrary set of only slightly superpolynomial size.

- The *decisional iterated squaring assumption* (defined in Section 5.2) is the natural decisional variant of the iterated squaring assumption. It states that given a random x in a group of unknown order, not only does it take time T to compute $y = x^{2^T}$, but already distinguishing y from z^2 for a random z is hard. The reason we only consider indistinguishability from a random square z^2 and not a random z is that otherwise the efficiently computable Jacobi symbol could be used to distinguish.

- The *generalized iterated squaring assumption* (defined in Section 7.2) states that given x, computing a tuple (r, y) such that $y = (x^r)^{2^T}$ requires time T. This generalizes the iterated squaring assumption where one requires $r = 1$.

1.2 Related Work

Time-Release Cryptography. VDFs are an example of timed-release cryptographic primitives [May94]. The first such primitives were time-lock puzzles (TLPs) [RSW96] and timed commitments [BN00]. A TLP can be seen as a delay function that also allows efficient computation (and hence verification) of its output via a trapdoor. The TLP from [RSW96] uses repeated squaring as the delay function, with the factorization of the modulus as the trapdoor. Prior to VDFs the notion of proofs of sequential work (PoSWs) was introduced by Mahmoody, Moran and Vadhan [MMV13]. Unlike TLPs, PoSWs can be constructed from random oracles [MMV11]. The construction from [MMV13] is based on random oracles but is not practical as the prover needs *space* linear in T to compute the PoSW. A construction using just $\log(T)$ space was given in [CP18]. Constructions with extra properties like being "reversible" [AKK+19] or "incremental" [DLM19] were proposed shortly afterwards. The sloth function of Lenstra and Wesolowski [LW17] is already close to a unique PoSW. However, verification takes time linear in T. Finally, VDFs were introduced by Boneh et al. [BBBF18]. Mahmoody, Smith and Wu [MSW20] showed that VDFs cannot be built from hash functions.

Proofs of Exponentiation. One way to obtain practical VDFs is to rely on iterated squaring in a hidden order group as the delay function and then construct a proof of exponentiation (PoE) to make the result verifiable. The first PoEs were introduced concurrently by Wesolowski [Wes19] and Pietrzak [Pie19]. Block et al. [BHR+21] presented the first statistically-sound PoE in any group, which they use to build polynomial commitment schemes. The PoE in [HHK+22] is built on their protocol and reduces the complexity, whenever the exponent can be chosen in a special way. [HHKK23] construct a PoE in an extension field of $\mathbb{Z}_N$ to build a VDF from a potentially weaker assumption than iterated squaring. In [HHKP23] a PoE for Proth number groups is given to certify proofs of non-primality. Batching protocols for PoEs can be found in [Rot21] and [HHI24].

Other Verifiable Delay Functions. There are several candidate VDFs not based on iterated squaring, such as the isogenies-based constructions [DMPS19, Sha19, CSRHT22], the permutation-polynomial based construction [BBBF18] and the constructions from lattice problems [LM23, CLM23]. Freitag, Pass and Sirkin [FPS22] constructed VDFs from any sequentially hard function and polynomial hardness of learning with errors. While some of the above construction possibly provide post quantum security, they are currently not as efficient as the VDFs built from iterated squaring. Other VDF candidates rely on "arithmetization friendly" symmetric primitives and practically efficient SNARKs [BBBF18, SB19, KMT22].

Time-Based Deniability. Baldimtsi et al. [BKZZ16] build so called *proofs of work or knowledge*, with which a prover can prove that they either know the witness of a statement or it has solved a proof of work puzzle. However, in their work both cases are indistinguishable from the beginning, whereas in short-lived proofs the cases are indistinguishable only after time T has passed. Specter, Park and Green [SPG21] build protocols that prove that either a prover knows the witness of a statement or it has seen a value released at time T. Ferrari, Géraud and Sirkin [FGN15] construct *fading* signatures that also lose validity after a certain amount of time based on the RSW time-lock puzzle. However, they need to rely on a trusted authority that knows a trapdoor and they need that the verifier is more powerful than the prover.

Colburn [Col18] constructs short-lived proofs and signatures from proofs of work in his master thesis. The proofs of work in his thesis consist of finding preimages of hash functions and are thus parallelizable.

Wesolowski [Wes19] was the first one to use VDFs as a building block for time-based deniability. He presents an identification protocol based on a trapdoor VDF that loses its validity after time T. Finally, Arun, Bonneau and Clark [ABC22] were the first ones to build general short-lived proofs and signatures from VDFs.

2 Preliminaries

In the rest of the paper, we let λ denote a security parameter. We use $[n] := \{1, \ldots, n\}$ to denote the set of all positive integers smaller than or equal to n.

2.1 Relations and Interactive Proofs

A *relation* $\mathcal{R} \subset \mathcal{X} \times \mathcal{W}$ is a set of pairs (x, w), where x is called the instance and w is called the witness. The set of all values x for which there exists a witness w such that $(x, w) \in \mathcal{R}$ is called the *language* $L_{\mathcal{R}}$ for $\mathcal{R}$.

Definition 1 (interactive proof)**.** For a function $\varepsilon : \mathbb{N} \to [0, 1]$, an *interactive proof for a relation* $\mathcal{R}$ is a pair of interacting PPT algorithms $(\mathcal{P}, \mathcal{V})$, called the *prover* and the *verifier*, where $\mathcal{P}$ takes as input a pair $(x, w) \in \mathcal{R}$ and $\mathcal{V}$ takes as input x. We require the algorithms to satisfy the following properties

- **Completeness:** For every $x \in L_{\mathcal{R}}$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on the common instance x, then $\mathcal{V}$ accepts with probability 1.
- **Soundness:** For every $x \notin L_{\mathcal{R}}$ and every cheating prover strategy $\widetilde{\mathcal{P}}$, the acceptance probability of the verifier $\mathcal{V}$ when interacting with $\widetilde{\mathcal{P}}$ is less than $\varepsilon(|x|)$, where ε is called the *soundness error*.

Definition 2 (proof of knowledge)**.** A *proof of knowledge* is an interactive proof that is *knowledge sound*, i.e., for (potentially malicious) prover $\mathcal{P}^*$ that makes $\mathcal{V}$ accept proof π for instance x of bit-length n with probability δ, there exists an efficient extractor $\mathcal{E}$, which can interact with $\mathcal{P}^*$ and outputs a witness w such that $(x, w) \in \mathcal{R}$ with probability at least $(\delta - \varepsilon)/\mathsf{poly}(n)$, where poly is some positive polynomial and $\varepsilon \in [0, 1]$ is called the *soundness error*.

Definition 3 (honest verifier zero-knowledge proof)**.** An *honest verifier zero-knowledge proof* is an interactive proof that is *zero knowledge*, i.e., there exists an efficient simulator $\mathcal{S}$ that, given instance x and a value r, can output an accepting transcript for x with verifier's message r which is indistinguishable from a real transcript with an honest verifier.

It is well known that any constant round interactive proof in which the verifier messages only consist of random elements can be transformed into a non-interactive proof via the Fiat-Shamir heuristic [FS87] by deriving the verifier's messages via a suitable hash function. If the interactive proof is honest verifier zero-knowledge, the non-interactive version is fully *zero-knowledge* (i.e., no assumption on the behavior of the verifier is needed) in the *random oracle model*. In the random oracle model the hash function is modelled as a publicly available random function $\mathcal{O}$. The simulator $\mathcal{S}$ has *programmable* access to $\mathcal{O}$, which means that it can set the output of $\mathcal{O}$ to a value of its choice as long as the distribution of the output values is uniform.

Definition 4 (proof of exponentiation)**.** A *proof of exponentiation* (PoE) in a group $\mathbb{G}$ is an interactive proof for the language

$$L = \{(x, y, e) \in \mathbb{G}^2 \times \mathbb{N} \mid x^e = y\}.$$

2.2 Verifiable Delay Functions

Verifiable Delay Functions were introduced by Boneh et al. in [BBBF18].

Definition 5. A *verifiable delay function* (VDF) is a set of algorithms ($\texttt{Setup}$, $\texttt{Eval}$, $\texttt{Prove}$, $\texttt{Verify}$), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{Prove}(\mathbf{pp}, x, y, \alpha) \to (y, \pi)$ outputs a proof π for y.

$\texttt{Verify}(\mathbf{pp}, x, y, \pi) \to \texttt{accept/reject}$ checks that $y = \texttt{Eval}(\mathbf{pp}, x)$.

The algorithm Eval can compute the output y in T sequential steps. A VDF must additionally satisfy three properties:

Completeness: For all tuples $(\mathbf{pp}, x, y, \pi)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi = \texttt{Prove}(\mathbf{pp}, x, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs accept.

Sequentiality: Any parallel algorithm that uses at most $\mathsf{poly}(\lambda)$ processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Soundness: If $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs accept, then the probability that $y \neq \texttt{Eval}(\mathbf{pp}, x)$ is negligible.

2.3 Assumptions

In this paper we need the following well-known assumptions in hidden-order groups. In Sections 4, 5.2 and 7.2 we state the novel assumption that we need.

Definition 6 (strong RSA assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *strong RSA assumption* holds for GGen if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$ and an element $a \in \mathbb{G}$.
2. $\mathcal{A}$ outputs a pair $(e, b) \in \mathbb{Z} \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $e \neq 1$ and $b^e = a$.

The following assumption, which was first formalized in [BBF18], states that it is (computationally) hard to find elements of low order. Note that our assumption is a bit stronger than theirs because our upper bound on the order is $2^{3\lambda+2}$, while they assume the upper bound 2^λ. There are groups in which this assumption holds information theoretically because such elements do not exist: The group of signed quadratic residues QR_N^+ of an RSA integer $N = pq$, where p and q are safe primes, i.e., $p = 2p' + 1$ and $q = 2q' + 1$ for some primes p', q' that are larger than $2^{3\lambda+2}$.

Definition 7 (low order assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *low order assumption* holds for GGen if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$.
2. $\mathcal{A}$ outputs a pair $(d, a) \in [2^{3\lambda+2}] \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $a \neq 1$ and $a^d = 1$.

The following assumption was first stated by Rivest, Shamir and Wagner [RSW96].

Definition 8 (iterated squaring assumption)**.** Let $\texttt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *iterated squaring assumption* holds for GGen if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $\mathsf{poly}(\lambda)$ processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^\lambda)$, a random group element x and an integer T.
2. $\mathcal{A}$ wins if it outputs element $y = x^{2^T}$.

Instance: $(a_1, a_2, \mathbb{G})$, where $a_1, a_2 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow \pm[2^{3\lambda}]$ uniformly at random, computes $b := a_1^t$ and sends it to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$ and $a_1^s = ba_2^c$ and outputs `accept` or `reject` accordingly.

Figure 1: Proof of Knowledge of Discrete Log [Sch91, KTY04].

Remark 1. We note that, strictly speaking, the iterated assumption as stated above does not hold. Bernstein and Sorenson [BS07] showed that one can reduce the *sequential* time of computing an iterated squaring instance x^{2^T} from T to $T/\log\log(T)$ using at least T^2 processors. While this is a nice theoretical result, it is not practical in our setting. In practice the time parameter T will be at most 2^{32}, so the algorithm by Bernstein and Sorenson can reduce the sequential time by at most a factor of 6, for which it would need at least $T^2 = 2^{64}$ processors. For simplicity we will ignore this $\log\log(T)$ factor in the rest of the paper.

2.4 The Group of Signed Quadratic Residues

One example of a hidden order group that has useful properties is the group of signed quadratic residues [FS00, HK09] with a safe prime modulus. We call a prime number p *safe* if $p = 2p' + 1$ for a prime number p'. We say that $N = pq$ is a *safe prime modulus*, if both p and q are safe primes. Let $\mathbb{Z}_N^*$ denote the multiplicative group modulo N. The group of quadratic residues modulo N is defined as $\mathrm{QR}_N := \{a^2 \mod N : a \in \mathbb{Z}_N^*\}$ and the group of signed quadratic residues is defined as

$$\mathrm{QR}_N^+ := \{|\, b \,| : b \in \mathrm{QR}_N\},$$

where $|\, b \,|$ is the absolute value of b when representing the elements of $\mathbb{Z}_N$ as $\{-(N-1)/2, \ldots, (N-1)/2\}$. QR_N^+ is a cyclic group with group operation $a \circ b := |\, a \cdot b \mod N \,|$. Unlike in QR_N, membership in QR_N^+ can be efficiently tested: We have that $b \in \mathrm{QR}_N^+$ if $0 \leq b \leq (N-1)/2$ and the Jacobi symbol of b modulo N is $+1$.

3 Three Zero-Knowledge Proofs of Knowledge

We begin by presenting three zero-knowledge proofs of knowledge. We need the first one to construct a zero-knowledge proof of sequential work in Section 7, the second one to construct a watermarkable VDF in Section 5 and the third one to construct a zero-knowledge VDF in Section 6. The proofs of knowledge are not *tight*: while the size of the witness of an honest prover is bounded by 2^λ, the extractor might extract a witness of size up to $2^{3\lambda+2}$. We will see later that this is sufficient for our applications.

3.1 Proof of Knowledge of Discrete Log

In Figure 1 we present Schnorr's protocol [Sch91] in hidden order groups. We use it as a building block to construct the zero knowledge PoSW in Section 7. Schnorr originally defined and analyzed the protocol in prime order groups. Later, Kiayias, Tsiounis and Yung [KTY04] proved that it is also secure in hidden order groups, where knowledge soundness is based on the strong RSA assumption. We note that the soundness property only guarantees knowledge of an exponent in $\pm[2^{3\lambda+2}]$ instead of $\pm[2^\lambda]$. This is sufficient for our application. The authors of [CPP17] claim that knowledge soundness of the protocol can be based on the RSA assumption instead of the Strong-RSA assumption but we are not aware of a formal proof.

Theorem 1 ([KTY04])**.** *Under the strong RSA assumption, the protocol in Figure 1 is an honest verifier zero-knowledge proof of knowledge with soundness error* $1/2^\lambda$*. The soundness property guarantees knowledge of an exponent in* $\pm[2^{3\lambda+2}]$.

Theorem 1 is a special case of [KTY04, Theorem 10]. For completeness we restate the proof for this case.

Proof of Theorem 1. Completeness follows by inspection of the protocol. To prove knowledge soundness we construct an extractor $\mathcal{E}$ that outputs a witness w given two accepting transcripts (a_1, a_2, b_1, c, s) and $(a_1, a_2, b_1, c^*, s^*)$. Since both transcripts are accepting, it holds that $a_1^{s-s^*} = a_2^{c-c^*}$. Let $\gamma = \gcd(s-s^*, c-c^*)$ and α, β be such that $\gamma = \alpha(s-s^*) + \beta(c-c^*)$. With high probability it holds that γ is coprime to the order of $\mathbb{G}$ since otherwise we could factor the group order and in particular break the strong RSA assumption. We thus have

$$a_1^{\frac{s-s^*}{\gamma}} = a_2^{\frac{c-c^*}{\gamma}}$$

and hence

$$a_1 = a_1^{\alpha\frac{s-s^*}{\gamma} + \beta\frac{c-c^*}{\gamma}} = (a_2^\alpha a_1^\beta)^{\frac{c-c^*}{\gamma}}.$$

Now if $c - c^* > \gamma$, we can transform the prover into an algorithm that breaks the strong RSA assumption: $a_2^\alpha a_1^\beta$ is the $(c - c^*/\gamma)$-root of a_1. We therefore have that $c - c^* = \gamma$ and hence $w = (s - s^*/c - c^*)$ is the discrete log of a_2 with base a_1. Since $s, s' \in \pm[2^{3\lambda+1}]$ we have that $w \in \pm[2^{3\lambda+2}]$.

It remains to prove that the protocol is honest verifier zero knowledge. Consider the simulator $\mathcal{S}$ that takes as input a tuple (a_1, a_2, c^*), samples $s^* \leftarrow \pm[2^{3\lambda}]$ uniformly at random and computes $b^* = a_1^s a_2^{-c}$. To prove that this transcript is indistinguishable from a real transcript, we show that the statistical distance of the random variable $s^* \leftarrow \pm[2^{3\lambda}]$ to the random variable $s = t + cw$ for a fixed $w \in \pm[2^{2\lambda}]$ and uniformly random $t \leftarrow \pm[2^{3\lambda}]$ and $c \leftarrow [2^\lambda]$ is negligible. Since s^* is distributed uniformly over $\pm[2^{3\lambda}]$, it takes each value in this set with probability $1/2^{3\lambda+1}$. Now consider the distribution of s. Any value in the range $[-2^{3\lambda} + 2^{2\lambda}, 2^{3\lambda} - 2^{2\lambda}]$ is selected with probability $2^\lambda/(2^\lambda 2^{3\lambda+1}) = 1/2^{3\lambda+1}$ since for any choice of c we can find a t that yields the respective value. On the rest of the values the distributions might differ. If follows that the statistical distance of the two distributions is at most

$$1 - \frac{2^{3\lambda+1} - 2^{2\lambda+1}}{2^{3\lambda+1}} = \frac{1}{2^\lambda}.$$

□

3.2 Proof of Knowledge of Same Discrete Log

The main tool in our construction of a watermarkable signature scheme is a proof of knowledge of the same discrete log for two different bases. The protocol is a special case of the general proof of knowledge for "discrete-log relations sets" introduced by Kiayias, Tsiounis and Yung in [KTY04]. It was first constructed in prime order groups by Chaum and Pederson [CP93] We present it in Figure 2.

Theorem 2 ([KTY04, Theorem 10])**.** *Under the Strong-RSA assumption, the protocol in Figure 2 is an honest verifier zero-knowledge proof of knowledge with soundness error* $1/2^\lambda$*. The soundness property guarantees knowledge of an exponent in* $\pm[2^{3\lambda+2}]$.

Theorem 2 is a special case of [KTY04, Theorem 10]. The proof is very similar to the proof of Theorem 1 so we omit it. To make the protocol non-interactive, we apply the Fiat-Shamir heuristic, i.e., we replace the challenge sent by the verifier by a hash of the instance, the first message and an identifier `ID` of the prover. The protocol can be found in Figure 3. In our application to watermarkable VDFs, we need this protocol to be watermarked. We achieve this by including an ID of the prover in the input of the hash function that computes the Fiat-Shamir challenge.

Instance: $(a_1, a_2, a_3, a_4, \mathbb{G})$, where $a_1, a_2, a_3, a_4 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow [2^{3\lambda}]$ uniformly at random, computes $b_1 := a_1^t$ and $b_2 := a_3^t$ and sends (b_1, b_2) to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$, $a_1^s = b_1 a_2^c$ and $a_3^s = b_2 a_4^c$ holds and outputs `accept` or `reject` accordingly.

Figure 2: Proof of Knowledge of same discrete log [KTY04]

Instance: $(a_1, a_2, a_3, a_4, \mathtt{ID}, \mathbb{G})$, where $a_1, a_2, a_3, a_4 \in \mathbb{G}$ and `ID` is a unique identifier of $\mathcal{P}$

Parameters: statistical security parameter λ, hash function H

Witness: Exponent $w \in [2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol:

1. $\mathcal{P}$ samples $t \leftarrow [2^{3\lambda+1}]$ uniformly at random and computes $b_1 := a_1^t$ and $b_2 := a_3^t$.
2. $\mathcal{P}$ computes $c := H(a_1, a_2, a_3, a_4, b_1, b_2, \mathtt{ID})$
3. $\mathcal{P}$ computes $s := t + cw$ and publishes (b_1, b_2, c, s) as the proof.
4. To check the proof (b_1, b_2, c, s), $\mathcal{V}$ checks if $H(a_1, a_2, a_3, a_4, b_1, b_2, \mathtt{ID}) = c$, $s \in \pm[2^{3\lambda+1}]$ and if both $a_1^s = b_1 a_2^c$ and $a_3^s = b_2 a_4^c$ hold and outputs `accept` or `reject` accordingly.

Figure 3: `PoKsDL`: The watermarked non-interactive Proof of Knowledge of same discrete log

Instance: $(a_1, a_2, a_4, \mathbb{G})$, where $a_1, a_2, a_4 \in \mathbb{G}$

Parameters: statistical security parameter λ

Witness: Element a_3 and exponent $w \in \pm[2^\lambda]$ such that $a_1^w = a_2$ and $a_3^w = a_4$ in $\mathbb{G}$

Protocol: $\mathcal{P}$ and $\mathcal{V}$ repeat the following procedure λ times:

1. $\mathcal{P}$ samples $a_5 \leftarrow \mathbb{G}$ uniformly at random, computes $b_1 := a_5^w$ and sends b_1 to $\mathcal{V}$.
2. $\mathcal{V}$ samples a bit $b \leftarrow \{0, 1\}$ uniformly and random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ samples $t \leftarrow \pm[2^{3\lambda}]$ uniformly at random, computes $a_6 := a_5 a_3^b$, $b_2 := a_1^t$ and $b_3 := a_6^t$ and sends (a_6, b_2, b_3) to $\mathcal{V}$.
4. $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
5. $\mathcal{P}$ computes $s := t + cw$ and sends it to $\mathcal{V}$.
6. $\mathcal{V}$ checks if $s \in \pm[2^{3\lambda+1}]$, $a_1^s = b_2 a_2^c$ and $a_6^s = b_3(b_1 a_4^b)^c$ hold and outputs `accept` or `reject` accordingly.

Figure 4: Proof of Knowledge of same discrete log with one hidden base

3.3 Proof of Knowledge of Same Discrete Log with one Hidden Base

In our zero-knowledge VDF construction we need a proof of knowledge that's similar to the one in the last section but without revealing element a_3. The protocol is given in Figure 4. It is a combination of the protocol in Figure 2 and the well-known Guillou-Quisquater protocol [GQ90] for proving knowledge of a root.

Theorem 3. *Under the Strong-RSA assumption, the protocol in Figure 4 is an honest verifier zero-knowledge proof of knowledge with soundness error $1/2^\lambda$. The soundness property guarantees knowledge of an exponent in $\pm[2^{3\lambda+2}]$.*

Proof. Completeness follows by inspection of the protocol. To prove knowledge soundness we consider one of the λ many executions. We construct an extractor $\mathcal{E}$ that outputs a witness (w, a_3) given four accepting transcripts $(b_1, 0, a_6, b_2, b_3, c, s)$, $(b_1, 0, a_6, b_2, b_3, c^*, s^*)$, $(b_1, 1, a_6^*, b_2^*, b_3^*, c^{**}, s^{**})$ and $(b_1, 1, a_6^*, b_2^*, b_3^*, c^{***}, s^{***})$. $\mathcal{E}$ first extracts w and then a_3.

1. Since the first and the second transcripts are accepting, it holds that $a_1^{s-s^*} = a_2^{c-c^*}$. Let $\gamma = \gcd(s - s^*, c - c^*)$ and α, β be such that $\gamma = \alpha(s - s^*) + \beta(c - c^*)$. Then we have

   $$a_1^{\frac{s-s^*}{\gamma}} = a_2^{\frac{c-c^*}{\gamma}}$$

   and hence

   $$a_1 = a_1^{\alpha\frac{s-s^*}{\gamma}+\beta\frac{c-c^*}{\gamma}} = (a_2^\alpha a_1^\beta)^{\frac{c-c^*}{\gamma}}.$$

   Now if $c - c^* > \gamma$, we can transform the prover into an algorithm that breaks the strong RSA assumption: $a_2^\alpha a_1^\beta$ is the $(c - c^*/\gamma)$-root of a_1. We therefore have that $c - c^* = \gamma$ and hence $w = (s - s^*/c - c^*)$ is the discrete log of a_2 with base a_1 and the discrete log of b_1 with base a_6. By the same argument we get that $w = (s^{**} - s^{***}/c^{**} - c^{***})$ is the discrete log of $b_1 a_4$ with base a_6^*.

2. Now consider the second and third transcript. We have seen above that $a_6^w = b_1$ and $(a_6^*)^w = b_1 a_4$. This means that a_6^*/a_6 is a w-root of a_4.

It remains to prove honest verifier zero-knowledge. Given (a_1, a_2, a_4, b, c), the simulator $\mathcal{S}$ constructs an accepting transcript $(b_1, b, a_6, b_2, b_3, c, s)$ as follows: It first samples $s \leftarrow \pm[2^{3\lambda}]$ and $a_6 \leftarrow \mathbb{G}$ uniformly at random. If $b = 0$, $\mathcal{S}$ samples $e \leftarrow \pm[2^{\lambda}]$ uniformly at random and sets $b_1 = a_6^e$. If $b = 1$, it samples $b_1 \leftarrow \mathbb{G}$ uniformly at random. Finally, $\mathcal{S}$ computes $b_2 = a_1^s a_2^{-c}$ and $b_3 = a_6^s b_1^{-c} a_4^{-bc}$. Indistinguishability follows since the distribution of the simulated s has statistical distance $1/2^{\lambda}$ from the distribution of an honestly computed s as we have seen in the proof of Theorem 1. □

4 Modified Discrete-Log Assumptions

In our constructions we need to rely on the assumption that it is hard to recognize whether there exists a small discrete log between two given elements or not. Note that this is easy in one case: Given two elements $a, b \in \mathbb{G}$, where a is a square and b is a non-square, there exists no discrete log of b to base a since a raised to any power yields a square. We assume that it is hard in all other cases.

Definition 9 (discrete log assumption with small exponents)**.** Let $\texttt{GGen}(1^{\lambda})$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *discrete log assumption with small exponents* holds for $\texttt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^{\lambda})$, and two elements $a, b \in \mathbb{G}$, where a is uniformly random and $b = a^w$ for some $w \in \pm[2^{\lambda-1}]$.

2. $\mathcal{A}$ outputs an integer $w' \in \pm[2^{\lambda-1}]$.

3. $\mathcal{A}$ wins if and only if $b = a^{w'}$.

Definition 10 (decisional discrete log assumption with small exponents)**.** Let $\texttt{GGen}(1^{\lambda})$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *decisional discrete log assumption with small exponents* holds for $\texttt{GGen}$ if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\texttt{GGen}(1^{\lambda})$, and two elements $a, b \in \mathbb{G}$, where a is uniformly random and for b there are two possibilities of probability $1/2$ each: Either $b = a^w$ for some $w \in \pm[2^{\lambda-1}]$ or $b = z^{2^b}$ for a uniformly random group element $z \in \mathbb{G}$, where $b = 1$ if a is a square and $b = 0$ if not.

2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not $b = a^w$ for some $w \in \pm[2^{\lambda-1}]$.

3. $\mathcal{A}$ wins if and only if it outputs the correct bit with probability greater than $1/2$.

Remark 2 (the special case of QR_N^+)**.** Note that the decisional discrete log assumption with small exponents holds information theoretically in the group of signed quadratic residues QR_N^+, where N is a safe prime modulus, whenever the group order of QR_N^+ is at least 2^{λ}. This is because in this group almost all elements are generators and all elements are squares. Hence, if you pick two random group elements, the discrete log of one element to the base the other element exists with high probability so the two cases in the assumption are statistically indistinguishable.

Further, in this case, we have a straightforward reduction from the strong RSA assumption to the discrete log assumption with small exponents: Given a random group element g, one can solve the strong RSA challenge by sampling a random group element h and sending (h, g) to the adversary $\mathcal{A}$ that breaks the discrete log assumption with small exponents. When $\mathcal{A}$ outputs w, the reduction sends (w, h) to the strong RSA challenger.

5 Watermarkable VDFs

In this section we show how to transform any PoE into a watermarkable VDF. We begin by recalling the definition of watermarkable VDFs.

5.1 Definition

Watermarkable verifiable delay functions were informally introduced by Wesolowski [Wes19]. The first formal definition was given by Arun, Bonneau and Clark in [ABC22].

Definition 11. A watermarkable VDF is a set of algorithms ($\texttt{Setup}$, $\texttt{Eval}$, $\texttt{WatermarkProve}$, $\texttt{Verify}$), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{WatermarkProve}(\mathbf{pp}, x, \mu, y, \alpha) \to (y, \pi_\mu)$ outputs a proof for y with embedded watermark μ.

$\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu) \to \texttt{accept/reject}$ checks that $y = \texttt{Eval}(\mathbf{pp}, x)$ and that the watermark $\tilde{\mu}$ is embedded in π_μ.

The algorithm $\texttt{Eval}$ can compute the output y in T sequential steps. A watermarkable VDF must additionally satisfy four properties: The security properties of a basic VDF and watermark unforgeability. We state them informally below. The formal definitions can be found in [BBBF18] and [ABC22].

Completeness: For all tuples $(\mathbf{pp}, x, \mu, y, \pi_\mu)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi_\mu = \texttt{WatermarkProve}(\mathbf{pp}, x, \mu, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, \mu, y, \pi_\mu)$ outputs $\texttt{accept}$.

Sequentiality: Any parallel algorithm that uses at most $\mathsf{poly}(\lambda)$ processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Soundness: If $\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu)$ outputs $\texttt{accept}$, then the probability that $y \neq \texttt{Eval}(\mathbf{pp}, x)$ is negligible.

Watermark Unforgeability: For any pair of algorithms $(\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ runs in time $O(\mathsf{poly}(T, \lambda))$ and $\mathcal{A}_1$ runs in time less than T, the probability that $(\mathcal{A}_0, \mathcal{A}_1)$ wins the following game is negligible:

1. The challenger C runs $\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ and sends $\mathbf{pp}$ to $(\mathcal{A}_0, \mathcal{A}_1)$.
2. Precomputation algorithm $\mathcal{A}_0(\mathbf{pp})$ outputs advice string $\tilde{\alpha}$.
3. Challenger C samples a random input x, runs $\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ and sends $(x, y, \tilde{\alpha})$ to $\mathcal{A}_1$.
4. Online algorithm $\mathcal{A}_1$ sends q many watermark queries μ_i to C and obtains $\texttt{WatermarkProve}(\mathbf{pp}, x, \mu_i, y, \alpha) \to \pi_{\mu_i}$.
5. Algorithm $\mathcal{A}_1$ outputs a forgery pair (μ_*, π_{μ_*}) and wins if $\mu_* \neq \mu_i$ for all $i \in [q]$ and $\texttt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu)$ outputs $\texttt{accept}$.

5.2 The Decisional Iterated Squaring Assumption

For security of our construction we need to make the assumption that the decisional version of the iterated squaring assumption is also hard. We consider an adversary that gets as input a pair of group elements (x, y) and needs to decide whether or not $y = x^{2^T}$. The YES instances are pairs (x, x^{2^T}) for a uniformly random group element x. The NO instances are pairs (x, z^2) for uniformly random group elements x and z. Note that it is necessary to square the element z because x^{2^T} is a square and in RSA groups one can rule out that an element is a square, whenever its (efficiently computable) Jacobi symbol is -1.

$\mathtt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, H)$ outputs a finite abelian group $\mathbb{G}$ of unknown order and an efficiently computable hash function H.

$\mathtt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where $y = x^{2^T}$ and α is an advice string for PoE.

$\mathtt{WatermarkProve}(\mathbf{pp}, x, \mathtt{ID}, y, \alpha) \to (y, \pi_\mu)$ outputs y and

$$\pi_\mu = (x', y', \mathtt{PoKsDL}(\mathbf{pp}, x, x', y, y', \mathtt{ID}), \mathtt{PoE}(\mathbf{pp}, x', y', T, \alpha)),$$

where $x' := x^r$ and $y' := y^r$ for some uniformly random $r \leftarrow \pm[2^\lambda]$.

$\mathtt{Verify}(\mathbf{pp}, x, \tilde{\mu}, y, \pi_\mu) \to \mathtt{accept/reject}$ checks if both $\mathtt{PoKsDL}(\mathbf{pp}, x, x', y, y', \mathtt{ID})$ and $\mathtt{PoE}(\mathbf{pp}, x', y', T))$ verify.

Figure 5: A Watermarkable VDF from any proof of exponentiation PoE using the proof of knowledge PoKsDL presented in Figure 3.

Definition 12 (decisional iterated squaring assumption)**.** Let $\mathtt{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *decisional iterated squaring assumption* holds for GGen if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $\mathsf{poly}(\lambda)$ processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by $\mathtt{GGen}(1^\lambda)$, a random group element x, an integer T and a group element y which, with probability 1/2 each, takes one of the following two forms: either $y = z^2$ for a uniformly random group element z or $y = x^{2^T}$.
2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not $y = x^{2^T}$.
3. $\mathcal{A}$ wins if it outputs the correct bit with probability greater than 1/2.

5.3 Construction

In Figure 5 we present our watermarkable VDF. The main idea is to randomize the instance (x, y) to $(x', y') := (x^r, y^r)$ with a secret exponent r and then provide a PoE for the statement $(x')^{2^T} = y'$ and a watermarked proof of knowledge for r using the protocol in Figure 2. We present the protocol as non-interactive since only non-interactive proofs need to be watermarked.

Theorem 4. *Let* PoE *be a complete and sound proof of exponentiation. The algorithms in Figure 5 define a sound and complete VDF, relative to the iterated squaring assumption, the strong RSA assumption and the low order assumption.*

Proof. Sequentiality of the VDF follows immediately from the iterated squaring assumption. Completeness follows by inspection of the protocol from the completeness property of PoE. Soundness follows from the low order assumption, the strong RSA assumption and soundness of PoE. To see this, we show how to transform an adversary $\mathcal{A}$ that outputs an accepting proof

$$\pi_\mu = (x', y', \mathtt{PoKsDL}(\mathbf{pp}, x, x', y, y', \mathtt{ID}), \mathtt{PoE}(\mathbf{pp}, x', y', T))$$

with $x^{2^T} \neq y$ with probability δ into an adversary $\mathcal{B}$ that breaks either the low order assumption, the strong RSA assumption or soundness of PoE with probability δ. The adversary $\mathcal{B}$ does the following:

1. Try to extract the secret exponent r from PoKsDL. If this is not possible, use $\mathcal{A}$ to break the strong RSA assumption similar to the proof of Theorem 2.

2. If r is extractable, compute $\tilde{y} := x^{2^T}$ and $\alpha := \tilde{y}y^{-r}$. If $\alpha \neq 1$, check if $\alpha^r = 1$. If so, then α is an element of low order and r is a multiple of its order. $\mathcal{B}$ outputs (α, r) and breaks the low order assumption.

3. If $\alpha = 1$ or $\alpha^r \neq 1$, then $(x')^{2^T} \neq y'$, so $\texttt{PoE}(\mathbf{pp}, x', y', T)$ is a proof for a false statement, which is a contradiction to the assumption that $\texttt{PoE}$ is sound.

Whenever the proof output by $\mathcal{A}$ is accepting but $x^{2^T} \neq y$, algorithm $\mathcal{B}$ terminates in one of the steps, which concludes the proof. □

Remark 3 (On the running time of algorithm $\mathcal{B}$)**.** Note that in the above proof the running time of algorithm $\mathcal{B}$ might be linear in the time parameter T because it needs to solve an iterated squaring instance in the second step. This means that, to break the low order assumption or the soundness of the PoE, it needs at least time T. Giving an adversary time linear in T to break the soundness of the PoE is necessary for a meaningful soundness definition since an honest prover also needs time T to compute the result of an instance and the corresponding proof. Giving an adversary against the low order assumption time T to break it, is in line with its usage in the literature (see [Wes19, Pie19], where it is needed for soundness of PoEs). If $\mathcal{B}$ breaks the strong RSA assumption it is much faster since it never gets to step 2. In particular, we have that its running time is independent of T in this case.

Theorem 5. *Let PoE be a complete and sound proof of exponentiation. The VDF defined by the algorithms in Figure 5 is watermark unforgeable in the random oracle model, relative to the strong RSA assumption, the decisional discrete log assumption with small exponents and the decisional iterated squaring assumption.*

Proof. We show how to transform an adversary $\mathcal{A}$ that wins the watermark unforgeability game with probability δ into an adversary $\mathcal{B}$ that breaks either the soundness of $\texttt{PoKsDL}$ (and hence the strong RSA assumption), the discrete log assumption or the decisional iterated squaring assumption with probability $\delta/2$.

1. Let q be the number of queries that $\mathcal{A}$ is allowed to make. Upon receiving as input a group $\mathbb{G}$ and a time parameter T, $\mathcal{B}$ precomputes q' tuples $\{(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))\}_{i \in [q']}$, where for all $i \in [q']$, $x_i \leftarrow \mathbb{G}$ is a uniformly random group element, $y_i = x_i^{2^T}$ and $\texttt{PoE}(\mathbf{pp}, x_i, y_i, T)$ is an honestly computed proof of exponentiation. Call L the list of those tuples. $\mathcal{B}$ computes those tuples until L contains q entries in which x_i is a square.

2. $\mathcal{B}$ gets as input a discrete log challenge (g, g^a), where g is a random group element in $\mathbb{G}$ and a is a random number in $\pm[2^{\lambda-1}]$. Note that by the decisional discrete log assumption with small exponents, $\mathcal{B}$ should not be able to find a.

3. $\mathcal{B}$ sends $\mathbb{G}$ to $\mathcal{A}_0$ and obtains advice string $\tilde{\alpha}$.

4. $\mathcal{B}$ gets as input a decisional iterated squaring challenge consisting of two group elements x_d, y_d that are either uniformly random elements in $\mathbb{G}$ or x_d is uniformly random in $\mathbb{G}$ and $y_d = x_d^{2^T}$

5. To simulate the watermark unforgeability game for the statement $x^{2^T} = y$, it chooses one of the following two strategies at random, each with probability 1/2. Note that $\mathcal{B}$ can always forge a proof of knowledge of same discrete log since $\texttt{PoKsDL}$ is honest verifier zero-knowledge and the random oracle is programmable.

   Strategy 1: Compute $y := g^{2^T}$ and $y' := (g^a)^{2^T}$ and send $(\mathbb{G}, H, g, y, \tilde{\alpha})$ to $\mathcal{A}_1$. When $\mathcal{A}_1$ makes a watermark query $\texttt{ID}_i$, sample a random $r \leftarrow \pm[2^{\lambda-1}]$, forge $\texttt{PoKsDL}(\mathbf{pp}, g, g^{a+r}, y, (y')^r, \texttt{ID}_i)$ and compute $\texttt{PoE}(\mathbf{pp}, g^{a+r}, (y')^r, T)$. Send

   $$\pi_i := (g^{a+r}, (y')^r, \texttt{PoKsDL}(\mathbf{pp}, g, g^{a+r}, y, (y')^r, \texttt{ID}_i), \texttt{PoE}(\mathbf{pp}, g^{a+r}, (y')^r, T))$$

   to $\mathcal{A}_1$. If $\mathcal{A}_1$ wins the game, it outputs

   $$(\texttt{ID}_*, \pi_* = (x_*, y_*, \texttt{PoKsDL}(\mathbf{pp}, g, x_*, y, y_*, \texttt{ID}_*)), \texttt{PoE}(\mathbf{pp}, x_*, y_*, T)).$$

If $(x_*, y_*) \neq (x_i, y_i)$ for all $i \in [q]$, abort. Else, let $\ell \in [2^\lambda]$ be such that $(x_*, y_*) = (g^{a+\ell}, (y')^\ell)$. $\mathcal{B}$ tries to extract $a + \ell$ from PoKsDL. If it is successful, it subtracts ℓ from the extracted value and outputs a, thereby breaking the the discrete log assumption. If it is not able to extract, it can use $\mathcal{A}$ to break the strong RSA assumption similiar to the proof of Theorem 2.

Strategy 2: Send $(\mathbf{pp}, x_d, y_d, \tilde{\alpha})$ to $\mathcal{A}_1$. If x_d is a square, remove all tuples $(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$, where x_i is not a square, from the list L. When $\mathcal{A}_1$ makes a watermark query $\texttt{ID}_i$, pick an unused tuple $(x_i, y_i, \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$ from L, forge $\texttt{PoKsDL}(\mathbf{pp}, x_d, x_i, y_d, y_i, \texttt{ID}_i)$ and send

$$\pi_i := (x_i, y_i, \texttt{PoKsDL}(\mathbf{pp}, x_d, x_i, y_d, y_i, \texttt{ID}_i), \texttt{PoE}(\mathbf{pp}, x_i, y_i, T))$$

to $\mathcal{A}_1$. By the decisional discrete log assumption with small exponents and the zero-knowledge property of PoKsDL, π_i is indistinguishable from an honestly computed watermarked proof. If $\mathcal{A}_1$ outputs

$$(\texttt{ID}_*, \pi_* = (x_*, y_*, \texttt{PoKsDL}(\mathbf{pp}, x_d, x_*, y_d, y_*, \texttt{ID}_*), \texttt{PoE}(\mathbf{pp}, x_*, y_*, T)),$$

check if $(x_*, y_*) = (x_i, y_i)$ for some $i \in [q]$ and abort if it holds. Otherwise, try to extract the secret r from PoKsDL. If this is not possible, use $\mathcal{A}$ to break the strong RSA assumption as above. If it is possible, we have that the statement $x_d^{2^T} = y_d$ holds since the PoE is sound. In this case $\mathcal{B}$ sends 1 to the decisional iterated squaring challenger. If $\mathcal{A}_1$ does not output a tuple of the form above, $\mathcal{B}$ sends 0 or 1 to the decisional iterated squaring challenger each with probability 1/2.

If adversary $\mathcal{B}$ does not abort in Strategy 1, it breaks either the strong RSA assumption or the decisional discrete log assumption with small exponents with probability δ. If $\mathcal{B}$ does not abort in Strategy 2, it either breaks the strong RSA assumption with probability δ or it recognizes a true instance in the decisional iterated squaring game with probability $1/2 + \delta/2$. Since aborting in Strategy 1 and aborting in Strategy 2 are mutually exclusive, the claim follows. □

Remark 4 (On the running time of algorithm $\mathcal{B}$)**.** Note that in the first strategy $\mathcal{B}$ runs in time linear in T to break the strong RSA assumption or the discrete log assumption. We therefore need to assume that these assumptions are secure against adversaries that run in time linear in T, which is at most 2^{32} in practice.

The next corollary follows from the discussion in Remark 2.

Corollary 1. *Let PoE be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 5 is a watermarkable VDF in $\mathbb{G}$ relative to the decisional iterated squaring assumption and the strong RSA assumption.*

Efficiency Watermarking a PoE with the construction in Figure 5 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by 4 group elements and one integer of size at most $2^{3\lambda+1}$.
- The verifier needs to perform 4 additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 2 group multiplications.
- The prover needs to perform 4 additional small exponentiations (with exponents of size at most $2^{3l\lambda}$).

6 Zero-Knowledge VDFs

6.1 Definition

Zero-knowledge verifiable delay functions were introduced by Arun, Bonneau and Clark in [ABC22].

Definition 13. A zero-knowledge VDF is a set of algorithms (Setup, Eval, Prove Verify, Sim), where

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

$\texttt{Eval}(\mathbf{pp}, x) \to (y, \alpha)$ on input $(\mathbf{pp}, x, T)$ outputs (y, α), where α is an advice string.

$\texttt{Prove}(\mathbf{pp}, x, y, \alpha) \to \pi$ outputs a proof π of knowledge of element y.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks that π is a valid proof of knowledge.

$\texttt{Sim}(\mathbf{pp}, x, c^*, \tilde{\alpha}) \to \pi^*$ outputs a simulated proof of knowledge π^* using randomness c^* and precomputed advice string $\tilde{\alpha}$.

The algorithm Eval can compute the output y in T sequential steps. A zero-knowledge VDF must additionally satisfy four properties: Completeness, sequentiality, knowledge soundness and zero-knowledge.

Completeness: For all tuples $(\mathbf{pp}, x, y, \pi)$, where $y = \texttt{Eval}(\mathbf{pp}, x)$ and $\pi = \texttt{Prove}(\mathbf{pp}, x, y, \alpha)$, algorithm $\texttt{Verify}(\mathbf{pp}, x, y, \pi)$ outputs accept.

Sequentiality: Any parallel algorithm that uses at most $\mathsf{poly}(\lambda)$ processors and outputs $y = \texttt{Eval}(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Knowledge Soundness: For any adversary $\mathcal{A}$ that outputs a proof π for instance x of bit-length n, such that $\texttt{Verify}(\mathbf{pp}, x, \pi)$ outputs accept with probability δ, there exists an extractor $\mathcal{E}$ that with probability at least $(\delta - \varepsilon)/\mathsf{poly}(n)$ outputs element $y = \texttt{Eval}(\mathbf{pp}, x)$ in time less than T, where poly is some positive polynomial and $\varepsilon \in [0, 1]$ is called the *soundness error*.

Zero Knowledge: There exists a simulator $\mathcal{S}$ that, given instance x, randomness c^* and a precomputed advice string $\tilde{\alpha}$, outputs a proof π^* in time less than T such that $\texttt{Verify}(\mathbf{pp}, x, \pi^*)$ outputs accept and π^* is indistinguishable from an honestly computed proof.

6.2 Construction

Our construction of a zero-knowledge VDF can be found in Figure 6. We note that this construction can be transformed into a watermarkable zero-knowledge VDF by including a unique identifier in the computation of the randomness in the proof of knowledge of same discrete log. Since this extension is a straightforward combination of our two constructions, we refrain from analyzing it formally.

Theorem 6. *Let PoE be a complete and sound proof of exponentiation. The algorithms in Figure 6 define a zero-knowledge VDF, relative to the iterated squaring assumption, the strong RSA assumption, the low order assumption and the decisional discrete log assumption with small exponents.*

Proof. Sequentiality of the VDF follows immediately from the iterated squaring assumption. Completeness follows by inspection of the protocol and from the completeness property of PoE. Knowledge soundness follows from the low order assumption, the strong RSA assumption and soundness of PoE. To see this, we describe an extractor $\mathcal{E}$ that outputs $y = x^{2^T}$ in time less than T by interacting with an adversary $\mathcal{A}$ that outputs an accepting proof

$$\pi = (x', y', \texttt{PoKsDLh}(\mathbf{pp}, x, x', y'), \texttt{PoE}(\mathbf{pp}, x', y', T)).$$

The extractor $\mathcal{E}$ first tries to extract an exponent r and a base element $\tilde{y}$ from PoKsDLh such that $x^r = x'$ and $\tilde{y}^r = y'$. If this is not possible, it can break the strong RSA assumption similar to the proof of Theorem 3. Assume that $\tilde{y} \neq y$. Then we would have that $(\tilde{y}/y)^r = 1$ and hence $\tilde{y}/y$ would be an element of low order. Hence, by the low order assumption $\tilde{y} = y$. Since the running time of the extractor is independent of T, knowledge soundness follows.

It remains to prove zero knowledge. Consider the simulator Sim. From the zero-knowledge property of PoKsDLh and the decisional discrete log assumption with small exponents, we follow that the simulated proof π^* is computationally indistinguishable from an honest proof. □

$\mathtt{Setup}(1^\lambda, T) \rightarrow \mathbf{pp} = (\mathbb{G}, H)$ outputs a finite abelian group $\mathbb{G}$ of unknown order and an efficiently computable hash function H.

$\mathtt{Eval}(\mathbf{pp}, x, T) \rightarrow (y, \alpha)$ outputs (y, α), where $y = x^{2^T}$ and α is an advice string for $\mathtt{PoE}$.

$\mathtt{Prove}(\mathbf{pp}, x, \mathtt{ID}, y, \alpha) \rightarrow \pi$ outputs

$$\pi = (x', y', \mathtt{PoKsDLh}(\mathbf{pp}, x, x', y'), \mathtt{PoE}(\mathbf{pp}, x', y', T)),$$

where $x' := x^r$ and $y' := y^r$ for a uniformly random $r \leftarrow \pm[2^\lambda]$.

$\mathtt{Verify}(\mathbf{pp}, x, \pi) \rightarrow \mathtt{accept/reject}$ checks if $\mathtt{PoKsDLh}(\mathbf{pp}, x, x', y')$ and $\mathtt{PoE}(\mathbf{pp}, x', y', T)$ verify.

$\mathtt{Sim}(\mathbf{pp}, x, c^*, x_1^*, y_1^*, \mathtt{PoE}(\mathbf{pp}, x_1^*, y_1^*, T), , x_2^*, y_2^*, \mathtt{PoE}(\mathbf{pp}, x_2^*, y_2^*, T)) \rightarrow \pi^*$ on input $\mathbf{pp}, x, c^*,$ a random square x_1^*, a random non-square x_2^* and the corresponding PoEs, simulates $\mathtt{PoKsDLh}(\mathbf{pp}, x, x^*, y^*)$ with randomness c^* and outputs

$$\pi^* = (x^*, y^*, \mathtt{PoKsDLh}(\mathbf{pp}, x, x^*, y^*), \mathtt{PoE}(\mathbf{pp}, x^*, y^*, T)),$$

for $x^* := x_1^*$, if x is a square and $x^* \leftarrow \{x_1^*, x_2^*\}$ uniformly random if x is a non-square.

Figure 6: A zero-knowledge VDF from any proof of exponentiation PoE. PoKsDLh is the non-interactive version of the proof of knowledge presented in Figure 4.

The next corollary follows from the discussion in Remark 2.

Corollary 2. *Let PoE be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 6 is a zero-knowledge VDF in $\mathbb{G}$ relative to the iterated squaring assumption and the strong RSA assumption.*

Efficiency Transforming a PoE into a zero-knowledge VDF with the construction in Figure 6 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by $4\lambda + 2$ group elements and λ many integers of size at most $2^{3\lambda+1}$.
- The verifier needs to perform 4λ additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 3λ group multiplications.
- The prover needs to perform $3\lambda + 2$ additional small exponentiations (with exponents of size at most $2^{3\lambda}$).

7 Zero-Knowledge Proofs of Sequential Work

Our construction of a general zero-knowledge VDF is not practical, with the proof of knowledge of y being the bottleneck. Without it our construction does not satisfy the definition of a zero-knowledge VDF: If the prover just needed to output x, x', y' and a proof of knowledge of r such that $x^r = x'$, then it could first raise x to a random r and then compute $y' = (x')^{2^T}$. In particular, it would produce the output without ever knowing y. The main observation in this section is that, while this protocol does not satisfy the definition of a zero-knowledge VDF, it is still sufficient for the application to short-lived proofs presented in [ABC22, Section 7] because the prover still needs time T to compute the output. The protocol can be found in Figure 7 and the application to short-lived proofs in the next section.

7.1 Definition

Definition 14. A zero-knowledge proof of sequential work is a set of algorithms (`Setup`, `Prove` `Verify`, `Sim`), where

`Setup`$(1^\lambda, T) \rightarrow \mathbf{pp}$ on input statistical security parameter 1^λ and time parameter T outputs public parameters $\mathbf{pp}$.

`Prove`$(\mathbf{pp}, x, y, \alpha) \rightarrow \pi$ outputs a proof π of sequential work of time T.

`Verify`$(\mathbf{pp}, x, \pi) \rightarrow$ `accept/reject` checks that π is a valid proof of sequential work.

`Sim`$(\mathbf{pp}, x, c^*, \tilde{\alpha}) \rightarrow \pi^*$ outputs a simulated proof of sequential work π^* using randomness c^* and precomputed advice string $\tilde{\alpha}$.

The algorithm `Eval` can compute the output y in T sequential steps. A zero-knowledge proof of sequential work must additionally satisfy four properties: Completeness, sequentiality, soundness and zero-knowledge.

Completeness: For all tuples $(\mathbf{pp}, x, y, \pi)$, where $y =$ `Eval`$(\mathbf{pp}, x)$ and $\pi =$ `Prove`$(\mathbf{pp}, x, y, \alpha)$, algorithm `Verify`$(\mathbf{pp}, x, y, \pi)$ outputs `accept`.

Sequentiality: Any parallel algorithm that uses at most $\mathsf{poly}(\lambda)$ processors and outputs $\pi =$ `Prove`$(\mathbf{pp}, x)$ with noticeable probability runs in time at least T.

Zero Knowledge: There exists a simulator $\mathcal{S}$ that, given instance x and randomness c^* and a precomputed advice string $\tilde{\alpha}$, outputs a proof π^* in time less than T such that `Verify`$(\mathbf{pp}, x, \pi^*)$ outputs `accept` and π^* is indistinguishable from an honestly computed proof.

7.2 The Generalized Iterated Squaring Assumption

For the security of our construction we need to make the following assumption.

Definition 15 (generalized iterated squaring assumption)**.** Let `GGen`(1^λ) be a randomized algorithm that outputs the description of a hidden-order group $\mathbb{G}$. We say that the *generalized iterated squaring assumption* holds for `GGen` if, for any probabilistic parallel algorithm $\mathcal{A}$ that uses at most $\mathsf{poly}(\lambda)$ processors and runs in time less than T, the probability of winning the following game is negligible in λ:

1. $\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by `GGen`(1^λ), a random group element x and an integer T.
2. $\mathcal{A}$ outputs a pair $(r, y) \in \mathbb{Z} \times \mathbb{G}$.
3. $\mathcal{A}$ wins if and only if $r \neq 0$ and $y = (x^r)^{2^T}$.

7.3 Construction

Theorem 7. *Let `PoE` be a complete and sound proof of exponentiation. The algorithms in Figure 7 define a zero-knowledge PoSW, relative to the generalized iterated squaring assumption, the strong RSA assumption and the decisional discrete log assumption with small exponents.*

Proof. Completeness follows by inspection of the protocol and from the completeness property of `PoE`. Sequentiality of the PoSW follows from the generalized iterated squaring assumption, the RSA assumption and soundness of `PoE`: Assume that an adversary $\mathcal{A}$ can output π in time less than T. We construct an adversary $\mathcal{B}$ that breaks either the RSA assumption or the generalized iterated squaring assumption as follows:

1. $\mathcal{B}$ obtains as input a the description of a group $\mathbb{G}$ and a generalized iterated squaring challenge $x \in \mathbb{G}$.

$\texttt{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, H)$ outputs a finite abelian group $\mathbb{G}$ of unknown order and an efficiently computable hash function H.

$\texttt{Prove}(\mathbf{pp}, x) \to \pi$ outputs

$$\pi := (x', y', \texttt{PoKDL}(\mathbf{pp}, x, x'), \texttt{PoE}(\mathbf{pp}, x', y', T, \alpha)),$$

where $x' := x^r$ and for some uniformly random $r \leftarrow \pm[2^\lambda]$.

$\texttt{Verify}(\mathbf{pp}, x, \pi) \to \texttt{accept/reject}$ checks if both $\texttt{PoKDL}(\mathbf{pp}, x, x')$ and $\texttt{PoE}(\mathbf{pp}, x', y', T))$ verify.

$\texttt{Sim}(\mathbf{pp}, x, c^*, x_1^*, y_1^*, \texttt{PoE}(\mathbf{pp}, x_1^*, y_1^*, T), x_2^*, y_2^*, \texttt{PoE}(\mathbf{pp}, x_2^*, y_2^*, T)) \to \pi^*$ on input $\mathbf{pp}, x, c^*$, a random square x_1^*, a random non-square x_2^* and the corresponding PoEs, simulates $\texttt{PoKDL}(\mathbf{pp}, x, x^*)$ with randomness c^* and outputs

$$\pi^* = (x^*, y^*, \texttt{PoKDL}(\mathbf{pp}, x, x^*), \texttt{PoE}(\mathbf{pp}, x^*, y^*, T)),$$

for $x^* := x_1^*$, if x is a square and $x^* \leftarrow \{x_1^*, x_2^*\}$ uniformly random if x is a non-square.

Figure 7: A Zero-Knowledge Proof of Sequential Work from any proof of exponentiation PoE. PoKDL is the non-interactive version of the proof of knowledge presented in Figure 1.

2. $\mathcal{B}$ forwards $\mathbb{G}$ and x to adversary $\mathcal{A}$.
3. If $\mathcal{A}$ is successful, it outputs a valid proof
$$\pi = (x', y', \texttt{PoKDL}(\mathbf{pp}, x, x'), \texttt{PoE}(\mathbf{pp}, x', y', T, \alpha)).$$
4. $\mathcal{B}$ first tries to extract the secret exponent r from PoKDL. If this is not possible, it can use $\mathcal{A}$ to break the RSA assumption similar to the proof of Theorem 1.
5. If it is possible, $\mathcal{B}$ outputs (r, y') to break the generalized iterated squaring assumption.

The running time of $\mathcal{B}$ is independent of T. By soundness of PoE we have that $\mathcal{B}$ breaks one of the two assumptions with the same probability as the winning probability of $\mathcal{A}$. It remains to prove zero knowledge. Consider the simulator Sim. From the zero-knowledge property of PoKDL and the decisional discrete log assumption with small exponents, we follow that the simulated proof π^* is computationally indistinguishable from an honest proof. □

The next corollary follows from the discussion in Remark 2.

Corollary 3. *Let PoE be a complete and sound proof of exponentiation and let $\mathbb{G} = \mathrm{QR}_N^+$, where N is a safe prime modulus. The construction in Figure 7 is a zero-knowledge PoSW in $\mathbb{G}$ relative to the generalized iterated squaring assumption and the strong RSA assumption.*

Efficiency Transforming a PoE into a zero-knowledge proof of sequential work with the construction in Figure 7 increases the complexity of the underlying PoE scheme as follows:

- The proof size grows by 3 group elements and one integer of size at most $2^{3\lambda+1}$.
- The verifier needs to perform 2 additional small group exponentiations (with exponents of size at most $2^{3\lambda+1}$) and 1 group multiplications.
- The prover needs to perform 3 additional small exponentiations (with exponents of size at most $2^{3\lambda}$).

$\texttt{Setup}(1^\lambda, T) \rightarrow \mathbf{pp} = (\mathbf{pp}_{\texttt{zkPoSW}}, \mathbf{pp}_{\mathcal{R}}) = (\mathbb{G}, H, \mathbf{pp}_{\mathcal{R}})$ outputs a finite abelian group $\mathbb{G}$ of unknown order, an efficiently computable hash function H and the public parameters of the proof system for $\mathcal{R}$.

$\texttt{Precompute}(\mathbf{pp}_{\texttt{zkPoSW}}, T) \rightarrow (b^*, y^*, \pi^*_{\texttt{PoE}})$ outputs a uniformly random $b^* \leftarrow \mathbb{G}$, $y^* = (b^*)^{2^T}$ and $\pi^*_{\texttt{PoE}} = \texttt{PoE}(\mathbf{pp}_{\texttt{zkPoSW}}, b^*, y^*, T)$.

$\texttt{Prove}(\mathbf{pp}, T, x, b, w, b^*, y^*, \pi^*_{\texttt{PoE}}) \rightarrow (\pi_{\texttt{zkPoSW}}, \pi_{\mathcal{R}})$ outputs

- Forged
$$\pi_{\texttt{zkPoSW}} = (b^*, y^*, \texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b^*), \pi_{\texttt{PoE}}),$$
where $\texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b^*)$ is forged with challenge c_1.
- An honestly computed proof $\pi_{\mathcal{R}}$ with random challenge c_2 such that $c_1 + c_2 = c = H(x, b, a)$, where a is the first element of $\pi_{\mathcal{R}}$.

$\texttt{Forge}(\mathbf{pp}, T, x, b) \rightarrow (\tilde{\pi}_{\texttt{zkPoSW}}, \tilde{\pi}_{\mathcal{R}})$ outputs

- Honestly computed
$$\tilde{\pi}_{\texttt{zkPoSW}} = (b', y', \texttt{PoKDL}(\mathbf{pp}_{\texttt{zkPoSW}}, b, b'), \texttt{PoE}(\mathbf{pp}_{\texttt{zkPoSW}}, b', y', T))$$
with random challenge c_1.
- A forged proof $\tilde{\pi}_{\mathcal{R}}$ with challenge c_2 such that $c_1 + c_2 = c = H(x, b, a)$, where a is the first element of $\pi_{\mathcal{R}}$.

$\texttt{Verify}(\mathbf{pp}, x, \pi_{\texttt{zkPoSW}}, \pi_{\mathcal{R}}) \rightarrow \texttt{accept/reject}$ checks $\pi_{\texttt{zkPoSW}}$ and $\pi_{\mathcal{R}}$ and outputs $\texttt{accept}$ if and only if both proofs verify.

Figure 8: A short-lived proof from our zero knowledge PoSW.

8 Short-Lived Proofs from our Zero-Knowledge PoSW

In this section we discuss how one can use our zero knowledge PoSW in the short-lived proof construction of [ABC22]. The main idea in the construction of [ABC22] is to transform any sigma protocol Σ for a relation $\mathcal{R}$ into a short lived proof for $\mathcal{R}$ by combining Σ with a zero-knowledge VDF (which is also a sigma protocol) via the standard OR combination of sigma protocols. Since anyone can construct a valid VDF proof in time T, the combined proof loses its validity after time T. The zero-knowledge property of the VDF is needed since an honest prover needs to be able to forge a VDF proof in time less than T, which is indistinguishable from an honest proof also after time T has passed. We first recall some facts about sigma protocols before presenting our construction of a short-lived proof.

8.1 Sigma Protocols

Definition 16 (sigma protocol)**.** A *sigma protocol* (Σ-protocol) is an interactive honest verifier zero-knowledge proof of knowledge consisting of three messages:

- a first message by $\mathcal{P}$ denoted by u,
- a second message by $\mathcal{V}$ denoted by c and
- a third message by $\mathcal{P}$ denoted by z.

Cramer, Damgård and Schoenmakers [CDS94] showed that the set of relations with Σ-protocols is closed under disjunction: Let $\Sigma_1 = (u_1, c_1, z_1)$ be a sigma protocol for relation $\mathcal{R}_1$ and $\Sigma_2 = (u_2, c_2, z_2)$ be a sigma protocol for relation $\mathcal{R}_2$ and let x_1 be an instance of $\mathcal{R}_1$ and x_2 an instance of $\mathcal{R}_2$. The following protocol is an honest verifier zero-knowledge proof of knowledge of either a witness w_1 for x_1 or a witness w_2 for x_2. Assume without loss of generality that $\mathcal{P}$ knows witness w_1.

1. $\mathcal{P}$ picks a random c_2 and simulates $\Sigma_2 = (u_2, c_2, z_2)$.
2. $\mathcal{P}$ computes the message u_1 and sends (u_1, u_2) to $\mathcal{V}$.
3. $\mathcal{V}$ sends a random message c to $\mathcal{P}$.
4. $\mathcal{P}$ computes $c_1 = c \oplus c_2$, computes the honest third message z_1 and sends (z_1, z_2) to $\mathcal{V}$.
5. $\mathcal{V}$ accepts if and only if $c_1 \oplus c_2 = c$ and the transcripts for both Σ_1 and Σ_2 are valid.

8.2 Our Construction

In this section we show how to transform any sigma protocol Σ for a relation $\mathcal{R}$ into a short-lived proof for relation $\mathcal{R}$. The protocol can be found in Figure 8. It differs from the construction of [ABC22] in three ways:

- We don't work with a zero-knowledge VDF but a zero-knowledge PoSW. This is possible because the protocol does not need the uniqueness property of the VDF.
- We need the honest prover to precompute a PoE because it simulates the outputs of the zkPoSW and the simulator takes as input a precomputed PoE in our construction. This means that in our construction the prover needs to spend time T once to compute a PoE. Afterwards, it can compute more valid proofs by re-randomizing this PoE, which is more efficient than T. With Pietrzak's PoE [Pie19] it takes time $T/\sqrt{T}$.
- Our zkPoSW is not a sigma protocol but the proof of knowledge `PoKDL` is. In our construction it is sufficient to combine Σ and `PoKDL` via the standard disjunction of sigma protocols.

Using Pietrzak's PoE [Pie19] one can not only re-randomize the precomputed PoEs but also the PoEs needed for the forged proofs. Hence, it achieves much faster forging times than the construction based on a zero-knowledge version of Wesolowksi's proof given in [ABC22].

9 Conclusion and Open Problems

In this work we have seen how to efficiently watermark any proof of exponentiation to obtain practical watermarkable VDFs. We also constructed practical zero-knowledge proofs of sequential work that can be used to build short-lived proofs for any NP statement with fast forging times. Our zero-knowledge VDF construction is asymptotically efficent but not practical: The proof size grows by a factor λ because the proof of knowledge that is being used as a building block needs λ repetitions to be sound. We conclude with three interesting open problems:

- One interesting open problem that remains is to construct a *practical* zero-knowledge version of Pietrzak's VDF, by either removing the need for λ repetitions in our general construction or by working directly with Pietrzak's protocol.
- The simulator of our zero-knowledge proof of sequential work takes as input a precomputed PoE. In the application to short-lived proofs the honest prover plays the role of this simulator so it also needs to precompute a PoE, which takes time T. When using Pietrzak's PoE this work T only needs to be performed once since one can re-randomize PoEs in time $T/\sqrt{T}$. However, it would be nice to remove the need for precomputation altogether, which we leave as an open problem.

- In this work we made two new sequentiality assumptions: the decisional iterated squaring assumption and the generalized iterated squaring assumption. While those assumption are very similar to the iterated squaring assumption, we don't have a reduction from iterated squaring to either of the new assumptions. It would be nice to give more evidence of hardness by either constructing such a reduction or computing lower bounds in an idealized model.

References

[ABC22] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 487–516, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany. 3, 4, 6, 14, 17, 19, 22, 23

[AKK+19] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 277–291, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 6

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 3, 6, 7, 14

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712. 8

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. 6

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 902–933, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany. 6

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany. 6

[BS07] D.J. Bernstein and J.P. Sorenson. Modular exponentiation via the explicit chinese remainder theorem. *Mathematics of Computation*, 76:443–454, 2007. 9

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany. 5

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany. 23

[CE12] Jeremy Clark and Aleksander Essex. CommitCoin: Carbon dating commitments with Bitcoin - (short paper). In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 390–398, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Heidelberg, Germany. 3

[CLM23] Valerio Cini, Russell W. F. Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In *Advances in Cryptology – CRYPTO 2023, Part II*, Lecture Notes in Computer Science, pages 72–105, Santa Barbara, CA, USA, August 2023. Springer, Heidelberg, Germany. 6

[Col18] Michael Colburn. Short-lived signatures. Master's thesis, Concordia University, 2018. 6

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. 10

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 3, 6

[CPP17] Geoffroy Couteau, Thomas Peters, and David Pointcheval. Removing the strong RSA assumption from arguments over the integers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 321–350, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 10

[CSRHT22] Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 441–460, Cham, 2022. Springer International Publishing. 6

[DLM19] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 292–323, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 6

[DMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. 6

[FGN15] Houda Ferradi, Rémi Géraud, and David Naccache. Slow motion zero knowledge identifying with colliding commitments. In *Revised Selected Papers of the 11th International Conference on Information Security and Cryptology - Volume 9589*, Inscrypt 2015, page 381–396, Berlin, Heidelberg, 2015. Springer-Verlag. 6

[FPS22] Cody Freitag, Rafael Pass, and Naomi Sirkin. Parallelizable delegation from LWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 623–652, Chicago, IL, USA, November 7–10, 2022. Springer, Heidelberg, Germany. 6

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. 7

[FS00] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, March 2000. 9

[GQ90] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany. 12

[HHI24] Charlotte Hoffmann, Pavel Hubáček, and Svetlana Ivanova. Practical batch proofs of exponentiation. Cryptology ePrint Archive, Paper 2024/145, 2024. https://eprint.iacr.org/2024/145. 6

[HHK$^+$22] Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 370–399, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. 6

[HHKK23] Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, and Tomáš Krňák. (verifiable) delay functions from lucas sequences. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 336–362. Springer, 2023. 6

[HHKP23] Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, and Krzysztof Pietrzak. Certifying giant nonprimes. In *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, Lecture Notes in Computer Science, pages 530–553. Springer, Heidelberg, Germany, May 10–13, 2023. 6

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany. 9

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: Candidate sequential function for ethereum VDF. Cryptology ePrint Archive, Report 2022/1626, 2022. https://eprint.iacr.org/2022/1626. 6

[KTY04] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. 9, 10, 11

[LM23] Russell W. F. Lai and Giulio Malavolta. Lattice-based timed cryptography. Lecture Notes in Computer Science, pages 782–804, Santa Barbara, CA, USA, August 2023. Springer, Heidelberg, Germany. 6

[LSS20] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 541–558, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Heidelberg, Germany. 3

[LW17] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017. 6

[May94] Timothy C. May. Timed-release crypto, 1994. 6

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. 6

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013: 4th Innovations in Theoretical Computer Science*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. Association for Computing Machinery. 6

[MSW20] Mohammad Mahmoody, Caleb Smith, and David J. Wu. Can verifiable delay functions be based on random oracles? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020: 47th International Colloquium on Automata, Languages and Programming*, volume 168 of *LIPIcs*, pages 83:1–83:17, Saarbrücken, Germany, July 8–11, 2020. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. 6

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. LIPIcs. 6, 16, 23

[Rab83] Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983. 3

[Rot21] Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 382–414, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany. 6

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996. 6, 8

[SB19] Kineret Segal and Tom Brand. Presenting: VeeDo a STARK-based VDF service. Technical report, StarkWare, 2019. 6

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, jan 1991. 9

[Sha19] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Paper 2019/205, 2019. https://eprint.iacr.org/2019/205. 6

[SJH$^+$21] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In *ISOC Network and Distributed System Security Symposium – NDSS 2021*, Virtual, February 21–25, 2021. The Internet Society. 3

[SPG21] Michael A. Specter, Sunoo Park, and Matthew Green. KeyForge: Non-attributable email from forward-forgeable signatures. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1755–1773. USENIX Association, August 11–13, 2021. 6

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. 3, 6, 14, 16