IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 1, 19 pages.

https://doi.org/10.62056/av7tudhdj

A Survey of Two Verifiable Delay Functions Using Proof of Exponentiation

Dan Boneh[1], Benedikt Bünz[2] and Ben Fisch[3]

[1] Stanford University, Computer Science, Stanford, U.S.A
[2] New York University, Computer Science, New York, U.S.A
[3] Yale University, Computer Science, New Haven, U.S.A

Abstract. A verifiable delay function (VDF) is an important tool used for adding delay in decentralized applications. This paper surveys and compares two elegant verifiable delay functions, one due to Pietrzak (ITCS'19), and the other due to Wesolowski (Eurocrypt'19). We provide a new computational proof of security for one of them, present an attack on a natural but incorrect implementation of the other, and compare the complexity assumptions needed for both schemes.

Keywords: Verifiable Delay Functions · Proof of Exponentiation · Groups of Unknown Order

1 Introduction: What is a Verifiable Delay Function?

A verifiable delay function (VDF) [LW17, BBBF18] is a function $f : \mathcal{X} \to \mathcal{Y}$ that takes a prescribed time to compute, even on a parallel computer. However once computed, the output can be quickly verified by anyone. Moreover, every input $x \in \mathcal{X}$ must have a unique valid output $y \in \mathcal{Y}$.

In more detail, a VDF is a tuple of three algorithms:

- $\mathit{Setup}(1^\lambda, T) \to pp$ is a PPT algorithm that takes as input a security parameter λ and a time bound T, both positive integers, and outputs public parameters pp. This pp includes a description of the domain $\mathcal{X}$ and range $\mathcal{Y}$ of the VDF.
- $\mathit{Eval}(pp, x) \to (y, \pi)$ takes an input pp and $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$ and a proof π. Let $\mathit{Eval}_1(pp, x)$ be the algorithm that runs $\mathit{Eval}(pp, x)$ and outputs only $y \in \mathcal{Y}$. While Eval can be randomized, we require that for every (pp, x) there is a unique $y \in \mathcal{Y}$ such that $\mathit{Eval}_1(pp, x; r) = y$, for all random bit strings r consumed by Eval.
- $\mathit{Verify}(pp, x, y, \pi) \to \{\mathit{accept}, \mathit{reject}\}$ outputs accept if y is the correct evaluation of the VDF on input x. The algorithm must run in time at most $\mathrm{poly}(\lambda, \log T)$.

We require that for all pp output by $\mathit{Setup}(1^\lambda, T)$ and all $x \in \mathcal{X}$ we have that

$$\text{if } (y, \pi) \leftarrow \mathit{Eval}(pp, x) \text{ then } \Pr[\mathit{Verify}(pp, x, y, \pi) = \mathit{accept}] = 1.$$

A VDF must satisfy three properties. We state these properties informally and refer to [BBBF18] for a complete definition:

- **ϵ-evaluation time:** algorithm $\mathit{Eval}(pp, x)$ runs in time at most $(1 + \epsilon)T$ for all pp output by $\mathit{Setup}(1^\lambda, T)$, all $x \in \mathcal{X}$, and all random bits consumed by Eval. We will explain how to measure run time in the next section.

E-mail: dabo@cs.stanford.edu (Dan Boneh), bbuenz@gmail.com (Benedikt Bünz), benafisch@gmail.com (Ben Fisch)

Received: 2024-01-08 Accepted: 2024-03-05

- **Sequentiality:** a parallel algorithm $\mathcal{A}$, using at most $poly(\lambda)$ processors, that runs in time less than T cannot compute the function. That is, for every such $\mathcal{A}$, if pp is sampled by $Setup(1^\lambda, T)$ and x is uniform in $\mathcal{X}$, then $\Pr\left[\mathcal{A}(pp, x) = Eval_1(pp, x)\right]$ is negligible in λ. This should hold even if $\mathcal{A}$ can spend time $\text{poly}(\lambda, T)$ to preprocess pp before x is given. Here we are assuming that T is at most poly-log in $|\mathcal{X}|$.
- **Uniqueness:** for an input $x \in \mathcal{X}$, exactly one $y \in \mathcal{Y}$ will be accepted by *Verify*. Specifically, for every PPT algorithm $\mathcal{A}$, if pp is sampled by $Setup(1^\lambda, T)$ and $\mathcal{A}(pp)$ outputs (x, y, π), then

$$\Pr\left[\, Verify(pp, x, y, \pi) = accept \;\text{ AND }\; Eval_1(pp, x) \neq y\right]$$

  is a negligible function of λ.

VDFs have many applications. They are useful for constructing a verifiable randomness beacon [BN22], and they provide a "proof of elapsed time" for certain blockchain designs [CP18]. We refer to [BBBF18, Sec. 2] and [MLQ23] for a survey of their applications.

How to build a VDF? A VDF is built from a computational task that cannot be sped up substantially by parallelism. Some candidate tasks include:

- Iterating a cryptographic hash function $H : \mathcal{X} \to \mathcal{X}$. In particular, for certain H it is believed that computing $y \leftarrow H(H(H(\cdots H(x)\cdots)))$ for a random $x \in \mathcal{X}$ cannot be substantially sped up by parallelism. The challenge in using this for a VDF is designing a fast verifier to check that y is correct. We briefly discuss VDFs based on this task in Section 7. These VDFs use a general purpose succinct argument system, a SNARK, to convince the verifier that y is correct.
- Exponentiation in a finite abelian group of unknown order [Wes19, Wes20, Pie19]. Constructing VDFs this way is the main topic of this survey.
- Exponentiation in a finite abelian group of known order. The Minroot VDF candidate [KMT22] uses several rounds of exponentiation in the multiplicative group $\mathbb{Z}/p$, for some fixed 256-bit prime p. Unfortunately, it was later discovered that exponentiation in $\mathbb{Z}/p$ can be sped up by parallel processing using sufficiently many processors [LMPR23].
- Sequential composition of isogenies on elliptic curves [DMPS19]. We discuss VDFs based on this task in Section 7.

We also mention that the mere existence of *some* task that cannot be parallelized, is sufficient to construct a delay mechanism [BGJ+16, JMRR21]. These generic constructions rely on obfuscation or other heavy tools, and cannot, currently, be practically instantiated.

This survey. We survey two VDFs built from exponentiation in a finite abelian group of unknown order. For certain families of such groups it is believed that exponentiation cannot be substantially sped up by parallelism. Rotem and Segev [RS20] proved that speeding up repeated squaring in a generic ring is equivalent to factoring. Rivest, Shamir, and Wagner [RSW96] relied on this property to construct a time-lock puzzle.

Two elegant VDF proposals, one due to Wesolowski [Wes19, Wes20], and another due to Pietrzak [Pie19], make use of the serial nature of exponentiation. These two constructions are the main topic of this survey. Our goal is to present, analyze, and compare both constructions using a unified notation. Along the way we make two observations. First, in Theorem 1 we give a new security analysis of the Pietrzak proof of exponentiation that expands the set of groups where the protocol is sound. This is needed in instantiations of the protocol that use class groups. Second, in Section 3.3 we present an attack on a natural but incorrect implementation of the Wesolowski proof of exponentiation.

2 Two Verifiable Delay Functions

We begin by describing a general framework for building a VDF using exponentiation in a finite abelian group of unknown order. Both the VDF constructions of Pietrzak [Pie19] and Wesolowski [Wes19] follow this framework. This abstract VDF operates as follows.

- The setup algorithm $\mathit{Setup}(1^\lambda, T)$ outputs two objects:
  - A finite abelian group $\mathbb{G}$ of unknown order; we discuss specific families of groups in Section 6;
  - An efficiently computable hash function $H : \mathcal{X} \to \mathbb{G}$ that is modeled as a random oracle.

  The public parameters pp are set to be $pp := (\mathbb{G}, H, T)$.
- The evaluation algorithm $\mathit{Eval}(pp, x)$ is defined as follows:
  - compute $y \leftarrow H(x)^{(2^T)} \in \mathbb{G}$ by squaring $H(x)$ a total of T times in $\mathbb{G}$,
  - compute the proof π as described later,
  - output (y, π).

We measure the evaluation running time in terms of the number of group operations in $\mathbb{G}$ needed to compute the function. For certain families of groups, it is believed that computing y requires T sequential squarings in $\mathbb{G}$ even on a parallel computer with $\mathit{poly}(\lambda)$ processors. As we will see, computing the proof π increases the running time to $(1 + \epsilon)T$, which satisfies ϵ-evaluation time. In practice one might set $T = 2^{30}$ and $\epsilon = 0.01$.

The remaining question is how a public verifier $\mathit{Verify}(pp, x, y, \pi)$ can quickly check that the output y is correct, namely that $y = H(x)^{(2^T)}$. This is where the two VDF constructions differ. They give two different public coin succinct arguments for proving that the output y is correct. Thanks to the public coin nature of these arguments, they can be made non-interactive using the Fiat-Shamir transformation [FS87].

Proving correctness of the output y. To state the problem more abstractly, let us use the following notation:

- let $\mathbb{G}$ be a finite abelian group whose order is unknown to the VDF evaluator;
- let $g := H(x) \in \mathbb{G}$ be the base element given as input to the VDF evaluator;
- let $h := y \in \mathbb{G}$ be the purported output of the VDF, namely $h = g^{(2^T)}$;
- let $T > 0$ be a publicly known integer.

The VDF evaluator needs to produce a proof that a given tuple $(\mathbb{G}, g, h, T)$ satisfies $h = g^{(2^T)}$ in $\mathbb{G}$. More precisely, we need a succinct public coin interactive argument for the language

$$\mathcal{L}_{\text{EXP}} := \Big\{ (\mathbb{G}, g, h, T) \ : \ h = g^{(2^T)} \text{ in } \mathbb{G} \Big\}. \tag{1}$$

There is a simple private coin proof system for this language [Mao01, Sec. 4]. However, designing a succinct *public coin* argument is harder. A public coin protocol is important for practical applications because it can be made non-interactive using the Fiat-Shamir transform.

The prover and verifier take as input $(\mathbb{G}, g, h, T)$ and do:

0. The verifier checks that g, h are in $\mathbb{G}$ and outputs *reject* if not.
1. The verifier sends to the prover a random prime ℓ sampled uniformly from the set $\mathit{Primes}(\lambda)$.
2. The prover computes $q, r \in \mathbb{Z}$ such that $2^T = q\ell + r$ with $0 \leq r < \ell$, and sends $\pi \leftarrow g^q \in \mathbb{G}$ to the verifier.
3. The verifier computes $r \leftarrow 2^T \bmod \ell$ and outputs *accept* if $\pi \in \mathbb{G}$ and $h = \pi^\ell g^r$ in $\mathbb{G}$.

Figure 1: Wesolowski's succinct argument for $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{EXP}}$

Proofs of exponentiation (PoE). The remainder of the paper focuses on two public coin succinct arguments for $\mathcal{L}_{\text{EXP}}$, one by Wesolowski and one by Pietrzak. These arguments are called *Proof of Exponentiation*, or PoE, and they have been used in many applications beyond VDFs. For example, these PoE have been used to build efficient accumulators [BBF19, LM19] and polynomial commitments [BFS20, AGL+23, BHR+21]. They are deployed in the Great Internet Mersenne Prime Search (GIMPS) project to prove that every Mersenne probable prime test (PRP) was done correctly. We stress that these PoE are sound only when the order of the group $\mathbb{G}$ is unknown to the prover.

2.1 Wesolowski's succinct argument for $\mathcal{L}_{\text{EXP}}$

Wesolowski [Wes19, Wes20] presents a succinct public coin interactive argument for the language $\mathcal{L}_{\text{EXP}}$ defined in (1). Specifically, given a tuple $(\mathbb{G}, g, h, T)$ as input, the prover and verifier engage in the protocol in Figure 1 to prove that $h = g^{(2^T)}$ in $\mathbb{G}$. Here we use $\mathit{Primes}(\lambda)$ to denote the set containing the first 2^λ primes, namely $2, 3, 5, 7$, etc. We prove security of the protocol in Section 3.

We note that the protocol works equally well when the exponent 2^T is an arbitrary integer e, not necessarily a power of two. The verifier just needs a quick way to compute $r := e \bmod \ell$ in step (3).

Non-interactive variant. When the protocol is made non-interactive using the Fiat-Shamir transform it is necessary to generate the prime ℓ in step (1) from the set $\mathit{Primes}(2\lambda)$, the set containing the first $2^{2\lambda}$ primes. Generating ℓ from $\mathit{Primes}(\lambda)$, as in the interactive protocol in Figure 1, results in an insecure protocol; we present an attack in Section 3.3. With this modification, the non-interactive variant obtained by applying Fiat-Shamir, works by having the prover first generate ℓ using a hash function that maps the input $(\mathbb{G}, g, h, T)$ to an element of $\mathit{Primes}(2\lambda)$. The analysis will assume that this hash function is a random oracle. The prover computes $\pi \leftarrow g^q$ as in step (2) in Figure 1, and outputs this $\pi \in \mathbb{G}$ as the proof. The verifier computes ℓ the same way as the prover and decides to accept or reject as in step (3). Overall, the proof π is a single element in $\mathbb{G}$.

The Verifier's work. The verifier needs to hash the transcript into $\mathit{Primes}(2\lambda)$. It also needs to compute $r \leftarrow 2^T \bmod \ell$, which only takes $\log_2 T$ multiplications in $\mathbb{Z}/\ell$. Beyond that, the verifier only computes two small exponentiations in $\mathbb{G}$. The verifier was further optimized in [AVD22].

The Prover's work. The prover needs to compute $\pi = g^q \in \mathbb{G}$ where $q = \lfloor 2^T/\ell \rfloor$. Because T is large, we cannot write out q as an explicit integer exponent. Nevertheless,

The prover and verifier take as input $(\mathbb{G}, g, h, T)$ and do:

0. The verifier checks that g, h are in $\mathbb{G}$ and outputs *reject* if not.
1. If $T = 1$ the verifier checks that $h = g^2$ in $\mathbb{G}$, outputs *accept* or *reject*, and stops.
2. If $T > 1$ the prover and verifier do:
   (a) The prover computes $v \leftarrow g^{(2^{T/2})} \in \mathbb{G}$ and sends v to the verifier. The verifier checks that $v \in \mathbb{G}$ and, if not, outputs *reject* and stops.

   Next, the prover needs to convince the verifier that $h = v^{(2^{T/2})}$ and $v = g^{(2^{T/2})}$, which together prove that $h = g^{(2^T)}$. Because the same exponent is used in both equalities, the two checks can be folded into a single equality test by checking a random linear combination, namely that

   $$v^r h = (g^r v)^{(2^{T/2})} \quad \text{for a random } r \text{ in } \{1, \ldots, 2^\lambda\}.$$

   The verifier and prover do so as follows.
   (b) The verifier sends to the prover a random r in $\{1, \ldots, 2^\lambda\}$.
   (c) Both the prover and verifier compute $g_1 \leftarrow g^r v$ and $h_1 \leftarrow v^r h$ in $\mathbb{G}$.
   (d) The prover and verifier recursively engage in an interactive proof that $(\mathbb{G}, g_1, h_1, T/2) \in \mathcal{L}_{\text{EXP}}$, namely that $h_1 = g_1^{(2^{T/2})}$ in $\mathbb{G}$.

Figure 2: Pietrzak's succinct argument for $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{EXP}}$

the prover can compute $\pi = g^q$ in at most $2T$ group operations and constant space using the long-division algorithm, where the quotient is computed "in the exponent" base g. The algorithm works as follows:

$\pi \leftarrow 1 \in \mathbb{G}, \quad r \leftarrow 1 \in \mathbb{Z}$

repeat T times:

$b \leftarrow \lfloor 2r/\ell \rfloor \in \{0, 1\}$ and $r \leftarrow (2r \bmod \ell) \in \{0, \ldots, \ell - 1\}$

$\pi \leftarrow \pi^2 g^b \in \mathbb{G}$

output π *// this π equals g^q*

The running time can be reduced to about T group operations using a windowing method where we process k bits of 2^T per iteration, for some parameter $k \geq 1$, say $k = 5$.

In Appendix A we describe an extension that speeds up the computation of g^q by a factor of s using s processors. Hence, the VDF output and the proof π can be computed in total time approximately $(1 + \frac{1}{s})T$ with s processors and space s. Wesolowski [Wes19, Wes20] shows that with space 2^k one can further speed up the computation by a factor of k.

2.2 Pietrzak's succinct argument for $\mathcal{L}_{\text{EXP}}$

Pietrzak [Pie19] presents a different succinct public coin interactive argument for the language $\mathcal{L}_{\text{EXP}}$ defined in (1). Given a tuple $(\mathbb{G}, g, h, T)$ as input, the prover and verifier engage in a recursive protocol shown in Figure 2 to prove that $h = g^{(2^T)}$ in $\mathbb{G}$. For simplicity, we assume that T is a power of two in which case the protocol takes $\log_2 T$ rounds. The protocol can be adjusted to handle arbitrary T, including a T that is not a power of two [Pie19]. We prove security of this protocol in Section 3.

Non-interactive variant. When the protocol is made non-interactive using Fiat-Shamir the prover generates the challenge r in every level of the recursion by hashing the entire transcript to this point, and appends v to the overall proof π. Hence, the overall proof π contains $\log_2 T$ elements in $\mathbb{G}$.

The Verifier's work. At every level of the recursion the verifier does two λ-bit exponentiations in $\mathbb{G}$ to compute g_1 and h_1 for the next level. Hence, verifying the proof takes about $2\log_2 T$ small exponentiations in $\mathbb{G}$.

The Prover's work. The prover computes the quantity v at every level of the recursion. We let v_1, r_1 be the values of v and r at the top level of the recursion, v_2, r_2 the values at the next level, and so on. Unwinding the recursion shows that these quantities are:

$$
\begin{aligned}
v_1 &= g^{(2^{T/2})} \\
v_2 &= g_1^{(2^{T/4})} = (g^{r_1} v_1)^{(2^{T/4})} = \left(g^{(2^{T/4})}\right)^{r_1} g^{(2^{3T/4})} \\
v_3 &= g_2^{(2^{T/8})} = (g_1^{r_2} v_2)^{(2^{T/8})} = (g^{r_1 r_2} v_1^{r_2} v_2)^{(2^{T/8})} \\
&\qquad = \left(g^{(2^{T/8})}\right)^{r_1 r_2} \left(g^{(2^{3T/8})}\right)^{r_1} \left(g^{(2^{5T/8})}\right)^{r_2} g^{(2^{7T/8})} \\
v_4 &= g_3^{(2^{T/16})} = \text{a power product of eight elements} \\
&\qquad\qquad g^{(2^{T/16})},\ g^{(2^{3T/16})},\ g^{(2^{5T/16})},\ \ldots,\ g^{(2^{15T/16})}
\end{aligned}
$$

The pattern that emerges suggests an efficient way to construct the proof π. When the VDF evaluator first computes the VDF output $h = g^{(2^T)}$ it stores 2^d group elements $g^{(2^{(i \cdot T/2^d)})}$ for $i = 0, \ldots, 2^d - 1$ as they are encountered along the way. Later, as it constructs the proof π, these 2^d stored values let it compute the group elements $v_1, \ldots, v_d$ needed for the proof using a total of about 2^d small exponentiations in $\mathbb{G}$ (each exponent is a product of some subset of $\{r_1, \ldots, r_d\}$ and is therefore at most $d\lambda$ bits). The prover computes the remaining elements $v_{d+1}, v_{d+2}, \ldots, v_{\log T}$ from scratch by raising $g_{d+1}, g_{d+2}, \ldots, g_{\log T}$ to the appropriate exponents. This step takes a total of $T/2^d$ multiplications in $\mathbb{G}$. Hence, the total time to compute the proof is about $2^d + T/2^d$, which suggests that $d = \frac{1}{2}\log_2 T$ is optimal. Hence, the VDF output and the proof π can be computed in total time approximately $(1 + \frac{2}{\sqrt{T}})T$.

3 Security assumptions needed to prove soundness

To analyze security of these interactive arguments for $\mathcal{L}_{\text{EXP}}$ we rely on two complexity assumptions: the low order assumption and the adaptive root assumption. We prove soundness of Pietrzak's argument in groups where the low order assumption holds. We prove soundness of Wesolowski's argument in groups where the adaptive root assumption holds. We discuss the relation between these assumptions in Section 4.

Notation. In what follows we use $x \xleftarrow{R} S$ to denote an independent uniform random variable over the set S, and use $y \xleftarrow{R} \mathcal{A}(x)$ to denote the random variable that is the output of a randomized algorithm $\mathcal{A}$ on input x. We say that a function $f : \mathbb{Z} \to \mathbb{R}$ is a **negligible** function of λ if $|f(\lambda)| = o(1/\lambda^d)$ for all $d > 0$. A **group generator** is a PPT algorithm $GGen(1^\lambda)$ that outputs the description of a group $\mathbb{G}$. In Section 6 we will discuss group generators that are believed to output the description of a group of unknown order.

We say that an interactive argument for $\mathcal{L}_{\text{EXP}}$ has **negligible soundness error** if for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ the following function is negligible in λ

$$\mathsf{adv}_{\mathcal{A},V}(\lambda) := \Pr\left[\begin{array}{ll} (\mathbb{G}, g, h, T) \notin \mathcal{L}_{\text{EXP}}, & \mathbb{G} \xleftarrow{R} GGen(1^\lambda), \\ \langle \mathcal{A}_1(\mathsf{state}), V(\mathbb{G}, g, h, T)\rangle = \mathit{accept} & : \ (g, h, T, \mathsf{state}) \xleftarrow{R} \mathcal{A}_0(\mathbb{G}) \end{array}\right]$$

where $\langle \mathcal{A}_1(\mathsf{state}), V(\mathbb{G}, g, h, T)\rangle = \mathit{accept}$ is the event that the verifier V with input $(\mathbb{G}, g, h, T)$ outputs accept when interacting with $\mathcal{A}_1(\mathsf{state})$ as the prover. The definition extends to the random oracle model by giving $\mathcal{A}_0$, $\mathcal{A}_1$, and V access to the random oracle.

3.1 Security of Pietrzak's succinct argument

The low order assumption says that it is hard to find a low order element in a group $\mathbb{G}$ output by a group generator $GGen(1^\lambda)$.

Definition 1. We say that the **low order assumption** holds for $GGen$ if there is no PPT algorithm $\mathcal{A}$ that takes as input the description of a group $\mathbb{G}$ generated by $GGen(1^\lambda)$, and outputs a pair (μ, d) where $\mu^d = 1$ for $1 \neq \mu \in \mathbb{G}$ and $1 < d < 2^\lambda$. We say that $\mathcal{A}$ outputs a low order element μ in $\mathbb{G}$. More precisely, the advantage

$$\text{LOadv}_{\mathcal{A},GGen}(\lambda) := \Pr\left[\mu^d = 1, \quad 1 \neq \mu \in \mathbb{G}, \quad 1 < d < 2^\lambda \quad : \quad \begin{array}{l} \mathbb{G} \xleftarrow{R} GGen(1^\lambda), \\ (\mu, d) \xleftarrow{R} \mathcal{A}(\mathbb{G}) \end{array}\right]$$

is a negligible function of λ.

The following theorem proves soundness of Pietrzak's succinct argument using the low order assumption. The proof is given in Section 5.

Theorem 1. *Suppose the low order assumption holds for GGen. Then Pietrzak's interactive succinct argument has negligible soundness error.*

> *Concretely, let $\mathcal{A}$ be an algorithm that succeeds with probability ϵ in the following task: $\mathcal{A}$ takes a description of $\mathbb{G} \xleftarrow{R} GGen(1^\lambda)$ as input, outputs a tuple $(\mathbb{G}, g, h, T) \notin \mathcal{L}_{EXP}$ where $1 \leq T < 2^t$ is a power of two, and convinces the verifier to incorrectly accept this tuple. Then there is an algorithm $\mathcal{B}$, whose running time is about twice that of $\mathcal{A}$, that breaks the low order assumption for GGen with advantage at least $\epsilon' := (\epsilon^2/t) - (\epsilon/2^\lambda)$. Hence if ϵ' is negligible then so must be ϵ.*

Necessity of the low order assumption. The low order assumption is necessary for soundness of the protocol; if the assumption does not hold for $GGen$ then the protocol becomes insecure. To see why, let $\mathbb{G} \xleftarrow{R} GGen(1^\lambda)$ and let $\mu \in \mathbb{G}$ be a known element of low order $d > 1$ (i.e., low order is broken). Let $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{EXP}}$. Then the prover can cause the tuple $(\mathbb{G},\ g,\ h\mu,\ T) \notin \mathcal{L}_{\text{EXP}}$ to be incorrectly accepted by the verifier with probability $1/d$. To do so, the prover sends $v \leftarrow g^{(2^{T/2})}\mu \in \mathbb{G}$ to the verifier. We claim that this causes $(\mathbb{G},\ g,\ h\mu,\ T)$ to be incorrectly accepted whenever the verifier chooses an r satisfying $r+1 \equiv 2^{T/2} \pmod d$. This happens with probability $1/d$, which is non-negligible when d is small.

To see why $(\mathbb{G},\ g,\ h\mu,\ T)$ is incorrectly accepted when $r + 1 \equiv 2^{T/2} \pmod d$ observe that the tuple $\big(\mathbb{G},\ g^r v,\ v^r(h\mu),\ T/2\big)$ generated in the recursive step, Step (2d) in Figure 2, is a tuple in $\mathcal{L}_{\text{EXP}}$. To see why, recall that the prover sets $v = g^{(2^{T/2})}\mu$ and moreover $\mu^{2^{(T/2)}} = \mu^{r+1}$. Hence,

$$\begin{aligned}(g^r v)^{2^{(T/2)}} &= \big(g^r \cdot g^{(2^{T/2})}\mu\big)^{2^{(T/2)}} = \big(g^r\big)^{2^{(T/2)}} \cdot \big(g^{2^{(T/2)}}\big)^{2^{(T/2)}} \cdot \mu^{2^{(T/2)}} \\ &= \Big(g^{2^{(T/2)}}\Big)^r \cdot g^{(2^T)} \cdot \mu^{2^{(T/2)}} = (v/\mu)^r \cdot h \cdot \mu^{r+1} = v^r h\mu.\end{aligned}$$

Therefore $\big(\mathbb{G},\ g^r v,\ v^r(h\mu),\ T/2\big)$ is in $\mathcal{L}_{\text{EXP}}$, which means that the verifier accepts it in the recursive step. As a result, the verifier incorrectly accepts $\big(\mathbb{G},\ g,\ h\mu,\ T\big)$ whenever it chooses an r satisfying $r+1 \equiv 2^{T/2} \pmod{d}$. This happens with probability $1/d$.

Concretely, this attack shows that for 128-bit security, the adversary had better not be able to find an element in $\mathbb{G}$ of order less than 2^{128}.

Remark 1. If the group $\mathbb{G}$ contains no low order elements, other than the identity, then the low order assumption holds unconditionally, as does soundness of Pietrzak's succinct argument. The original analysis of Pietrzak [Pie19] focused on this case, which excludes some desirable instantiations, such as class groups, where low order elements may exist. We discuss this further in Section 6.

Remark 2. Even the if the adversary has an element in $\mathbb{G}$ of very low order, say order 2, the attack above only lets it fool the verifier with probability 1/2. Block et al. [BHR+21] use this to show that by running λ instances of Pietrzak's protocol in parallel, we obtain a proof system that is unconditionally sound, with soundness error of about $1/2^\lambda$. Of course, this comes at the cost of making the proof λ times bigger and λ times harder to generate. For special exponents e that come up in applications, Hoffmann et al. [HHK+22] show that one can reduce the number of required parallel instances to about $\lambda/\log\log e$.

3.2 Security of Wesolowski's succinct argument

For the next assumption recall that $\mathit{Primes}(\lambda)$ denotes the set of first 2^λ positive integer primes.

Definition 2. We say that the **adaptive root assumption** holds for GGen if there is no PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in the following task. First, $\mathcal{A}_1$ outputs an element $w \in \mathbb{G}$ and some state. Then, a random prime ℓ in $\mathit{Primes}(\lambda)$ is chosen and $\mathcal{A}_2(\ell, \mathsf{state})$ outputs $w^{1/\ell} \in \mathbb{G}$. More precisely, the advantage

$$\mathrm{ARadv}_{(\mathcal{A}_1,\mathcal{A}_2),\mathit{GGen}}(\lambda) := \Pr\left[u^\ell = w \neq 1 \ : \begin{array}{l} \mathbb{G} \xleftarrow{R} \mathit{GGen}(1^\lambda), \\ (w, \mathsf{state}) \xleftarrow{R} \mathcal{A}_1(\mathbb{G}), \\ \ell \xleftarrow{R} \mathit{Primes}(\lambda), \\ u \xleftarrow{R} \mathcal{A}_2(\ell, \mathsf{state}) \end{array} \right] \tag{2}$$

is a negligible function of λ.

The advantage is always at least $1/|\mathit{Primes}(\lambda)|$. Indeed, if the adversary $(\mathcal{A}_1, \mathcal{A}_2)$ correctly guesses $\ell \in \mathit{Primes}(\lambda)$ ahead of time, then $\mathcal{A}_1$ would output $w \leftarrow u^\ell$, for some $u \in \mathbb{G}$, and $\mathcal{A}_2$ would output this u. This is why we must choose the set $\mathit{Primes}(\lambda)$ to be sufficiently large. The reason we cannot choose ℓ uniformly in some interval, but must choose it from $\mathit{Primes}(\lambda)$, is because a random ℓ in $\{1, \ldots, 2^\lambda\}$ has a reasonable chance of being a smooth integer. The adversary can then win by having $\mathcal{A}_1$ output $w \leftarrow u^B$ where B is a product of all small prime powers up to some bound k, and having $\mathcal{A}_2$ output $u^{B/\ell}$. This works whenever ℓ is a k-smooth integer, so that B/ℓ is an integer. Choosing ℓ as a prime number eliminates this attack.

The following theorem proves soundness of Wesolowski's succinct argument using the adaptive root assumption. The proof is given in Section 5.

Theorem 2 (Wesolowski [Wes19])**.** *Suppose the adaptive root assumption holds for GGen. Then Wesolowski's interactive succinct argument has negligible soundness error.*

> *Concretely, let $\mathcal{A}$ be an algorithm that succeeds with probability ϵ in the following task: $\mathcal{A}$ takes $\mathbb{G} \xleftarrow{R} \mathit{GGen}(1^\lambda)$ as input, outputs a tuple $(\mathbb{G}, g, h, T) \notin \mathcal{L}_{\mathit{EXP}}$, and convinces the verifier to incorrectly accept this tuple. Then there is an adversary $(\mathcal{B}_1, \mathcal{B}_2)$ whose combined running time is about the same as the running time of $\mathcal{A}$ plus the time to compute T squarings in $\mathbb{G}$. This $(\mathcal{B}_1, \mathcal{B}_2)$ breaks the adaptive root assumption for GGen with the same advantage ϵ that $\mathcal{A}$ breaks soundness.*

Necessity of the adaptive root assumption. The adaptive root assumption is necessary for soundness of the protocol; if the assumption does not hold for $GGen$ then the protocol becomes insecure. To see why, let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adaptive root adversary and let $\mathbb{G} \xleftarrow{R} GGen(1^\lambda)$. To break the protocol using $(\mathcal{A}_1, \mathcal{A}_2)$ choose an arbitrary $g \in \mathbb{G}$, fix some T, and run $(w, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathbb{G})$, where $w \neq 1$. Let $h \leftarrow g^{(2^T)}$. Now, let us see how to convince the verifier to incorrectly accept the tuple $(\mathbb{G},\ g,\ wh,\ T) \notin \mathcal{L}_{\text{EXP}}$. The verifier outputs a random $\ell \in Primes(\lambda)$ and we need to produce a π such that $wh = \pi^\ell g^r$, where $r \equiv 2^T \bmod \ell$ and $0 \leq r < \ell$. To do so, calculate q and $r \in [0, \ell)$ such that $2^T = q\ell + r$. Then run $\mathcal{A}_2(\ell, \mathsf{state})$ to a get $u \in \mathbb{G}$ such that $u^\ell = w$. Now $\pi := ug^q$ is a valid proof because

$$\pi^\ell g^r = (ug^q)^\ell g^r = u^\ell g^{q\ell+r} = u^\ell g^{(2^T)} = wh,$$

as required. Concretely, this attack shows that for 128-bit security, we must ensure that the challenge space from which ℓ is chosen contains at least 2^{128} primes.

3.3 Security of the non-interactive variants

While Theorems 1 and 2 analyze the interactive variants of the protocols, the non-interactive variants obtained by applying the Fiat-Shamir transform can be similarly shown to be secure by an analysis in the random oracle model.

Interestingly, for Wesolowski's succinct argument, there is a loss of soundness between the the interactive and non-interactive variants of the protocol, as already mentioned in Section 2.1. To explain the issue, let us first recall a basic result about 2-special sound sigma protocols, such as Schnorr's proof of knowledge of discrete log [BS22, Ch. 19]. For such protocols, if the interactive version is secure with a λ-bit verifier challenge, then the non-interactive version obtained via Fiat-Shamir is also secure with a λ-bit challenge [AFK23]. One might expect that the same holds for Wesolowski's argument. However, that is not the case. We show that when the Fiat-Shamir hash function outputs a challenge ℓ in $Primes(\lambda)$, there is a practical $\tilde{O}(2^{\lambda/2})$ time attack on Wesolowski's non-interactive argument that succeeds with probability close to 1. That is, a malicious prover can fool the verifier into accepting an incorrect statement $(\mathbb{G}, g, h, T) \notin \mathcal{L}_{\text{EXP}}$ with $\tilde{O}(2^{\lambda/2})$ work. Concretely, taking $\lambda = 128$ results in an insecure argument that can be defeated by a malicious prover with about 2^{64} work.

Our attack matches the security analysis of the non-interactive variant. Wesolowski shows in [Wes19] that if the Fiat-Shamir hash function, modeled as a random oracle, outputs a prime ℓ in $Primes(2\lambda)$, then a malicious prover will fool the verifier in the non-interactive argument with probability at most $Q^2/2^{2\lambda}$, where Q is the maximum number of queries that the adversary makes to the random oracle. Hence, any malicious prover that succeeds with probability close to 1, must make at least $Q \geq 2^\lambda$ queries to the random oracle, and therefore must run in time at least 2^λ, which is infeasible. Our attack described below shows that the expression $Q^2/2^{2\lambda}$ is not an artifact of the proof, but an inherent property of the protocol.

The attack and the matching security analysis shows that for Wesolowski's non-interactive argument to be secure, one must use a Fiat-Shamir hash function that outputs a challenge ℓ in $Primes(2\lambda)$. Concretely for 128-bit security, the hash function must output a 256-bit prime ℓ.

The attack. Let us see a $\tilde{O}(2^{\lambda/2})$ time attack on Wesolowski's non-interactive argument when the Fiat-Shamir hash function outputs a prime ℓ in $Primes(\lambda)$. Let H be a hash function that maps an input $(\mathbb{G}, g, h', T)$ to a prime ℓ in $Primes(\lambda)$.

- Let $S = \{\ell_1, \ldots, \ell_d\} \subset Primes(\lambda)$ be a subset of $d := 2^{\lambda/2}$ primes.
- Set $\quad L := \prod_{i=1}^{d} \ell_i \in \mathbb{Z}, \quad w := g^L \in \mathbb{G}, \quad$ and $\quad h := g^{(2^T)} \in \mathbb{G}.$

Then $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{EXP}}$. We can assume that $w \neq 1$, otherwise choose a different subset S. Now, the attacker finds the smallest $k \geq 1$ such that

$$H(\mathbb{G},\ g,\ hw^k,\ T) = \ell \quad \text{and} \quad \ell \in S. \tag{3}$$

If we model H as a random oracle, then each candidate $k = 1, 2, \ldots$ satisfies (3) with probability $d/2^\lambda = 2^{-\lambda/2}$. Therefore, finding k takes an expected $O(2^{\lambda/2})$ tries. Once k is found, observe that

$$\pi := g^{\lfloor 2^T/\ell \rfloor} \cdot g^{k(L/\ell)} \in \mathbb{G} \qquad \text{satisfies} \qquad \pi^\ell = \left(h \cdot g^{-(2^T \bmod \ell)}\right) \cdot w^k$$

because $(g^{k(L/\ell)})^\ell = w^k$ and $\ell \cdot \lfloor 2^T/\ell \rfloor = 2^T - (2^T \bmod \ell)$. The attacker can compute this π because the exponent (L/ℓ) is an integer. This π incorrectly convinces the verifier that $h' := hw^k$ satisfies $h' = g^{(2^T)}$. Indeed, the verification condition

$$\pi^\ell \cdot g^{(2^T \bmod \ell)} = hw^k = h' \qquad \text{where} \qquad \ell := H(\mathbb{G}, g, h', T)$$

is satisfied, thus breaking soundness. The complete attack runs in expected time $\tilde{O}(2^{\lambda/2})$, which is the time needed to compute w and to find k, as promised.

Concretely, for 128-bit security, this attack shows that the verifier must generate its random challenge ℓ from $\mathit{Primes}(2^{256})$. Generating ℓ from $\mathit{Primes}(2^{128})$ would make the non-interactive version of the protocol insecure. In the interactive version of the protocol, sampling ℓ from $\mathit{Primes}(2^{128})$ is fine, as implied by Theorem 2.

4 Comparing the two protocols

Each proof system has its own strengths and no one dominates the other. The proof system of Wesolowski [Wes19, Wes20] produces shorter proofs (one group element versus $\log_2 T$ elements) and proof verification is faster (two exponentiations versus $2\log_2 T$). However, the proof of Pietrzak [Pie19] has two advantages discussed below.

Prover efficiency. For the VDF application, Pietrzak's prover is more efficient. It takes $O(\sqrt{T})$ group operations to construct the proof, where as for Wesolowski it takes $O(T)$. Both provers parallelize well and can be sped up by a factor of s using s processors, for a moderate value of s.

Comparing the assumptions. If Wesolowski's protocol is secure then so is Pietrzak's, but the converse is not known to be true. The reason is that if the adaptive root assumption holds then so must the low order assumption. In other words, adaptive root is potentially a stronger assumption than low order.

To show that the adaptive root assumption implies the low order assumption we show the contra-positive: if low order is broken then so is adaptive root. Let $\mathbb{G} \xleftarrow{R} \mathit{GGen}(1^\lambda)$ and let $1 \neq \mu \in \mathbb{G}$ be a public element satisfying $\mu^d = 1$ for a known $1 < d < 2^\lambda$ (i.e., low order is broken). To break the adaptive root assumption, the adversary $\mathcal{A}_1$ outputs μ, and when given a random prime number $\ell \in \mathit{Primes}(\lambda)$, adversary $\mathcal{A}_2$ computes $\mu^{1/\ell}$ as $\mu^{(\ell^{-1} \bmod d)}$. This works as long as $\gcd(\ell, d) = 1$. Since ℓ is a prime, this is equivalent to ℓ is not a factor of d, which holds with overwhelming probability. Hence, $\mathcal{A}_2$ succeeds with overwhelming probability.

5 Security proofs

Proof of Theorem 2. We construct an adaptive root adversary $(\mathcal{B}_1, \mathcal{B}_2)$ that uses $\mathcal{A}$. When $\mathcal{B}_1$ is initialized with input $\mathbb{G}$, it runs $\mathcal{A}(\mathbb{G})$ and gets back $(\mathbb{G}, g, h, T) \notin \mathcal{L}_{\text{EXP}}$.

Algorithm $\mathcal{B}_1$ then outputs $w \leftarrow h/g^{(2^T)} \in \mathbb{G}$, $\mathsf{state} \leftarrow (\mathbb{G}, g, h, T, w)$ and exits. Note that because $h \neq g^{(2^T)}$ we have that $w \neq 1$, as required of an adaptive root adversary.

Next, a random $\ell \in \mathit{Primes}(\lambda)$ is chosen and $\mathcal{B}_2(\ell, \mathsf{state})$ is activated. Let $2^T = q\ell + r$ with $0 \leq r < \ell$. Algorithm $\mathcal{B}_2$ sends the ℓ it was given to $\mathcal{A}$, and $\mathcal{A}$ outputs $\pi \in \mathbb{G}$. Now, $\mathcal{B}_2$ outputs $u \leftarrow \pi/g^q \in \mathbb{G}$ and exits. If $\mathcal{A}$ outputs a valid proof, namely π satisfies $h = \pi^\ell g^r$, then

$$u^\ell = (\pi/g^q)^\ell = \pi^\ell/g^{q\ell} = \pi^\ell g^r/g^{q\ell+r} = h/g^{(2^T)} = w.$$

Hence, $(\mathcal{B}_1, \mathcal{B}_2)$ succeeds in breaking the adaptive root assumption with the same advantage as $\mathcal{A}$ succeeds in breaking soundness, as required. □

Proof of Theorem 1. We use a forking argument to construct an adversary $\mathcal{B}$ that breaks the low order assumption using $\mathcal{A}$. Recall that 2^t is an upper bound on the value T output by $\mathcal{A}$.

Let $\mathcal{A}(\mathbb{G}, r_0, \ldots, r_{t-1}; R)$ denote an execution of $\mathcal{A}$ with random tape R, where $r_0, \ldots, r_{t-1} \in \{1, \ldots, 2^\lambda\}$ are the verifier's challenges at each level of the recursion. The adversary $\mathcal{A}$ outputs the protocol transcript which is a sequence of $t+1$ tuples:

$$(P_0, v_0), \ldots, (P_t, v_t)$$

where $P_i = (\mathbb{G}, g_i, h_i, T/2^i)$ is the input to the recursion at level i, and $v_i \in \mathbb{G}$ is the prover's message at level i, for $i = 0, 1, \ldots, t$. Here $P_0 = (\mathbb{G}, g_0, h_0, T) \notin \mathcal{L}_{\text{EXP}}$ is the tuple chosen by $\mathcal{A}$ that it is trying to get the verifier to accept incorrectly. $\mathcal{A}$ does so by choosing $v_0, \ldots, v_t \in \mathbb{G}$ in response to the random challenges $r_0, \ldots, r_{t-1}$. The tuples $P_1, \ldots, P_t$ are defined by the protocol in Figure 2 so that $g_i \leftarrow g_{i-1}^{r_{i-1}} v_{i-1}$ and $h_i \leftarrow v_{i-1}^{r_{i-1}} h_{i-1}$ for $i = 1, \ldots, t$. We assume that $T = 2^t$, but if T is a power of two less than 2^t, then we replicate the last pair $(P_{\log_2 T}, v_{\log_2 T})$ to get a full transcript of $t+1$ tuples.

Next, define the following probabilistic experiment EXP:

1. choose a random tape R for $\mathcal{A}$.
2. choose uniform $r_0, \ldots, r_{t-1}$ in $\{1, \ldots, 2^\lambda\}$.
3. run $\mathcal{A}(\mathbb{G}, r_0, \ldots, r_{t-1}; R)$ to get $(P_0, v_0), \ldots, (P_t, v_t)$.
4. if $P_0 \notin \mathcal{L}_{\text{EXP}}$ but $P_t \in \mathcal{L}_{\text{EXP}}$ (i.e., the verifier incorrectly accepts P_0) then:
   - let j be the lowest index for which $P_j \notin \mathcal{L}_{\text{EXP}}$ but $P_{j+1} \in \mathcal{L}_{\text{EXP}}$.
   - choose fresh uniform $r'_j, \ldots, r'_{t-1}$ in $\{1, \ldots, 2^\lambda\}$.
   - run $\mathcal{A}(\mathbb{G}, r_0, \ldots, r_{j-1}, r'_j, \ldots, r'_{t-1}; R)$ to get

     $$(P_0, v_0),\ \ldots,\ (P_j, v_j),\ (P'_{j+1}, v'_{j+1}),\ \ldots,\ (P'_t, v'_t).$$

   - if $P'_{j+1} \in \mathcal{L}_{\text{EXP}}$ and $r_j \neq r'_j$, output $(g_j, h_j, T/2^{j+1}, v_j, r_j, r'_j)$ and stop.
5. in all other cases output *fail*.

Let $\mathcal{E}$ be the event that EXP does not output *fail*. When $\mathcal{E}$ happens we have $P_j \notin \mathcal{L}_{\text{EXP}}$ and $P_{j+1}, P'_{j+1} \in \mathcal{L}_{\text{EXP}}$. Therefore, if EXP outputs $\left(g, h, \hat{T}, v, r, r'\right)$ we have that

$$h \neq g^{(2^{2^{\hat{T}}})} \qquad \text{and} \qquad (g^r v)^{(2^{\hat{T}})} = v^r h \qquad \text{and} \qquad (g^{r'} v)^{(2^{\hat{T}})} = v^{r'} h. \tag{4}$$

Re-arranging terms of the two equalities on the right we get

$$\left(g^{(2^{\hat{T}})}/v\right)^r = h/v^{(2^{\hat{T}})} \qquad \text{and} \qquad \left(g^{(2^{\hat{T}})}/v\right)^{r'} = h/v^{(2^{\hat{T}})}. \tag{5}$$

Dividing the left equality by the right we obtain

$$\left(g^{(2^{\hat{T}})}/v\right)^{r-r'} = 1.$$

Hence $\mu := g^{(2^{\hat{T}})}/v$ is an element of order at most $d := |r - r'|$ in $\mathbb{G}$, where $0 < d < 2^{\lambda}$.

We show that $\mu \neq 1$. Suppose towards a contradiction that $\mu = 1$. Then $v = g^{(2^{\hat{T}})}$ which implies by (5) that $h = v^{(2^{\hat{T}})}$. But these two equalities together imply that $h = g^{(2^{2\hat{T}})}$, and this contradicts the left side of (4). Hence $\mu \neq 1$, and we know that $\mu^d = 1$ where $d := |r - r'| < 2^{\lambda}$.

To summarize, our adversary $\mathcal{B}$ runs experiment EXP, and if it does not fail, that is, if event $\mathcal{E}$ happens, then $\mathcal{B}$ outputs a pair (μ, d) that breaks the low order assumption. It remains to determine how likely is event $\mathcal{E}$ to happen. Fortunately this has already been worked out in the generalized forking lemma of Bellare and Neven [BN06, Lemma 1]. An application of their lemma shows that if $\mathcal{A}$ succeeds in fooling the verifier with probability ϵ, then event $\mathcal{E}$ happens with probability at least $(\epsilon^2/t) - (\epsilon/2^{\lambda})$, as required. □

6 Concrete groups

Next, we briefly discuss two families of finite abelian groups of unknown order in which the low order and adaptive root assumptions are believed to hold.

6.1 The RSA group

Let *GGen* be an algorithm that outputs an odd integer N with an unknown factorization. Computing the order of the multiplicative group $\mathbb{G} := (\mathbb{Z}/N)^*$ is as hard as factoring N, and therefore $\mathbb{G}$ can be used as a group of unknown order. However, the low order assumption is trivially false in such groups because $(-1) \in \mathbb{Z}/N$ is an element of order two. Fortunately, this is the only impediment and it is easily corrected by instead working in the group $\mathbb{G}^+ := \mathbb{G}/\{\pm 1\}$. Elements in this group are represented as cosets $\{x, -x\}$ for $x \in \mathbb{G}$ and multiplication is defined as $\{x, -x\} \cdot \{y, -y\} = \{xy, -xy\}$. Of course when computing in this group it suffices to represent a coset $\{x, -x\}$ by a single number, either x or $-x$, whichever is in the range $[0, N/2)$. The low order assumption is believed to hold for a group generator *GGen* that generates such groups.

Pietrzak [Pie19] proves soundness of his proof of exponentiation (Figure 2) for integers N that are a product of strong primes. Recall that a prime number p is *strong* if $(p-1)/2$ is also a prime number. If $N = p \cdot q$ is a product of distinct strong primes then the group $\mathbb{G}_{\text{QR}}$ of quadratic residues in $(\mathbb{Z}/N)^*$, namely the group $\mathbb{G}_{\text{QR}} := \{z^2 : z \in (\mathbb{Z}/N)^*\}$, contains no elements of low order other than 1. Hence, the low order assumption holds unconditionally in $\mathbb{G}_{\text{QR}}$. Pietrzak proved unconditional soundness of the protocol when used in this group $\mathbb{G}_{\text{QR}}$.

Theorem 1 shows that by relying on the low order assumption in the multiplicative group $\mathbb{G}^+ := \mathbb{G}/\{\pm 1\}$ we are able to prove soundness even when N is not a product of strong primes. We note that an inconvenience in using $\mathbb{G}_{\text{QR}}$ is that checking membership in this group is difficult, and this complicates Pietrzak's protocol. In contrast, checking membership in $\mathbb{G}^+$ is easy and therefore the protocol in Figure 2 can be used as is. Moreover, Seres and Burcsi [SB20] point out that when $\gcd(p-1, q-1) = 2$, a low order element $\mu \neq \pm 1$ in $\mathbb{Z}/N$ can be used to factor N by computing $\gcd(\mu - 1, N)$. Hence, for moduli $N = pq$ where $\gcd(p-1, q-1) = 2$, the low order assumption is equivalent to the hardness of factoring assumption.

The difficulty with the group $(\mathbb{Z}/N)^*$ is that the group generator *GGen* must be trusted not to reveal the factorization of N. One can instead make *GGen* use *public* randomness to choose a sufficiently large N so that factoring N is hard. However the resulting N must

be so large as to be impractical. The problem is that arithmetic modulo such a large modulus presents too many opportunities for parallelizing the squaring algorithm.

6.2 The class group of an imaginary quadratic number field

To avoid the trusted setup problem one can instead use the class group of the number field $\mathbb{Q}(\sqrt{-p})$, where p is a prime $p \equiv 3 \bmod 4$, as suggested by Wesolowski [Wes19, Wes20]. This class group has odd order and computing its order is believed to be difficult when p is large. See [BH01] for a discussion on the choice of cryptographic parameters for such groups. Concretely, the group generator $GGen(1^\lambda)$ outputs a prime p from which the class group of $\mathbb{Q}(\sqrt{-p})$ is completely specified. Several papers have proposed efficient ways to hash into such groups [SBK24, CLR24].

The Cohen-Lenstra heuristics [CL84] suggest that for imaginary quadratic number fields:

- the frequency of fundamental discriminants for which the odd part of the class group is cyclic is about 97.6%,
- the frequency $f(d)$ of fundamental discriminants for which the order of the class group is divisible by d is approximately:

$$f(3) = 44\%, \qquad f(5) = 24\%, \qquad f(7) = 16\%.$$

These heuristics suggest that the class group is often cyclic, but often contains elements of small odd order. The question is how hard is it to find an element of small odd order, if one exists?

Shanks [Sha69] gives an example where finding an element of low order is easy. Specifically, when $p = 2^n - 1$ is a Mersenne prime, Shanks constructs an explicit element of order $n-2$ in $\mathbb{Q}(\sqrt{-p})$. Belabas, Kleinjung, Sanso, and Wesolowski [BKSW20] build on this and construct other examples of weak discriminants where the low order assumption is false.

An approach to finding low order elements in class groups. The low order assumption in the class group of an imaginary quadratic extension has not been studied much, and is a fascinating avenue for future work. For example, can we find an element of order three if one exists?

We mention one possible avenue for attack based on the work of Ellenberg and Venkatesh [EV07]. Let I be an ideal of order 3 in the class group of $\mathbb{Q}(\sqrt{-p})$. Then I^3 is principle meaning that $I^3 = \langle a + b\sqrt{-p} \rangle$ for some $a, b \in \mathbb{Z}$. Then the ideal norm $N(I)$ satisfies $N(I)^3 = N(I^3) = a^2 + pb^2$. Setting $z = N(I)$ we see that the existence of an ideal of order three implies an integral point (a, b, z) on the surface

$$z^3 = a^2 + pb^2 \tag{6}$$

where

$$|z| \leq \sqrt{p}, \qquad |a| \leq p^{3/4}, \qquad |b| \leq p^{1/4}. \tag{7}$$

The first inequality follows from the fact that we can take I to be a reduced ideal in the class group. The second and third inequalities follow from the first.

If we could find an integral point (a, b, z) satisfying (7) on the surface (6), where z is not a perfect square, then we will likely break the low order assumption in the class group of $\mathbb{Q}(\sqrt{-p})$. We want a point (a, b, z) where $|z| \leq \sqrt{p}$ is not a perfect square to ensure that z is not the norm of a principal ideal. Fortunately for this paper, the bounds (7) are out of reach for Coppersmith's method for finding low-norm integral points on curves and surfaces [Cop97]. However, perhaps Coppersmith's method can be tuned specifically for this family of surfaces? We leave that for future work.

7 Post-quantum secure VDFs

We conclude by pointing out that the two VDFs surveyed here are insecure against an adversary who has access to a sufficiently large quantum computer. A quantum computer can efficiently calculate the order of the group $\mathbb{G}$ using Shor's period finding algorithm and break the VDF.

Some of the VDFs studied in [BBBF18], such as the one based on iterated hashing, are plausibly post-quantum secure. Buterin [But18] develops and implements this construction using a combination of the MiMC hash function and a STARK proof. A similar approach, based on the ZKBoo proof system, was suggested by Tan et al. [TSL+23]. This approach was further developed in [DGMV20, MSW20].

A very different approach uses sequential composition of isogenies on elliptic curves. An interesting proposal for a VDF from isogenies is reported in [DMPS19]. The construction is not designed to be post-quantum secure, and requires pairings and a trusted setup. Shani [Sha19] gave a construction that avoids pairings. Ahrens and Zumbrägel [AZ23] propose a different scheme using isogenies that may be post-quantum secure.

Additional constructions for a post-quantum secure VDF is a fruitful avenue for further research.

Acknowledgments

We thank Krzysztof Pietrzak, Benjamin Wesolowski, and Justin Drake for their helpful comments about this writeup.

A A parallel algorithm for quotients in the exponent

Let us briefly describe how to compute $\pi = g^{\lfloor 2^T/\ell \rfloor} \in \mathbb{G}$ in parallel, as needed to speed up Wesolowski's succinct argument. We can accelerate the prover's time to compute π by a factor of s using s processors. We do so by storing s group elements as the prover evaluates the VDF. Taking $s = 100$ seems reasonable in practice.

So, let $b := \lfloor T/(s-1) \rfloor$. As the prover computes the VDF it stores the following s group elements as they are encountered along the way:

$$u_0 = g, \quad u_1 = g^{(2^b)}, \quad u_2 = g^{(2^{2b})}, \quad \ldots, \quad u_{s-1} = g^{(2^{(s-1)b})}.$$

Next, our algorithm to compute π uses the following subroutine exp, which is essentially the same as the algorithm from Section 2.1. Here $0 \le d < \ell$ is an additional input parameter.

$\exp(h, t, d, \ell)$: // *output* $a = h^{\lfloor d2^t/\ell \rfloor} \in \mathbb{G}$
  $a \leftarrow 1 \in \mathbb{G}, \quad r \leftarrow d \in \{0, \ldots, \ell-1\}$
  repeat t times:
    $q \leftarrow \lfloor 2r/\ell \rfloor \in \{0, 1\}, \quad r \leftarrow (2r \bmod \ell) \in \{0, \ldots, \ell-1\}$
    $a \leftarrow a^2 \cdot h^q \in \mathbb{G}$
  output a // *this* a *is equal to* $h^{\lfloor d2^t/\ell \rfloor} \in \mathbb{G}$

Using subroutine exp we can compute $\pi = g^{\lfloor 2^T/\ell \rfloor} \in \mathbb{G}$ in time $O(T/s)$ using the algorithm in Figure 3. The algorithm starts by quickly computing all the remainders needed for the s steps of long division, and then runs these s steps in parallel. The bulk of the work happens on lines (1) and (2), where each call to the function exp requires b sequential squarings. The point is that all the calls to exp can be processed in parallel. The algorithm needs enough memory to store only s group elements.

input: g, T, ℓ, s as well as $u_i = g^{(2^{ib})} \in \mathbb{G}$ for $i = 0, \ldots, s-1$
where $s > 1$ and $T > s(s-2)$
output: $\pi := g^{\lfloor 2^T/\ell \rfloor} \in \mathbb{G}$ computed with s-way parallelism in time $O(T/s)$

$b \leftarrow \lfloor T/(s-1) \rfloor$ *// batch size*

// compute remainders by a quick sequential process
$r_0 \leftarrow (2^{(T \bmod b)} \bmod \ell) \in \{0, \ldots, \ell-1\}$
for $i = 1, \ldots, s-1$:
$r_i \leftarrow \left(2^b \cdot r_{i-1} \bmod \ell\right) \in \{0, \ldots, \ell-1\}$

// compute π in parallel
(1) $\pi_0 \leftarrow \exp\big(u_{s-1},\ (T \bmod b),\ 1,\ \ell\big)$ *// compute* $\pi_0 \leftarrow (u_{s-1})^{\lfloor 2^{(T \bmod b)}/\ell \rfloor} \in \mathbb{G}$
for $i = 1, \ldots, s-1$:
(2) $\pi_i \leftarrow \exp\big(u_{s-1-i},\ b,\ r_{i-1},\ \ell\big)$ *// compute* $\pi_i \leftarrow (u_{s-1-i})^{\lfloor r_{i-1} \cdot 2^b/\ell \rfloor} \in \mathbb{G}$

output $\pi \leftarrow \prod_{i=0}^{s-1} \pi_i$

Figure 3: Computing $\pi := g^{\lfloor 2^T/\ell \rfloor} \in \mathbb{G}$

References

[AFK23] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs (extended version). *Journal of Cryptology*, 36(4):36, October 2023. doi:10.1007/s00145-023-09478-y.

[AGL+23] Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 542–571. Springer, Heidelberg, May 2023. doi:10.1007/978-3-031-31371-4_19.

[AVD22] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. Efficient verification of the wesolowski verifiable delay function for distributed environments. Cryptology ePrint Archive, Report 2022/520, 2022. https://eprint.iacr.org/2022/520.

[AZ23] Knud Ahrens and Jens Zumbrägel. DEFEND: towards verifiable delay functions from endomorphism rings. *IACR Cryptol. ePrint Arch.*, page 1537, 2023. URL: https://eprint.iacr.org/2023/1537.

[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. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96884-1_25.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_20.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_24.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356. Association for Computing Machinery, January 2016. doi:10.1145/2840728.2840745.

[BH01] Johannes Buchmann and Safuat Hamdy. A survey on IQ cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2001. doi:10.1515/9783110881035.1.

[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 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84259-8_5.

[BKSW20] Karim Belabas, Thorsten Kleinjung, Antonio Sanso, and Benjamin Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020. https://eprint.iacr.org/2020/1310.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399. ACM Press, October / November 2006. doi:10.1145/1180405.1180453.

[BN22] Joseph Bonneau and Valeria Nikolaenko. Public randomness and randomness beacons, 2022. URL: https://a16zcrypto.com/posts/article/public-randomness-and-randomness-beacons/.

[BS22] Dan Boneh and Victor Shoup. *A graduate course in applied cryptography, version 0.6*. Cambridge, 2022. URL: cryptobook.us.

[But18] Vitalik Buterin. STARKs, part 3: Into the weeds, 2018. https://vitalik.ca/general/2018/07/21/starks_part_3.html.

[CL84] Henri Cohen and Hendrik W Lenstra. Heuristics on class groups of number fields. In *Number Theory Noordwijkerhout 1983*, pages 33–62. Springer, 1984.

[CLR24] Kostas Kryptos Chalkias, Jonas Lindstrøm, and Arnab Roy. An efficient hash function for imaginary class groups. Cryptology ePrint Archive, Paper 2024/295, 2024. https://eprint.iacr.org/2024/295. URL: https://eprint.iacr.org/2024/295.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, September 1997. doi:10.1007/s001459900030.

[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. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78375-8_15.

[DGMV20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 65–84. Springer, Heidelberg, September 2020. doi:10.1007/978-3-030-57990-6_4.

[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. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34578-5_10.

[EV07] Jordan Ellenberg and Akshay Venkatesh. Reflection principles and bounds for class group torsion. *International Mathematics Research Notices*, 2007, 2007. doi:10.1093/imrn/rnm002.

[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. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[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. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15979-4_13.

[JMRR21] Samuel Jaques, Hart Montgomery, Razvan Rosie, and Arnab Roy. Time-release cryptography from minimal circuit assumptions. In *Progress in Cryptology – INDOCRYPT 2021*, volume 13143 of *Lecture Notes in Computer Science*, pages 584–606. Springer, 2021. doi:10.1007/978-3-030-92518-5_26.

[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.

[LM19] Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 530–560. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_19.

[LMPR23] Gaëtan Leurent, Bart Mennink, Krzysztof Pietrzak, and Vincent Rijmen. Analysis of MinRoot: Public report, 2023. URL: https://crypto.ethereum.org/events/minrootanalysis2023.pdf.

[LW17] Arjen K Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *International Journal of Applied Cryptography*, 3(4):330–343, 2017. doi:10.1504/IJACT.2017.089354.

[Mao01] Wenbo Mao. Timed-release cryptography. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 342–358. Springer, Heidelberg, August 2001. doi:10.1007/3-540-45537-X_27.

[MLQ23] Liam Medley, Angelique Faye Loe, and Elizabeth A. Quaglia. SoK: Delay-based cryptography. In *CSF 2023: IEEE 36th Computer Security Foundations Symposium*, pages 169–183. IEEE Computer Society Press, July 2023. doi:10.1109/CSF57540.2023.00028.

[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. Schloss Dagstuhl, July 2020. doi:10.4230/LIPIcs.ICALP.2020.83.

[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. LIPIcs, January 2019. doi:10.4230/LIPIcs.ITCS.2019.60.

[RS20] Lior Rotem and Gil Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 481–509. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56877-1_17.

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

[SB20] István András Seres and Péter Burcsi. A note on low order assumptions in RSA groups. Cryptology ePrint Archive, Report 2020/402, 2020. https://eprint.iacr.org/2020/402.

[SBK24] István András Seres, Péter Burcsi, and Péter Kutas. How (not) to hash into class groups of imaginary quadratic fields? Cryptology ePrint Archive, Paper 2024/034, 2024. https://eprint.iacr.org/2024/034. URL: https://eprint.iacr.org/2024/034.

[Sha69] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Sympos. Pure Math.*, volume 29, page 415–440. Amer. Math. Soc., 1969. doi:10.1090/pspum/020/0316385.

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

[TSL+23] Teik Guan Tan, Vishal Sharma, Zengpeng Li, Pawel Szalachowski, and Jianying Zhou. ZKBdf: A ZKBoo-based quantum-secure verifiable delay function with prover-secret. In *Applied Cryptography and Network Security Workshops – ACNS satellite workshops 2023*, volume 13907 of *Lecture Notes in Computer Science*, pages 530–550. Springer, 2023. doi:10.1007/978-3-031-41181-6_29.

[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. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17659-4_13.

[Wes20] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, October 2020. doi:10.1007/s00145-020-09364-x.