Blind Verifiable Delay Functions

Charlotte Hoffmann and Krzysztof Pietrzak

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

Abstract

A verifiable delay function (VDF), introduced by Boneh et al. [CRYPTO'18], on input (x, T) produces an output (y, π) where computing y requires T inherently sequential steps, and π is a proof certifying correctness. VDFs have found numerous applications, and many of these rely on the assumption that honest parties can evaluate the VDF nearly as fast as adversaries.

To support this assumption, significant effort has been invested in developing dedicated VDF hardware (ASICs). However, such devices are expensive and produced in limited quantities, leaving only a small number of parties with access to fast VDF evaluation. A natural workaround is to offer "VDFs as a service," where a server equipped with specialized hardware evaluates instances on behalf of clients. This approach, however, is unsuitable when the VDF input must remain private.

In this work, we introduce and construct *blind* VDFs, enabling secure VDF outsourcing without revealing the input. Inspired by blind signatures, our protocol allows a client to transform an input x into a blinded instance α, which is evaluated by the server. The server returns a result β, from this and its local state, the client can unblind to recover the correct output (y, π), while the server learns nothing about x.

We realize this notion by constructing a blinded version of Pietrzak's VDF [ITCS'19]. In the original scheme, computing y requires T sequential steps, while generating the proof π can be done in $O(T/S)$ parallel time using $O(S)$ space (e.g., $O(\sqrt{T})$ time and space). In our blind variant, the server performs the T sequential steps and sends $O(S)$ data to the client, who completes the unblinding in $O(T/S)$ time. By allowing slightly more interaction, we obtain a protocol where the client's work is small, while the server performs essentially the same computation as in the unblinded VDF.

1 Introduction

A central challenge in distributed systems and cryptographic protocols is the ability to enforce some delay. Many applications, such as randomness beacons, fair ordering, leader election, timed commitments, and anti-spam mechanisms, require guarantees that a certain amount of *real time* must elapse between an input and its corresponding output.

Timed Cryptography. While one can't cryptographically prove that time has passed, Rivest, Shamir and Wagner invented the field of "timed cryptography" [25] introducing *time-lock puzzles*, where one must perform an inherently sequential computation to decrypt a message. "Inherently sequential" means that there should be basically no way to speed up the computation when given many processors.

Another timed-cryptography primitive was later introduced by Mahmoody, Moran and Vadhan [22], who define and construct *proofs of sequential work* (PoSW), where a prover on input a challenge x and a time parameter T, must perform T sequential steps (in their work, a "step" is an evaluation of a hash function) to compute an efficiently and publicly verifiable proof that a sequential computation of almost T steps was performed.

Verifiable Delay Functions. Many applications, like randomness beacons or leader election, basically require a PoSW were one additionally has some *unique* value that is only learned at the end of the sequential computation. Boneh, Bonneau, Bünz and Fisch [3] solve this challenge by introducing and constructing *Verifiable delay functions* (VDFs). In their construction, one performs a simple sequential computation, and at the same time computes a succinct, incrementally verifiable proof certifying that this computation was preformed correctly. Computing this proof is computationally costly, but parallelizable, and requires the prover to have a GPU.

Two much simpler constructions were then proposed by Wesolowski [29] and Pietrzak [24]. In their construction, they use basically the same sequential function as in the time-lock puzzles paper [25], namely squaring in groups of unknown order, like RSA groups: on input a group defined by the modulus N, an input x and time parameter T, the VDF computation outputs $y = x^{2^T} \bmod N$, which can be computed by squaring x sequentially T times, and a "proof of exponentiation" (PoE) certifying that y has been correctly computed. [29] and [24] differ in how this PoE looks like.

VDF applications and hardware. Since their introduction, VDFs have been proposed as a building block in a variety of applications, including public randomness beacons, blockchain consensus protocols, fair sequencing mechanisms, rate-limiting systems, and timed cryptographic protocols. In many of these settings, the delay parameter must be sufficiently large to achieve meaningful security or fairness guarantees. As a result, evaluating a VDF can be computationally expensive. More importantly, the efficiency, and sometimes even security, of those schemes relies on the assumption that honest parties can evaluate the VDF not much slower than a potential adversary. For this reason, there's been significant investments into building custom hardware, concretely, Application Specific Circuits (ASICs), to compute VDFs. The rationale is that after this investments, which are in the tens of millions of dollars, further significant improvements would either be impossible, or extremely costly.[1]

VDFs as a service. Unfortunately, such ASICs are quite expensive and produced in low numbers. This motives the concept of *VDFs as a service*. In this model, servers with access to specialized hardware, provide VDF evaluation as an external resource. A client submits an instance (x, T) to the service, which evaluates the VDF and returns the output together with a proof of correctness. The *timelords*[2] in the Chia blockchain can be seen in this way, another example is StarkWare's VeeDo service.[3]

Such a straight forward VDF-as-a-service model introduces an important privacy concern: the server learns the input to every VDF instance it evaluates, which may encode sensitive information. In many applications, the input may encode sensitive or strategic information. For example, it may correspond to a sealed bid in an auction [30] or some anti-MEV system using VDFs for transaction ordering [28]. Those privacy concerns pose a natural open question:

Is it possible to efficiently outsource VDF computations to a server in a way that the server learns nothing about the VDF input?

Another concern that arises when outsourcing VDFs is privacy of the client and ownership of the results. In many applications, the client will publish the results of the VDF computation at some point. Ideally, we would like that the server does not recognize that what the client has published corresponds to the VDF output is has computed. In particular, we do not want it to able able to watermark the VDF output it has produced. To prevent this, the second question we ask in this work is:

Is it possible to outsource VDF computation to a server in a way that the server cannot recognize the VDF output is has computed?

[1] For more information, check the VDF alliance https://www.vdfalliance.org/ page, which "is a collection of academic, non-profit, and corporate collaborators building open source hardware for the blockchain ecosystem."

[2] https://docs.chia.net/chia-blockchain/architecture/timelords/

[3] https://medium.com/starkware/presenting-veedo-e4bbff77c7ae

1.1 Our Contribution

In this work we answer both questions in the affirmative by showing that it is possible to outsource VDF computations in a completely blind manner. To this end, we give the first formal definition of blind VDFs. We also present a construction that satisfies this definition. We give three different variants of this construction with different trade-offs in terms of round and client complexity. In the group of signed quadratic residues, our constructions achieve *statistical* blindness. In general groups of hidden order we prove *computational* blindness based on a discrete-log type assumption.

Defining blind VDFs. Our first contribution is the formalization of the requirements for blind verifiable delay functions. A standard VDF is defined by three algorithms, a Setup which outputs public parameters, we'll ignore this in this introduction. Then we have $\mathsf{Eval}(x, T) \to (y, \pi)$ which maps the input x and time T to an output/proof tuple (y, π), which can be verified using $\mathsf{Verify}(x, y, \pi) \to \mathsf{reject/accept}$.

When using VDF as a service, the server would run $\mathsf{Eval}(x, T)$ and thus learn x. For a blind VDF, we assume three additional functions $\mathsf{Blind}, \mathsf{BlindEval}, \mathsf{Unblind}$. When using VDF as a service, the client $\mathcal{C}$, who would like to learn the VDF output $\mathsf{Eval}(x, T) \to (y, \pi)$, can first use $\mathsf{Blind}(x) \to (\mathsf{ust}, \alpha)$ to blind x. It stores the "unblinding state" ust and send the blinded input α to the server $\mathcal{S}$. The server will run $\mathsf{BlindEval}(\alpha) \to \beta$, and send the output β back to $\mathcal{C}$, who can then run $\mathsf{Unblind}(\mathsf{ust}, \beta)$ to learn (y, π).

We want $\mathcal{C}$ to be able to blind the input and compute the pair (y, π) with complexity much less than T. The blindness property should ensure that even if the server $\mathcal{S}$ sees the tuple (x, y, π) it should not be able to recognize that this tuple corresponds to an instance it was asked to compute in the past. In particular, (x, y, π) should look indistinguishable from an instance that was computed using the regular Eval.

We capture this intuition in the blindness definition with three conditions.

1. First, we require that a blinded input α reveals no information about the actual input x.

2. Then, we require that the unblinded outputs are indistinguishable from correctly evaluated ones. This means that the output $(y, \pi) \leftarrow \mathsf{Eval}(x, T)$ of the evaluation algorithm should be indistinguishable from the output of the unblinding algorithm. This must hold even if the server $\mathcal{S}$ is malicious and does not honestly execute $\mathsf{BlindEval}$. We don't have to worry about the y part of the output, as by soundness of a classic VDF it should be infeasible to come up with a proof for any output other than the correct y. However, the proof π need not be unique and hence we could imagine an adversarial server $\mathcal{S}$ that computes a malicious answer β, so that after unblinding, the proof π reveals something. This second condition ensures that this is not posssible.

3. Note that a malicious server can always make $\mathsf{Unblind}$ fail by simply sending some garbage message. We require that the probability of failure of the unblinding algorithm should not depend on the (unblinded) input. If a malicious server could find a way to correlate the probability of unblinding failing with the actual inputs, this would bias the distribution of unblinded values. Since we want the server to be completely oblivious to the VDF tuples it helped to compute, we prevent this scenario by requiring this third condition.

It's not immediately clear that the third requirement isn't already implied by the first two. To see this, consider any blind VDF with algorithms Blind, $\mathsf{BlindEval}$, $\mathsf{Unblind}$ satisfying all three properties. We can artificially change it into another construction Blind, $\mathsf{BlindEval}'$, $\mathsf{Unblind}'$ that will satisfy all properties except the third. For this, let $\mathsf{BlindEval}'$ be identical to $\mathsf{BlindEval}$, except that it adds an extra bit b, that is always set to 0, to its message. $\mathsf{Unblind}'$ behaves like $\mathsf{Unblind}$ if $b = 0$, but if $b = 1$ it will output $\perp$ in the case that the unblinded x starts with 0. If the server is honest, this change makes no difference. But a malicious server can set $b = 1$, and thus prevent unblinding whenever x starts with 0.

Blind VDFs vs. blind signatures. The notion of blind VDFs is inspired by blind signatures introduced by Chaum [6]. Chaum's signatures and our VDF both are defined over an RSA group, which has rich algebraic structure that can be used for blinding and unblinding. A fundamental difference is the fact

that, unlike for blind signatures, for blind VDFs the server has no secret. This is reflected in the security definitions. While for blind signatures there are security requirements for both, the client and server, with blind VDFs we only consider security for the client as outlined above. Informally, we don't want the server to learn anything beyond the fact that they solved some VDF instance with time parameter T.

Let us stress that the definition of blind VDFs does not imply that the public parameters are hidden from the server. In our construction those include a description of the group (like the RSA modulus N), which the server will receive.

Blinding the VDF input. Our blind VDF constructions are based on Pietrzak's VDF [24]. In Pietrzak's VDF the delay function is the repeated squaring function in a group of hidden order $\mathbb{G}$. Let T be the time parameter of the VDF. The input is a group element x and the output is the unique group element $y = x^{2^T}$ together with a proof π that certifies the result. To make this VDF blind, we first need to find a way to transform the input element x into a blinded input x_{blind} such that y can later be obtained from the element $y_{\mathsf{blind}} = x_{\mathsf{blind}}^{2^T}$. The first idea that comes to mind is inspired by blind RSA signatures [5]: Choose some random exponent r and compute x^r. Unfortunately, this approach does not work on its own since we cannot extract y in this case: Assume that on input x^r the server $\mathcal{S}$ computes $\tilde{y} = y^r = (x^r)^{2^T}$. Given r and $\tilde{y}$, to find the element y one would need to take the r-th root of $\tilde{y}$, which is believed to be a hard problem in hidden order groups. Further, in contrast to the RSA signature setting, we don't have a trapdoor since we don't want anyone to know the group order. Hence, it is also not possible to choose the exponent r in a clever way depending on the order of the group.

In this work we achieve blindness of the input as follows: We pick a random group element $a \in \mathbb{G}$ and include a and the element $a' = a^{2^T}$ in the public parameters. Then, we blind the input x by choosing a random exponent r from an appropriate set and compute $x_{\mathsf{blind}} = xa^r$. When given this to the server $\mathcal{S}$ as input, it returns $y_{\mathsf{blind}} = x_{\mathsf{blind}}^{2^T} = y(a^r)^{2^T}$. The client $\mathcal{C}$ can unblind by computing $y = y_{\mathsf{blind}}/(a')^r$. Note that if a is a generator of the group and the size of the randomness space of r is roughly as big as the group order, then x_{blind} is statistically close to a uniformly random group element. Jumping ahead, we prove in Section 7 that in non-cyclic groups that don't have single generators, the element x_{blind} is computationally indistinguishable from random based on a discrete-log type assumption. We have thus managed to blind the input x in such a way that $\mathcal{S}$ learns nothing about it and $\mathcal{C}$ can efficiently compute the output element y from the output of the server.

Obtaining a proof from the server. We are already half way to construct a blind VDF. The next thing we need is that $\mathcal{C}$ is able to construct the proof π from the output of $\mathcal{S}$. Let's first assume that $\mathcal{S}$ does not only output y_{blind} but also a proof π_{blind} that certifies that $y_{\mathsf{blind}} = (x_{\mathsf{blind}})^{2^T}$. One simple way for the client to obtain a valid proof for the statement $y = x^{2^T}$ would be to set $\pi = (x_{\mathsf{blind}}, y_{\mathsf{blind}}, \pi_{\mathsf{blind}}, r, a, a')$. Then, to check the proof π, one could verify that x_{blind} and y_{blind} are blinded versions of x and y and check the validity of π_{blind}. Unfortunately, this does not satisfy our blindness definition since we don't want $\mathcal{S}$ to be able to recognize a VDF tuple that it has computed itself. If $\mathcal{C}$ publishes the server output, this property cannot hold.

We show how to obtain a proof π for the statement $x^{2^T} = y$ that is indistinguishable from the proof output by the evaluation algorithm Eval of the VDF. In fact, since Pietrzak's VDF satisfies soundness with *unique* proofs (see Definition 3), we even achieve that the proof π is *the same* proof as output by Eval. We give three variants of our construction with different trade-offs between the round and the client's computational complexity.

1. Our *basic construction* uses the fact that, for any integer $d \leq \log T$, Pietrzak's VDF proof can be computed in time $2^d + T/2^d$ using 2^d elements stored during the initial VDF computation. Those elements are of the form $x^{2^{i \cdot T/2^d}}$ for $i \in \{1, \ldots, 2^d\}$. In the basic construction we chose $d = 1/2 \log T$ and we add the elements $a^{2^{i \cdot \sqrt{T}}}$ for $i \in \{1, \ldots, \sqrt{T}\}$ to the public parameters. Then, on input x_{blind}, we let the server output y_{blind} together with the elements $x_{\mathsf{blind}}^{2^{i \cdot \sqrt{T}}}$ for $i \in \{1, \ldots, \sqrt{T}\}$. The client can

unblind these elements by computing

$$x^{2^{i\cdot\sqrt{T}}} = x_{\mathsf{blind}}^{2^{i\cdot\sqrt{T}}} / (a^{2^{i\cdot\sqrt{T}}})^r$$

for all i. Then the client can construct the proof π from these elements in time roughly $2\sqrt{T}$.

2. Note that the basic construction already reduces the complexity of the client from $O(T)$ to $O(\sqrt{T})$. In our *client-optimized construction* we show how to bring the client complexity down to $O(\log T)$ by introducing interaction between the client and the server. In this construction the public parameters consist of $\log T + 2$ elements of the form $\{u, u^{2^T}, u^{T/2}, u^{T/2^2}, \ldots, u^2\}$. In Pietrzak's VDF the proof $\pi = (\pi_1, \ldots, \pi_{\log T})$ basically consists of $\log T$ VDF instances with time parameters $T/2, T/2^2, \ldots, 2$. In our client-optimized construction we let the client freshly blind each of these instances and send it to the server as a VDF instance. Hence, all the client has to do is blind and unblind proof elements for $\log T$ rounds so its complexity is $O(\log T)$.

3. The client-optimized construction achieves optimal client complexity at the cost of introducing $\log T$ rounds of interaction. In our *hybrid construction* we show how to significantly reduce the number of rounds of interaction while keeping the client complexity low. This approach is a combination of our other two constructions. For some $s > 1$, in the hybrid construction we let the server send $T^{1/s}$ elements to the client in every round. From these elements the client can compute $(\log T)/s$ proof elements. Then the client sends a new instance to the server with a time parameter reduced by a factor $T^{1/s}$ and recurses. This approach requires s rounds of interaction and the client complexity is $O(T^{1/s})$. In practice, one would choose $s = 4$ or $s = 8$ depending on the resources of the client and the number of rounds of interaction that is acceptable for the application. We give a detailed comparison of the protocols in Section 8.

To sum up, we introduce the notion of blind VDFs and give a construction based on Pietrzak's VDF. This construction can be parameterized to achieve different trade-offs in round complexity vs. client computation. We believe that blind VDFs will further broaden the domains in which timed cryptography, and VDFs in particular, can play a role. In particular privacy sensitive applications which rely on a small gap in the execution speed between honest and malicious parties, but for which "VDF as a service" so far was not an option due to privacy concerns.

1.2 More Related Work on VDFs

Verifiable delay functions were introduced by Boneh et al. [3]. The first two practical VDFs were constructed concurrently by Wesolowski [29] and Pietrzak [24]; see [4] for a discussion of both. The delay function in both constructions is the iterated squaring function in groups of hidden order [25]. To achieve verifiability, Wesolowski and Pietrzak construct different proofs of exponentiation (PoEs). While Wesolowski's proof is shorter, Pietrzak's proof is more efficient to compute, and its security relies on a weaker assumption.

Later, Block et al. [2] presented the first statistically sound PoE in any group, which they use to build polynomial commitment schemes. However, the proofs in [2] are relatively large. The PoE in [15] is a variant of their protocol with lower complexity, but requires the ability to choose an exponent other than 2 (i.e., the output is x^{q^T} rather than x^{2^T} for a specific q).

VDFs using sequential functions other than iterated squaring have also been proposed. These include isogeny-based constructions [9, 27, 7], a permutation-polynomial-based construction [3], and constructions from hard lattice problems [20, 8, 23]. Freitag, Pass, and Sirkin [13] construct VDFs from any sequential function based on the polynomial hardness of LWE. Lombardi and Vaikuntanathan [21] investigate standard-model instantiations of the Fiat–Shamir heuristic in Pietrzak's VDF. Hoffmann et al. [17] construct a VDF from Lucas sequences, relying on a potentially weaker assumption than iterated squaring.

Several VDFs with additional functionality have also been proposed. Hoffmann and Pietrzak [18] show how to construct zero-knowledge VDFs and watermarkable VDFs from any PoE in hidden-order groups. Batching protocols for PoEs are given in [26, 14]. Tight VDFs, which allow the proof π to be computed almost

simultaneously with the output y, were studied in [10]. Continuous VDFs [11] allow one to continuously verify intermediate steps of the computation. Finally, [16] uses techniques from VDFs to certify the non-primality of giant Proth numbers, as used in the Great Internet Mersenne Prime Search.

2 Preliminaries

In the rest of this paper, we let λ denote a security parameter. We use the notation $[n] := \{1, \ldots, n\}$ for the set of all positive integers smaller than or equal to n. We say that an algorithm is *PPT* if it runs in probabilitic polynomial time.

2.1 Number Theory

We say that a group $\mathbb{G}$ is of *hidden order* if it is computationally infeasible to find (a multiple of) the group order.

Definition 1 (safe prime)**.** A prime number *safe* if it is of the form $p = 2p' + 1$, where p' is also a prime number.

Let $\mathbb{Z}_N^*$ denote the multiplicative group modulo N.

Definition 2. *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 [19] is defined as

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

where $\mid b \mid$ 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 $a \circ b :=\mid a \cdot b \mod N \mid$. 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 $+1$. Both QR_N and QR_N^+ are groups of hidden order for $N = pq$, where p and q are large primes. If p and q are *safe* primes, those groups do not contain any low order elements [12, 19].

2.2 Verifiable Delay Functions

Verifiable delay functions were defined by Boneh et al. [3].

Definition 3 (Verifiable Delay Function)**.** A *verifiable delay function* (VDF) with input space $\mathcal{X}$, output space $\mathcal{Y}$ and proof space $\mathcal{PF}$ is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ defined as follows.

- $\mathsf{Setup}(1^\lambda) \to \mathbf{pp}$ takes as input a statistical security parameter λ and outputs public parameters $\mathbf{pp}$.
- $\mathsf{Eval}(\mathbf{pp}, x, T) \to (y, \pi)$ takes as input the public parameters $\mathbf{pp}$, an input string x and the time parameter T and outputs the output string y and a proof π. Eval runs in sequential time $T + o(T)$.
- $\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi) \to \mathsf{accept}/\mathsf{reject}$.

We want the VDF to satisfy the following properties.

- Completeness: For $\mathsf{Eval}(\mathbf{pp}, x, T) = (y, \pi)$, the algorithm $\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi)$ outputs accept.
- Sequentiality: No two stage PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_2$ is running in sequential time T', that is sufficiently smaller than T, can compute the correct VDF output.

$$\Pr\left[\begin{array}{c} \tilde{y} = y \text{ where} \\ (y, \pi) = \mathsf{Eval}(\mathbf{pp}, x, T) \end{array} \middle| \begin{array}{l} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{st} \leftarrow \mathcal{A}_1(\mathbf{pp}) \\ x \leftarrow_\$ \mathcal{X} \\ \tilde{y} \leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}, x) \end{array}\right] = \mathbf{negl}(\lambda)$$

$\mathsf{Setup}(1^\lambda, T) \to \mathbf{pp} = \mathbb{G}$, where $\mathbb{G}$ is a finite abelian group $\mathbb{G}$ of unknown order and λ is a statistical security parameter.

$\mathsf{Eval}(\mathbf{pp}, x, T) \to (y, \pi)$, where $y = x^{2^T}$ and π is a PPoE.

$\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi) \to$ `accept/reject` checks if $\mathsf{PPoE}(\mathbf{pp}, x, y, T)$ is accepting.

Figure 1: Pietrzak's VDF construction [24] using PPoE as in Fig.(2)

Input: $(\mathbb{G}, x, y, T)$, where $\mathbb{G}$ is a group of hidden order, $x \in \mathbb{G}$, $T = 2^t$ for some $t \in \mathbb{N}$ and $y = x^{2^T}$.

Protocol $\mathsf{PPoE}(\mathbb{G}, x, y, T)$:

1. If $T = 1$, the verifier $\mathcal{V}$ checks if $x^2 = y$ and outputs accept or reject accordingly.
2. If $T > 1$,
   (a) $\mathcal{P}$ computes $\pi_1 := x^{2^{T/2}}$ and sends it to $\mathcal{V}$.
   (b) $\mathcal{V}$ samples $c \leftarrow [2^\lambda]$ uniformly at random and sends it to $\mathcal{P}$.
   (c) Both $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{x} = x^c \pi_1$ and $\tilde{y} = \pi_1^c y$.
   (d) $\mathcal{P}$ and $\mathcal{V}$ recursively call $\mathsf{PPoE}(\mathbb{G}, \tilde{x}, \tilde{y}, T/2)$.

Figure 2: Pietrzak's PoE [24] PPoE as an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

- Soundness: No PPT adversary $\mathcal{A}$ can come up with a proof for a wrong statement, i.e.,

$$\Pr\left[\begin{array}{r|l} \mathsf{Verify}(\mathbf{pp}, x, \tilde{y}, T, \tilde{\pi}) = \mathsf{accept} & \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \wedge\, \tilde{y} \neq y & (x, \tilde{y}, \tilde{\pi}) \leftarrow \mathcal{A}(\mathbf{pp}) \\ & (y, \pi) \leftarrow \mathsf{Eval}(\mathbf{pp}, x, T) \end{array}\right] = \mathbf{negl}(\lambda).$$

- Soundness with unique proofs: Some VDFs achieve a stronger notion called "soundness with unique proofs", where it's even hard to come up with a different proof $\tilde{\pi} \neq \pi$ for the same y, i.e., for any PPT adversary $\mathcal{A}$ we have

$$\Pr\left[\begin{array}{r|l} \mathsf{Verify}(\mathbf{pp}, x, \tilde{y}, T, \tilde{\pi}) = \mathsf{accept} & \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \wedge\, (\tilde{y}, \tilde{\pi}) \neq (y, \pi) & (x, \tilde{y}, \tilde{\pi}) \leftarrow \mathcal{A}(\mathbf{pp}) \\ & (y, \pi) \leftarrow \mathsf{Eval}(\mathbf{pp}, x, T) \end{array}\right] = \mathbf{negl}(\lambda).$$

The VDFs based on iterated squaring achieve verifiability via so called proofs of exponentiations.

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

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

Figure 1 shows Pietrzak's VDF. In Figure 2 shows the interactive version of Pietrzak's PoE, this PoE is defined for exponents of the form $e = 2^T$ and can be made non-interactive using the Fiat-Shamir heuristic. It can also be adapted to other exponents (see [15]) but we stick with the original form for simplicity. We will also assume that $T = 2^t$ is a power of 2 as this will simplify exposition. Let us note that the non-interactive version of the protocol is deterministic once the hash function is fixed because the prover does not use any

Input: $(\mathbb{G}, H, x, y, T, \{x^{2^{i \cdot T/2^d}}\}_{i \in [2^d - 1]})$, where $\mathbb{G}$ is a group of hidden order, H is a hash function, $x \in \mathbb{G}$, $T \in \mathbb{N}$, $y = x^{2^T}$ and d is an integer.

Proof computation:

1. Set $x_1 = x, y_1 = y, \pi_1 = x^{2^{T/2}}$ using the input element with index $i = 2^{d-1}$.
2. For $j \in [d-1]$ do
   (a) Compute $c_j = H(x_1, y_1, \pi_1, \ldots, \pi_j)$.
   (b) Compute $x_{j+1} = x_j^{c_j} \pi_j$ and $y_{j+1} = \pi_j^{c_j} y_j$.
   (c) Compute $\pi_{j+1} = x_{j+1}^{2^{T/2^{j+1}}}$ using the input elements with indices $i = s \cdot 2^{d-j-1}$, where s ranges over all positive odd integers less than 2^{d-j-1}.
3. For $k \in \{d, \ldots, \log T - 1\}$ do
   (a) Compute $c_k = H(x_1, y_1, \pi_1, \ldots, \pi_d, \ldots, \pi_k)$.
   (b) Compute $x_{k+1} = x_k^{c_k} \pi_k$ and $y_{k+1} = \pi_k^{c_k} y_k$.
   (c) Compute $\pi_{k+1} = x_{k+1}^{2^{T/2^{k+1}}}$ from scratch by performing $T/2^{k+1}$ sequential squarings.

Figure 3: Constructing Pietrzak's PoE [24] PPoE with 2^d stored elements.

randomness. Pietrzak shows in [24, Section 6.2.3] how to compute his PoE in roughly $2^d + T/2^d$ steps by storing the 2^d elements $\{x^{2^{i \cdot T/2^d}}\}_{i \in [2^d]}$ obtained during the computation of y. This means that by choosing $d = 1/2 \log T$ the proof can be computed in roughly $2\sqrt{T}$ steps. In Figure 3 we present the non-interactive proof of Pietrzak's PoE using 2^d stored values. We note that the computation of the proof elements in Step 2c can be parallelized but the computation of the proof elements in Step 3c cannot.

2.3 Assumptions

In this work we need the following assumptions. The iterated squaring assumption was first stated in [25] in the context of time-lock puzzles. It says that in groups of hidden order the element $y = x^{2^T}$ cannot be computed significantly[4] faster than in T sequential steps.

Definition 5 (Iterated squaring assumption)**.** Let $\mathsf{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a group $\mathbb{G}$ of hidden order. 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 $\mathsf{GGen}(1^\lambda)$, a random group element x and a time parameter T.
2. $\mathcal{A}$ wins if it outputs element $y = x^{2^T}$ in less than T steps.

The low order assumption, first formalized in [4], states that in certain groups of hidden order, it is infeasible to find a group element of small order. In the group QR_N^+, where $N = pq$ and p and q are safe primes, the assumption holds unconditionally because low order elements do not exist.

[4] Bernstein and Sorenson [1] proved that one can reduce the *sequential* time of computing x^{2^T} from T to $T/\log\log(T)$ using T^2 processors. Since in our setting using T^2 processors is infeasible, we will ignore the $\log\log T$ factor in the rest of the paper.

Definition 6 (low order assumption)**.** Let $\mathsf{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a group $\mathbb{G}$ of hidden order. The *low order assumption* holds for GGen if, for any PPT 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 $\mathsf{GGen}(1^\lambda)$.
2. $\mathcal{A}$ outputs a pair $(g, d) \in \mathbb{G} \times [2^\lambda]$.
3. $\mathcal{A}$ wins if $d \neq 1$ and $g^d = 1$.

The following assumption was first formalized in [18]. It states that in a group of hidden order it is infeasible to decide if there there exists a discrete log of bounded size between two elements. We will rely on this assumption for security of our blind VDF in certain groups.

Definition 7 (decisional discrete log assumption with small exponents)**.** Let $\mathsf{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a group $\mathbb{G}$ of hidden order. The *decisional discrete log assumption with small exponents* (DDLSE) holds for GGen if, for any PPT 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 $\mathsf{GGen}(1^\lambda)$, and two group elements $g_1, g_2 \in \mathbb{G}$, where g_1 is sampled uniformly at random from $\mathbb{G}$. For g_2 there are two possibilities of probability $1/2$ each: Either $g_2 = g_1^w$ for some uniform $w \leftarrow [2^\lambda]$ or $g_2 = z^{2^b}$ for a uniformly random group element $z \leftarrow \mathbb{G}$, where $b = 1$ if g_1 is a square and $b = 0$ if not.
2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not $g_2 = g_1^w$ for some $w \in [2^\lambda]$.
3. $\mathcal{A}$ wins if and only if it outputs the correct bit with probability noticeably greater than $1/2$.

3 Definition

In this section we define our notion of blind verifiable delay functions. A blind VDF is basically a VDF with three additional algorithms: Blind, $\mathsf{BlindEval}$ and $\mathsf{Unblind}$. The first three properties below (completeness, sequentiality and soundness) are the same as in the classic VDF definition. The other three can be used to blindly outsource the evaluation of the VDF to an untrusted server.

Definition 8 (Blind Verifiable Delay Function)**.** A *blind verifiable delay function* (blind VDF) with input space $\mathcal{X}$, output space $\mathcal{Y}$ and proof space $\mathcal{PF}$ is a tuple of algorithms (Setup, Eval, Verify, Blind, $\mathsf{BlindEval}$, $\mathsf{Unblind}$) defined as follows.

- $\mathsf{Setup}(1^\lambda) \to \mathbf{pp}$ takes as input a statistical security parameter λ and outputs public parameters $\mathbf{pp}$.
- $\mathsf{Eval}(\mathbf{pp}, x, T) \to (y, \pi)$ takes as input the public parameters $\mathbf{pp}$, an input string x and the time parameter T and outputs the output string y and a proof π. Eval runs in sequential time $T + o(T)$.
- $\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi) \to \mathsf{accept/reject}$.
- $\mathsf{Blind}(\mathbf{pp}, x, T) \to (\alpha, \mathsf{ust})$ takes as input the public parameters $\mathbf{pp}$, an input string x and the time parameter T and outputs a string α and a state ust.
- $\mathsf{BlindEval}(\mathbf{pp}, \alpha) \to \beta$ takes as input the public parameters $\mathbf{pp}$ and the blinded string α and outputs a string β.
- $\mathsf{Unblind}(\mathbf{pp}, \mathsf{ust}, \beta) \to (y, \pi)$ outputs the unblinded output y and proof π.

We want the sum of the runtime of Blind and $\mathsf{Unblind}$ to be $\mathsf{poly}(\lambda)o(T)$. Additionally, we want the blind VDF to satisfy the following properties.

- Completeness: The algorithm $\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi)$ outputs accept whenever $\mathsf{Eval}(\mathbf{pp}, x, T) = (y, \pi)$.
- Sequentiality: No two stage PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_2$ is running in sequential time T' that is sufficiently smaller than T can compute the correct VDF output, i.e.,

$$\Pr\left[\begin{array}{c} \tilde{y} = y \text{ where} \\ (y,\pi) = \mathsf{Eval}(\mathbf{pp}, x, T) \end{array} \middle| \begin{array}{l} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{st} \leftarrow \mathcal{A}_1(\mathbf{pp}) \\ x \leftarrow_{\$} \mathcal{X} \\ \tilde{y} \leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}, x) \end{array}\right] = \mathbf{negl}(\lambda).$$

- Soundness: No PPT adversary $\mathcal{A}$ can come up with a proof for a wrong statement, i.e.,

$$\Pr\left[\mathsf{Verify}(\mathbf{pp}, x, \tilde{y}, T, \tilde{\pi}) = \mathsf{accept} \wedge \tilde{y} \neq y \middle| \begin{array}{r} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, \tilde{y}, \tilde{\pi}) \leftarrow \mathcal{A}(\mathbf{pp}) \\ (y, \pi) \leftarrow \mathsf{Eval}(\mathbf{pp}, x, T) \end{array}\right] = \mathbf{negl}(\lambda).$$

- Blindness: We require that (1) blinded strings reveal no information about the actual input, (2) unblinded outputs look like outputs from Eval and (3) the probability of $\mathsf{Unblind}$ outputting $\perp$ is independent of the input. These properties are formally captured below. Each of the adversaries $\mathcal{A}_i$ considered below can be PPT or unbounded, in our main construction they can all be unbounded and $\mathbf{negl}(\lambda)$ will be exponentially small, i.e., $2^{-\Theta(\lambda)}$.
  1. Blinded strings do not reveal any information about the input, i.e.,

$$\Pr\left[b \leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}, \alpha) \middle| \begin{array}{r} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x_1, x_2, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathbf{pp}) \\ b \leftarrow \{1, 2\} \\ (\alpha, \mathsf{ust}) \leftarrow \mathsf{Blind}(\mathbf{pp}, x_b, T) \end{array}\right] = \frac{1}{2} \pm \mathbf{negl}(\lambda).$$

  2. Unblinded outputs are indistinguishable from correctly evaluated ones. Below, $\mathsf{ub} \in \{(\mathcal{Y} \times \mathcal{PF}), \perp\}$ denotes the unblinded output, which can be invalid. ev is the output of Eval, which we set to $\perp$ whenever $\mathsf{ub} = \perp$ to avoid trivial distinguishers. For any three stage adversary $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ we have

$$\Pr\left[0 \leftarrow \mathcal{A}_3(\mathbf{pp}, \mathsf{st}_2, \mathsf{ub}) \middle| \begin{array}{r} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathbf{pp}) \\ (\alpha, \mathsf{ust}) \leftarrow \mathsf{Blind}(\mathbf{pp}, x, T) \\ (\beta, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}_1, \alpha) \\ \mathsf{ub} \leftarrow \mathsf{Unblind}(\mathbf{pp}, \mathsf{ust}, \beta) \end{array}\right]$$

$$- \Pr\left[0 \leftarrow \mathcal{A}_3(\mathbf{pp}, \mathsf{st}_2, \mathsf{ev}) \middle| \begin{array}{r} \mathbf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathbf{pp}) \\ (\alpha, \mathsf{ust}) \leftarrow \mathsf{Blind}(\mathbf{pp}, x, T) \\ (\beta, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}_1, \alpha) \\ \mathsf{ub} \leftarrow \mathsf{Unblind}(\mathbf{pp}, \mathsf{ust}, \beta) \\ (y, \pi) \leftarrow \mathsf{Eval}(\mathbf{pp}, x, T) \\ \text{if } \mathsf{ub} = \perp, \text{ then } \mathsf{ev} := \perp \\ \text{if } \mathsf{ub} \neq \perp, \text{ then } \mathsf{ev} := (y, \pi) \end{array}\right] = \mathbf{negl}(\lambda).$$

3. The probability of $\mathsf{Unblind}$ outputting $\perp$ is independent of the input. Formally, for any two stage adversary $(\mathcal{A}_1, \mathcal{A}_2,)$ we have

$$|\Pr[\mathsf{ub}_1 = \perp] - \Pr[\mathsf{ub}_2 = \perp]| = \mathbf{negl}(\lambda),$$

where $\mathsf{ub}_1, \mathsf{ub}_2 \in \{(\mathcal{Y} \times \mathcal{PF}), \perp\}$ are obtained as follows: Let $b \in \{1, 2\}$.

$$\begin{aligned} \mathbf{pp} &\leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x_1, x_2, \mathsf{st}) &\leftarrow \mathcal{A}_1(\mathbf{pp}) \\ (\alpha_b, \mathsf{ust}_b) &\leftarrow \mathsf{Blind}(\mathbf{pp}, x_b, T) \\ \beta_b &\leftarrow \mathcal{A}_2(\mathbf{pp}, \mathsf{st}, \alpha_b) \\ \mathsf{ub}_b &\leftarrow \mathsf{Unblind}(\mathbf{pp}, \mathsf{ust}_b, \beta_b). \end{aligned}$$

Before we proceed to our blind VDF construction, we discuss the second and third condition of the blindness property below.

Remark 1 (On the necessity of the third blindness condition)**.** On first glance the third condition of the blindness property might seem redundant since by the first requirement, the adversary is not able to distinguish the blinded inputs for x_1 and x_2. To show that it is necessary we now describe a (contrived) protocol that satisfies the first two requirements but not the third. Assume we have a blind VDF $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}, \mathsf{Blind}, \mathsf{BlindEval}, \mathsf{Unblind})$. We can transform this into a VDF that satisfies all properties of a blind VDF except for the third blindness condition. To this end, we let $\mathsf{BlindEval}$ output an additional bit $b \in \{0, 1\}$ and adapt the $\mathsf{Unblind}$ algorithm as follows: Whenever $b = 0$, $\mathsf{Unblind}$ does the same as before. Whenever $b = 1$, $\mathsf{Unblind}$ performs the unblinding, checks if the first bit of the output is 1 and aborts otherwise. In case $b = 0$, the blind VDF is defined exactly as before and all properties of a blind VDF are satisfied. In case $b = 0$, the blinded inputs are still indistinguishable from random and, in the case where $\mathsf{Unblind}$ does not abort, its output is indistinguishable from the output of Eval. However, if the adversary chooses some input x_1 that starts with bit 1 and an input x_2 that starts with 0, it knows that whenever $\mathsf{Unblind}$ does not abort, the first input was chosen.

We note that the third blindness condition is a property of the unblinding algorithm and in our constructions it will be trivially satisfied.

Remark 2 (Blind VDFs with unique proofs)**.** If the blind VDF satisfies soundness with unique proofs (as defined in Definition 3), the second condition of the blindness property can be easily satisfied. We just need to ensure that the $\mathsf{Unblind}$ algorithm checks that $\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi) = \mathsf{accept}$ before outputting (y, π). If Verify outputs reject, we let $\mathsf{Unblind}$ output $\perp$. We know that, in the case where Verify accepts, the probability that $\mathsf{Unblind}$ outputs a tuple different from Eval is negligible.

4 Basic Construction

Our basic construction can be found in Figure 4. In this section we prove that in the group QR_N^+, our basic construction is a blind VDF with statistical blindness. We will later prove in Section 7.1 that it satisfies computational blindness in other hidden order groups.

Theorem 1. *Let $\mathbb{G} = \mathrm{QR}_N^+$, where $N = pq$ for safe primes p and q and set $L = N/4$. Let a be a random generator of $\mathbb{G}$. The blind VDF presented in Figure 4 satisfies perfect completeness, sequentiality based on the iterated squaring assumption, statistical soundness with unique proofs and statistical blindness.*

Proof. Completeness follows by observing the protocol. Sequentiality and soundness (with unique proofs) follow from sequentiality and soundness (with unique proofs) of the original Pietrzak VDF.

It remains to prove that the construction satisfies statistical blindness. Since the protocol satisfies soundness with unique proofs it suffices to prove that the first requirement of the blindness property holds. To this end we need to show that for any $x_1, x_2 \in \mathbb{G}$ it holds that

$$(\mathbb{G}, N/4, a, a_1, \ldots, a_{\sqrt{T}}, x_1, x_2, T, x^{(1)}_{\mathsf{blind}}) \approx_s (\mathbb{G}, N/4, a, a_1, \ldots, a_{\sqrt{T}}, x_1, x_2, T, x^{(2)}_{\mathsf{blind}}),$$

$\mathsf{Setup}(1^\lambda, T) \to \mathbf{pp} = (\mathbb{G}, L, a, \{a_i := a^{2^{i\sqrt{T}}}\}_{i\in[\sqrt{T}]})$, where $\mathbb{G}$ is a finite abelian group $\mathbb{G}$ of unknown order, L is a bound for the blinding algorithm and $a \in \mathbb{G}$.

$\mathsf{Eval}(\mathbf{pp}, x, T) \to (y, \pi)$, where $y = x^{2^T}$ and π is a PPoE.

$\mathsf{Verify}(\mathbf{pp}, x, y, T, \pi) \to \texttt{accept/reject}$ checks if $\mathsf{PPoE}(\mathbf{pp}, x, y, T)$ is accepting.

$\mathsf{Blind}(\mathbf{pp}, x) \to (x_{\mathsf{blind}}, r)$, where $x_{\mathsf{blind}} = xa^r$ for a uniformly random $r \in [L]$.

$\mathsf{BlindEval}(\mathbf{pp}, x_{\mathsf{blind}}, T) \to (y_{\mathsf{blind}}, \{x_{\mathsf{blind}}^{2^{i\sqrt{T}}}\}_{i\in[\sqrt{T}-1]})$, where $y_{\mathsf{blind}} = x_{\mathsf{blind}}^{2^T}$.

$\mathsf{Unblind}(\mathbf{pp}, x, x_{\mathsf{blind}}, r, y_{\mathsf{blind}}, \{x_{\mathsf{blind}}^{2^{i\sqrt{T}}}\}_{i\in[\sqrt{T}-1]}, T) \to (y, \pi)$, where $y = y_{\mathsf{blind}}/a^r_{\sqrt{T}}$ and π is the PPoE constructed using the elements

$$\left\{x_{\mathsf{blind}}^{2^{i\sqrt{T}}}/a_i^r\right\}_{i\in[\sqrt{T}-1]}$$

as input to the protocol presented in Figure 3.

Figure 4: Our basic blind VDF construction using Pietrzak's PoE denoted by PPoE.

where the notation $\approx_s$ means that the two tuples are statistically indistinguishable. We first note that $a_1, \ldots, a_{\sqrt{T}}$ can be computed from a in time $\mathrm{poly}(\lambda, T)$. This means that the above equation is equivalent to

$$(\mathbb{G}, N/4, a, x_1, x_2, T, x_{\mathsf{blind}}^{(1)}) \approx_s (\mathbb{G}, N/4, a, x_1, x_2, T, x_{\mathsf{blind}}^{(2)}).$$

Let x^* be a uniformly random group element from $\mathbb{G}$. We will show that

$$\begin{aligned}(\mathbb{G}, N/4, a, x_1, x_2, T, x_{\mathsf{blind}}^{(1)}) &\approx_s (\mathbb{G}, N/4, a, x_1, x_2, T, x^*)\\ &\approx_s (\mathbb{G}, N/4, a, x_1, x_2, T, x_{\mathsf{blind}}^{(2)}).\end{aligned} \tag{1}$$

Now x^* is random and, for any $b \in \{1, 2\}$, we have that $x_{\mathsf{blind}}^{(b)} = x_b a^{r_b}$ for a uniformly random $r_b \in [N/4]$. Fix one of the choices for $b \in \{1, 2\}$ and denote by S the set $S := \{x_b a, \ldots, x_b a^{N/4-|\mathbb{G}|}\}$. Note that S is a subset of $\mathbb{G}$ of size $N/4 - |\mathbb{G}|$. Then the statistical distance of $x_{\mathsf{blind}}^{(b)}$ and x^* given x_b is equal to

$$\begin{aligned}&\frac{1}{2}\sum_{g\in\mathbb{G}} |\Pr[x^* = g] - \Pr[x_{\mathsf{blind}}^{(b)} = g]|\\ &\qquad= \frac{1}{2}\left(\sum_{g\notin S}\left|\frac{1}{|\mathbb{G}|} - \frac{4}{N}\right| + \sum_{g\in S}\left|\frac{1}{|\mathbb{G}|} - \frac{8}{N}\right|\right)\\ &\qquad= \frac{1}{2}\left((2|\mathbb{G}| - N/4)\left(\frac{1}{|\mathbb{G}|} - \frac{4}{N}\right) + (N/4 - |\mathbb{G}|)\left(\frac{8}{N} - \frac{1}{|\mathbb{G}|}\right)\right)\\ &\qquad= 3 - \frac{N}{4|\mathbb{G}|} - \frac{8|\mathbb{G}|}{N}\\ &\qquad= 1 - \frac{pq}{(p-1)(q-1)} + \frac{2}{p} + \frac{2}{q} - \frac{2}{pq}\\ &\qquad< \frac{2}{p} + \frac{2}{q} - \frac{2}{pq}.\end{aligned} \tag{2}$$

In the first equation we observe that the group elements contained in the set S correspond to *two* choices of exponents in the set $[N/4]$, while group elements that are not in S only correspond to one exponent in

Input: $(x, T, \mathbb{G}, H)$, where $\mathbb{G}$ is a group of hidden order, $x \in \mathbb{G}$ and $T \in \mathbb{N}$ and H is a hash function.

Public parameters: $\mathbf{pp} = \left(\mathbb{G}, L, u, v := u^{2^T}, \left\{u_i := u^{2^{T/2^i}}\right\}_{i \in [\log T]}\right)$ for $u \in \mathbb{G}$.

Protocol:

1. $\mathcal{C}$ samples $r_1 \leftarrow [L]$ uniformly at random, computes $x^{(1)}_{\mathsf{blind}} = xu^{r_1}$ and sends $x^{(1)}_{\mathsf{blind}}$ and T to $\mathcal{S}$.
2. $\mathcal{S}$ computes $y_{\mathsf{blind}}, \pi^{(1)}_{\mathsf{blind}} := (x^{(1)}_{\mathsf{blind}})^{2^{T/2}}$ and sends it to $\mathcal{C}$.
3. $\mathcal{C}$ computes $y = y_{\mathsf{blind}}/v^{r_1}$ and
$$\pi_1 := \pi^{(1)}_{\mathsf{blind}}/u_1^{r_1}.$$
4. $\mathcal{C}$ sets $x_1 := x$.
5. For $j = 1, \ldots, \log T - 1$, $\mathcal{C}$ and $\mathcal{S}$ do the following.
   (a) $\mathcal{C}$ computes $c_j = H(x, y, \pi_1, \ldots, \pi_j)$, $x_{j+1} = x_j^{c_j}\pi_j$ and $(x^{(j+1)}_{\mathsf{blind}}, r_{j+1}) = \mathsf{Blind}(\mathbf{pp}, x_{j+1})$ and sends $x^{(j+1)}_{\mathsf{blind}}$ and $T/2^j$ to $\mathcal{S}$.
   (b) $\mathcal{S}$ computes $\pi^{(j+1)}_{\mathsf{blind}} = (x^{(j+1)}_{\mathsf{blind}})^{T/2^j}$ and sends it to $\mathcal{C}$.
   (c) $\mathcal{C}$ computes $\pi_{j+1} := \pi^{(j+1)}_{\mathsf{blind}}/u_{j+1}^{r_{j+1}}$.
6. After $\mathcal{C}$ has constructed $\pi = (\pi_1, \pi_2, \ldots, \pi_{\log T})$, it checks if π is an accepting proof. If it is not accepting, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ outputs (y, π).

Figure 5: Our construction with optimal client complexity as an interactive protocol between client $\mathcal{C}$ and server $\mathcal{S}$.

$[N/4]$. In the second equation we count the number of elements that are not in S and the number of elements that are in S. The third equation just follows from basic calculations. In the fourth equation we use the fact that $N = pq$ and $|\mathbb{G}| = |\mathrm{QR}_N^+| = (p-1)(q-1)/4$. The final inequality follows from the fact that $pq(p-1)^{-1}(q-1)^{-1}$ is more than 1. We conclude that the statistical distance between the first two tuples and the last two tuples in Equation 1 is negligible for large enough primes p and q and the claim follows. □

5 Optimizing Client's Complexity

Our client-optimized construction can be found in Figure 5. We now prove that it satisfies statistical blindness in the group QR_N^+. We will later see in Section 7.2 that it satisfies computational blindness in other hidden order groups.

Theorem 2. *Let $\mathbb{G} = \mathrm{QR}_N^+$, where $N = pq$ for safe primes p and q and set $L = N/4$. Let u be a random generator of $\mathbb{G}$. The protocol presented in Figure 5 satisfies blindness.*

Proof. In this construction the client performs $\log T$ blindings with the same group element u. This means that, in the blindness game, we need to consider an adversary that chooses two elements $x_{1,1}$ and $x_{2,1}$ and then set the rest of the instances to $x_{b,i} := x_{b,i}^{2^{T/2^i}}$ for $i \in [\log T]$. Again we use the fact that Pietrzak's VDF satisfies soundness with unique proofs so we only need to prove that the first condition of the blindness

property holds. To this end, we show that for any choice of $x_{1,1}, x_{2,1}$ we have

$$\begin{aligned}&\left(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x_{\mathsf{blind}}^{(1,1)}, \ldots, x_{\mathsf{blind}}^{(1,\log T)}\right)\\ &\approx_s \left(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x_{\mathsf{blind}}^{(2,1)}, \ldots, x_{\mathsf{blind}}^{(2,\log T)}\right),\end{aligned}$$

with the elements defined as in Figure 5. Here $x_{b,1}, \ldots, x_{b,\log T}$ are the unblinded values corresponding to $x_{\mathsf{blind}}^{(b,1)}, \ldots, x_{\mathsf{blind}}^{(b,\log T)}$, i.e., for all $i \in [\log T]$ and $b \in \{1, 2\}$ we have

$$x_{\mathsf{blind}}^{(b,i)} = x_{b,i} u^{r_{b,i}},$$

for some $r_{b,i} \in [N/4]$. Let $x_1^*, \ldots, x_{\log T}^* \in \mathbb{G}$ be uniformly random. We will show that

$$\begin{aligned}&\left(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x_{\mathsf{blind}}^{(1,1)}, \ldots, x_{\mathsf{blind}}^{(1,\log T)}\right)\\ &\approx_s \left(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x_1^*, \ldots, x_{\log T}^*\right)\\ &\approx_s \left(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x_{\mathsf{blind}}^{(2,1)}, \ldots, x_{\mathsf{blind}}^{(2,\log T)}\right). \qquad (3)\end{aligned}$$

This means that we need to compute the statistical distance of the vectors $\mathbf{x}_b := (x_{\mathsf{blind}}^{(b,1)}, \ldots, x_{\mathsf{blind}}^{(b,\log T)})$ and $\mathbf{x}^* := (x_1^*, \ldots, x_{\log T}^*)$ given the other elements in the tuples. We have

$$\begin{aligned}\frac{1}{2} \sum_{\mathbf{g} \in \mathbb{G}^{\log T}} |\Pr[\mathbf{x}^* = \mathbf{g}] - \Pr[\mathbf{x} = \mathbf{g}]| &\leq \sum_{i \in [\log T]} \frac{1}{2} \sum_{g \in \mathbb{G}} |\Pr[x_i^* = g] - \Pr[x_{\mathsf{blind}}^{(b,i)} = g]|\\ &< \sum_{i \in [\log T]} \left(\frac{2}{p} + \frac{2}{q} - \frac{2}{pq}\right)\\ &= \log T \left(\frac{2}{p} + \frac{2}{q} - \frac{2}{pq}\right),\end{aligned}$$

where we have used the union-bound type on the statistical distance of product distributions in the first line and Equation 2 in the second line. We follow that the statistical distance of the first two tuples and the last two tuples of Equation 3 is negligible for large enough primes p and q and the claim follows. □

6 Hybrid Construction

The construction presented in the last section (Figure 5) significantly reduces the complexity of the client since it now only performs $O(\log L \log T)$ group operations. However, the complexity of the server is doubled since it basically needs to compute every proof element from scratch. Further, the construction requires $\log T$ rounds of interaction between the client and the server. To find a compromise between the client's and the server's complexity and the number of rounds of interaction one could also imagine a hybrid approach of our two constructions. Let $s > 1$. In our hybrid construction, presented in Figure 6, we let the server send $T^{1/s}$ elements to the client with which the client computes the first $1/s \cdot \log T$ elements of the proof. Then the client sends the next instance with time parameter $T^{(s-1)/s}$ to the server. Again, the server sends $T^{1/s}$ elements to the client with which the client can compute the next $1/s \cdot \log T$ elements of the proof. The client and the server repeat this for s rounds until the client has computed all $\log T$ proof elements. In this version the client needs to perform $O(T^{1/s})$ group operations. Note that those operations can be parallelized, which is in contrast to our basic construction, where the client has to perform at least $\sqrt{T}$ *sequential* group operations. Since this approach is a hybrid of the two construction above, blindness of this construction can be proven using the same tools. In the group QR_N^+ defined as above, if we set $L = N/4$, the statistical distance between the tuple of the s blinded values that the server sees and s uniformly random values is bounded above by $s(2/p + 2/q - 2/pq)$. In Section 8 we give a detailed comparison of the complexity of our three protocols.

Input: $(x, T, \mathbb{G}, H)$, where $\mathbb{G}$ is a group of hidden order, $x \in \mathbb{G}$ and $T \in \mathbb{N}$ and H is a hash function.

Public parameters: $\mathbf{pp} = \left(\mathbb{G}, L, u, h := g^{2^T}, \left\{g_{i,j} := g^{2^{jT^{(s-i)/s}}}\right\}_{i\in[s-1], j\in[T^{1/s}]}\right)$ for $g \in \mathbb{G}$.

Protocol:

1. $\mathcal{C}$ samples $r_1 \leftarrow [L]$ uniformly at random, computes $x^{(1)}_{\mathsf{blind}} = xg^{r_1}$ and sends $x^{(1)}_{\mathsf{blind}}$ and T to $\mathcal{S}$.
2. $\mathcal{S}$ computes $y_{\mathsf{blind}}, x^{(1,j)}_{\mathsf{blind}} := (x^{(1)}_{\mathsf{blind}})^{2^{jT^{(s-1)/j}}}$ for $j \in [T^{1/s}]$ and sends all elements to $\mathcal{C}$.
3. $\mathcal{C}$ computes $y = y_{\mathsf{blind}}/h^{r_1}$ and constructs the first $1/s \cdot \log T$ proof elements of PPoE using the elements
$$\{x^{(1,j)}_{\mathsf{blind}}/g^{r_1}_{1,j}\}_{j\in[T^{1/s}]}$$
as input to the protocol presented in Figure 3.
4. $\mathcal{C}$ sets $x_{1/s\log T}$ to be the last instance it used in PPoE and $\pi_{1/s\log T} = x^{2^{T^{(s-1)/s}}}_{1/s\log T}$ to be the last proof element it has constructed.
5. For $i = 1, \ldots, s-1$, $\mathcal{C}$ and $\mathcal{S}$ do the following.
   (a) $\mathcal{C}$ computes $c_i = H(x, y, \pi_1, \ldots, \pi_{i/s\log T})$, $x_{i/s\log T+1} = x^{c_i}_{i/s\log T}\pi_{i/s\log T}$ and $(x^{(i+1)}_{\mathsf{blind}}, r_{i+1}) = \mathsf{Blind}(\mathbf{pp}, x_{j/s\log T+1})$ and sends $x^{(i+1)}_{\mathsf{blind}}$ and $T^{(s-1)/s}$ to $\mathcal{S}$.
   (b) $\mathcal{S}$ computes $x^{(i+1,j)}_{\mathsf{blind}} := (x^{(i+1)}_{\mathsf{blind}})^{2^{jT^{(s-i-1)/j}}}$ for $j \in [T^{1/s}]$ and sends all elements to $\mathcal{C}$.
   (c) $\mathcal{C}$ computes the next $1/s \log T$ proof elements of PPoE using the elements
   $$\{x^{(i+1,j)}_{\mathsf{blind}}/g^{r_{i+1}}_{i+1,j}\}_{j\in[T^{1/s}]}$$
   as input to the protocol presented in Figure 3.
6. After $\mathcal{C}$ has constructed $\pi = (\pi_1, \pi_2, \ldots, \pi_{\log T})$, it checks if π is an accepting proof. If it is not accepting, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ outputs (y, π).

Figure 6: Our hybrid construction as an interactive protocol between client $\mathcal{C}$ and server $\mathcal{S}$.

7 Blindness in General Groups of Hidden Order

In this section we analyze blindness of our constructions in general groups of hidden order. So far we have proven our constructions secure in the group QR^+_N. In the constructions and blindness proofs we have relied on the fact that this group is cyclic and that all elements are squares. In general groups of hidden order are not cyclic so no single element can generate the whole group. Further, the groups can consist of both square and non-square elements. This means that we have to choose the public elements a in the basic construction, u in the client-optimized construction and g in the hybrid construction in a careful way.

The first thing to note about the choice of a,u and g is that whenever it is possible to verify that a group element is not a square, we cannot choose a, u or g as a square. Otherwise, given input x and a group element x', an adversary could decide if x' is a blinded element or a uniform element as follows: Compute x'/x and check if the element is a square. If $x' = xa^r$, then x'/x is always a square. If x (chosen by the adversary) is a square and x' is not a square, then x'/x is not a square. Since, in the second case, x' is a uniform element, it is not a square with constant probability. This shows that in groups where one can verify that elements are not squares, we need to choose non-square elements a, u and g in the public parameters. Such groups include RSA groups $\mathbb{Z}^*_N$, where all elements with Jacobi symbol -1 are non-squares. In this

section we will focus on groups where it is possible to verify that elements are not squares since in other groups the distinguishing attack described above does not work.

The second point we need to take care of in this case is that in non-cyclic groups the elements $x_{\mathsf{blind}} = xa^r$ and $x^* \leftarrow \mathbb{G}$ cannot be statistically close. Since a is not a generator of the group, the subgroup generated by a is strictly smaller than the group. This means that x^* can be any group element but x_{blind} (for fixed x and a) cannot. We prove *computational* blindness of our basic construction based on the DDLSE assumption presented in Definition 7. To prove blindness of our client-optimized assumption we need to slightly strengthen this assumption to multiple inputs. We introduce the n-DDLSE assumption in Section 7.2.

7.1 Blindness of the Basic Construction for General Groups

In this section we prove blindness of our basic construction for general groups of hidden order.

Theorem 3. *Let $\mathbb{G}$ be a group of hidden order with a constant fraction of non-square elements. Assume that there exists an efficient algorithm that can verify that an element of $\mathbb{G}$ is not a square. Set $L = 2^\lambda$ and let a be a random non-square of $\mathbb{G}$. The blind VDF presented in Figure 4 satisfies perfect completeness, sequentiality based on the iterated squaring assumption, soundness based on the low order assumption and blindness based on the* DDLSE *assumption.*

Proof. Completeness follows by observing the protocol. Sequentiality and soundness (with unique proofs) follow from sequentiality and soundness (with unique proofs) of the original Pietrzak VDF. It remains to prove that the construction satisfies blindness. Again it suffices to prove the first requirement of the blindness definition. To this end we need to show that

$$(\mathbb{G}, 2^\lambda, a, a_1, \ldots, a_{\sqrt{T}}, x_1, x_2, T, x^{(1)}_{\mathsf{blind}}) \approx_c (\mathbb{G}, 2^\lambda, a, a_1, \ldots, a_{\sqrt{T}}, x_1, x_2, T, x^{(2)}_{\mathsf{blind}}), \tag{4}$$

with the elements defined as in Figure 4. We again note that $a_1, \ldots, a_{\sqrt{T}}$ can be computed from a in time $\mathrm{poly}(\lambda, T)$. This means that Equation 4 is equivalent to

$$(\mathbb{G}, 2^\lambda, a, x_1, x_2, T, x^{(1)}_{\mathsf{blind}}) \approx_c (\mathbb{G}, 2^\lambda, a, x_1, x_2, T, x^{(2)}_{\mathsf{blind}}).$$

Since T is the same in both tuples and not correlated to any of the other elements, we can remove it from the tuples. Let x^* be a uniformly random group element. We will show that

$$(\mathbb{G}, 2^\lambda, a, x_1, x_2, x^{(1)}_{\mathsf{blind}}) \approx_c (\mathbb{G}, 2^\lambda, a, x_1, x_2, x^*) \approx_c (\mathbb{G}, 2^\lambda, a, x_1, x_2, x^{(2)}_{\mathsf{blind}}). \tag{5}$$

We say that the left and right tuples are *blinded* tuples and the tuple in the middle is a *random* tuple. Now x^* is a uniformly random group element from $\mathbb{G}$ and, for $b \in \{1, 2\}$, we have $x^{(b)}_{\mathsf{blind}} = x_b a^{r_b}$ for uniformly random $r_b \in [2^\lambda]$. We now show how to transform any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ that distinguishes between a blinded tuple $(\mathbb{G}, 2^\lambda, a, x_1, x_2, x^{(b)}_{\mathsf{blind}})$ and a random tuple $(\mathbb{G}, 2^\lambda, a, x_1, x_2, x^*)$ with noticeable probability into an adversary $\mathcal{B}$ that breaks the DDLSE assumption with noticeable probability.

Adversary $\mathcal{B}$ gets as input two elements $g_1, g_2 \in \mathbb{G}$, where g_1 is a uniformly random group element and either $g_2 = g_1^w$ for some $w \in [2^\lambda]$ or $g_2 = z^{2^\delta}$ for some uniformly random $z \in \mathbb{G}$. Here $\delta = 1$ if g_1 is a square and $\delta = 0$ otherwise. First $\mathcal{B}$ checks if g_1 is not a square. If g_1 is a square, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ can distinguish between the two options for g_2 as follows.

1. $\mathcal{A}_1$ sends (x_1, x_2, st) to $\mathcal{B}$, where $x \in \mathbb{G}$ and st is some state.
2. $\mathcal{B}$ sets $x' = x_b g_2$.
3. $\mathcal{B}$ sends the tuple $(\mathsf{st}, g_1, x_1, x_2, x')$ to $\mathcal{A}_2$.
4. If $\mathcal{A}_2$ decides that the tuple is a blinded, $\mathcal{B}$ decides that $g_2 = g_1^w$ for some $w \in [2^\lambda]$. If $\mathcal{A}_2$ decides that the tuple is random, $\mathcal{B}$ decides that g_2 is obtained randomly as described above.

Note that if g_2 is random, then x' is also random. Otherwise, x' is the blinding of x_b with public group element g_1. Let δ be the winning probability of $(\mathcal{A}_1, \mathcal{A}_2)$. We observe that $\mathcal{B}$ has the same winning probability as $(\mathcal{A}_1, \mathcal{A}_2)$, whenever g_1 is not a square. Since this holds for a constant fraction of elements in $\mathbb{G}$ by assumption, we follow that $\mathcal{B}$ has winning probability at least δ/c for some constant c. From the DDLSE assumption we follow that δ is negligible. We follow that the first to tuples and the last two tuples of Equation 5 are computationally indistinguishable which concludes the proof. □

7.2 Blindness of the Client-Optimized Construction for General Groups

To prove blindness of our construction we need to introduce a new assumption, which is a minor adaptation of the DDLSE assumption (see Definition 7). Instead of two elements, the adversary now gets as input n pairs of elements (g, h_i), where g is a uniformly random group element and either *all* h_i are uniformly random or for *all* h_i there exists some exponent w_i such that $h_i = g^{w_i}$. There is a short reduction from the n-DDLSE assumption to the DDLSE assumption: If an adversary can distinguish in the DDLSE game, it can also distinguish in the n-DDLSE game by choosing one of the input pairs at random.

Definition 9 (n-decisional discrete log assumption with small exponents)**.** Let $\mathsf{GGen}(1^\lambda)$ be a randomized algorithm that outputs the description of a group $\mathbb{G}$ of hidden order. The *n-decisional discrete log assumption with small exponents* (n-DDLSE) holds for GGen if, for any PPT 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 $\mathsf{GGen}(1^\lambda)$, and n pairs of group elements $(g, h_1), \ldots, (g, h_n) \in \mathbb{G}$, where g is sampled uniformly at random from $\mathbb{G}$. For $h_1, \ldots, h_n$ there are two possibilities of probability 1/2 each: Either we have that for all $i \in [n]$ there exists some $w_i \in [2^\lambda]$ such that $h_i = g^{w_i}$ for some uniform $w_i \leftarrow [2^\lambda]$ or $h_i = z_i^{2^{b_i}}$ for a uniformly random group element $z_i \leftarrow \mathbb{G}$, where $b_i = 1$ if g_i is a square and $b = 0$ if not.
2. $\mathcal{A}$ outputs 0 or 1 indicating whether or not there exist a $w_i \in [2^\lambda]$ such $h_i = g^{w_i}$ for all $i \in [n]$.
3. $\mathcal{A}$ wins if and only if it outputs the correct bit with probability noticeably greater than 1/2.

We are now ready to prove blindness of our client-optimized construction for general groups of hidden order.

Theorem 4. *Let $\mathbb{G}$ be a group of hidden order with a constant fraction of non-square elements. Assume that there exists an efficient algorithm that can verify that an element of $\mathbb{G}$ is not a square. Set $L = 2^\lambda$ and let a be a random non-square of $\mathbb{G}$. The protocol presented in Figure 5 satisfies blindness based on the* $\log T$-DDLSE *assumption.*

Proof. Let $x_1^*, \ldots, x_{\log T}^* \in \mathbb{G}$ be uniformly random group elements. We will show that

$$\begin{aligned}
&(\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots x_{1,\log T}, x_{2,1}, \ldots x_{2,\log T}, T, x_{\mathsf{blind}}^{(1,1)}, \ldots, x_{\mathsf{blind}}^{(1,\log T)}) \\
&\approx_c (\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots x_{1,\log T}, x_{2,1}, \ldots x_{2,\log T}, T, x_1^*, \ldots, x_{\log T}^*) \\
&\approx_c (\mathbb{G}, u, u_1, \ldots, u_{\log T}, x_{1,1}, \ldots x_{1,\log T}, x_{2,1}, \ldots x_{2,\log T}, T, x_{\mathsf{blind}}^{(2,1)}, \ldots, x_{\mathsf{blind}}^{(2,\log T)}),
\end{aligned} \tag{6}$$

with the elements defined as in Figure 5. Again, we say that the left and right tuples are *blinded* tuples and the middle tuple is a *random* tuple. Now $x_1^*, \ldots, x_{\log T}^*$ are uniformly random group elements from $\mathbb{G}$ and for all $i \in [\log T]$, we have $x_{\mathsf{blind}}^{(i)} = x_i u^{r_i}$ for a uniformly random $r_i \in [2^\lambda]$. We now show how to transform any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ that distinguishes between a blinded tuples and a random tuple with noticeable probability into an adversary $\mathcal{B}$ that breaks the $\log T$-DDLSE assumption with noticeable probability.

$\mathcal{B}$ gets as input $\log T$ pairs of elements $(g, h_1), \ldots, (g, h_{\log T}) \in \mathbb{G}$, where g is a uniformly random group element. Further, we have the following for all $i \in [\log T]$: either $h_i = g^{w_i}$ for some $w_i \in [2^\lambda]$ or $h_i = z_i^{2^\delta}$ for some uniformly random $z_i \in \mathbb{G}$. Here $\delta = 1$ if g is a square and $\delta = 0$ otherwise. $\mathcal{B}$ first checks if g is a non-square. If this is not the case, $\mathcal{B}$ aborts. Then $\mathcal{B}$ can distinguish between the two options for the h_i as follows.

	Pietrzak's VDF [24]	Basic construction (Figure 4)	Client-optimized construction (Figure 5)	Hybrid construction (Figure 6)
Blind?	No	Yes	Yes	Yes
Client complexity	0	$2\sqrt{T} + O(\log L \log T)$	$O(\log L \log T)$	$s \cdot T^{1/s} + O(\log L \log T)$
Server complexity	$T + 2\sqrt{T}$	T	$2T$	$T + \sum_{i \in [s-1]} T^{(s-i)/s}$
$\lvert\mathbf{pp}\rvert$	0	$\sqrt{T} + 1$	$\log T + 2$	$s \cdot T^{1/s} + 1$
elements sent	$\log T + 2$	$\sqrt{T} + 2$	$2 \log T + 2$	$s \cdot T^{1/s} + 2$
#rounds	0	0	$\log T$	s

Table 1: An overview of the complexity of outsourcing VDFs. The complexity rows count the number of group operations. $\log L$ is the size of the blinding exponent. $|\mathbf{pp}|$ counts the number of group elements given in the public parameters. #rounds counts the rounds of interaction between the client and the server. The parameter s can be set to any integer in the set $\{2, \ldots, \log T\}$.

1. $\mathcal{A}_1$ sends (x_1, x_2, st) to $\mathcal{B}$, where $x_1, x_2 \in \mathbb{G}$ and st is some state.
2. $\mathcal{B}$ computes $y = x_b^{2^T}$ and the corresponding PPoE π. Denote by $x_{b,1}, \ldots, x_{b,\log T}$ the corresponding instances encountered while computing the proof as it is done in Figure 3.
3. $\mathcal{B}$ sets $x'_i = x_{b,i} h_i$ for all $i \in [\log T]$.
4. $\mathcal{B}$ computes $g_i := g^{2^{T/2^i}}$ for $i \in [\log T]$.
5. $\mathcal{B}$ sends $(\mathsf{st}, g, g_1, \ldots, g_{\log T}, x_{1,1}, \ldots, x_{1,\log T}, x_{2,1}, \ldots, x_{2,\log T}, T, x'_1, \ldots, x'_{\log T})$ to $\mathcal{A}_2$.
6. If $\mathcal{A}_2$ decides that the tuple is a blinded, $\mathcal{B}$ decides that for all $i \in [\log T]$ there exists some $w_i \in [2^\lambda]$ such that $h_i = g^{w_i}$. If $\mathcal{A}_2$ decides that the tuple is random, $\mathcal{B}$ decides that the h_i are obtained randomly as described above.

Note that if the h_i are random, then the x'_i are also random. Otherwise, $x'_1, \ldots, x'_{\log T}$ are blindings of $x_{b,1}, \ldots, x_{b,\log T}$ with public group element g. Let δ be the winning probability of $(\mathcal{A}_1, \mathcal{A}_2)$. We observe that $\mathcal{B}$ has the same winning probability as $(\mathcal{A}_1, \mathcal{A}_2)$, whenever g is not a square. Since a $1/c$ fraction of elements in $\mathbb{G}$ are not squares by assumption, the probability that this is the case is roughly $1/c$. We follow that $\mathcal{B}$ has winning probability at least δ/c. From the $\log T$-DDLSE assumption we follow that δ is negligible. □

7.3 Blindness of the Hybrid Construction for General Groups

As discussed above, blindness of the hybrid construction (see Figure 6) can be proven using the same tools as for the other two constructions. This means that in the same group $\mathbb{G}$ as above, computational blindness of the hybrid construction holds relative to the s-DDLSE assumption.

8 Complexity

In Table 1 we compare the complexity of our three protocols with the complexity of outsourcing Pietrzak's VDF without blinding the input. Note that without any outsourcing the computational complexity of the client would be $T + 2\sqrt{T}$.

The basic construction presented in Figure 4 already reduces the computational work for the client from $O(T)$ to $O(\sqrt{T})$. The server complexity is T computations since only the initial computation x^{2^T} is outsourced in this case. The size of the public parameters and the elements sent between the client and the server are roughly $\sqrt{T}$. The biggest advantage of this blinding protocol is that it requires no interaction.

Our client-optimized construction presented in Figure 5 is the blinding protocol with the lowest complexity of the client. Here, all the client does is blind and unblind $\log T + 1$ elements and the server performs all sequential computations to compute the outputs of the VDF and the proof. Since in this case the server needs to compute all proof elements from scratch, its complexity is $T + T/2 + T/4 + T/8 + ... \approx 2T$. Also the size of the public parameters and the number of elements sent are the lowest in this case. This protocol requires $\log T$ rounds of interaction between the client and the server.

The hybrid construction presented in Section 6 is a hybrid of the two constructions above. Here, the server performs the initial computation of x^{2^T}. Let's assume, for example, that $T = 2^{32}$ and $s = 4$. Then, to compute the proof, the server performs roughly $\sum_{i\in[s-1]} T^{(s-i)/s} = T^{3/4} + T^{2/4} + T^{1/4}$ computations. The client performs roughly $s \cdot T^{1/s} = 2^{10}$ computations. Note however that the computations performed by the server are sequential while the computations performed by the client can be parallelized. This is in contrast to the basic construction where the client needs to perform roughly $\sqrt{T}$ sequential computations. The size of the public parameters and the number of elements sent between client and server are roughly $s \cdot T^{1/s} = 2^{10}$. The number of rounds of interaction is $s = 4$, which is a significant reduction of the number of rounds of the client-optimized construction.

We conclude that there are different trade-offs between our constructions and hence the optimal choice of blinding protocol has to be evaluated for each application individually.

9 Conclusion and Open Problems

In this work we introduced the notion of blind VDFs. A blind VDF enables a client $\mathcal{C}$ that is interested in the output of a VDF on input x to outsource the VDF computation to a server $\mathcal{S}$ such that the server learns nothing about the VDF input x. Further, even after the computation, the server $\mathcal{S}$ should not be able to recognize a VDF output (y, π) as being obtained by using its services. The first contribution of our work is to formalize this notion. Then we presented three different blind VDF constructions with different complexity trade-offs. We proved *statistical* blindness of our constructions in the group of signed quadratic residues, where the modulus N is a product of two safe primes. We also proved *computational* blindness of our constructions in general groups of hidden order. We conclude with two open problems.

1. It would be interesting to see if other VDFs than Pietrzak's VDF can be transformed into blind VDFs. The first VDF that comes to mind is Wesolowski's VDF [29] since it is widely used in practice and it also depends on the iterated squaring function. However, for Wesolowski's VDF it seems to be much more difficult to obtain blindness than in our case since in his protocol the proof is a group element of the form x^q, where $q = (2^T - r)/\ell$ for some random prime number ℓ that is sent by the verifier and some integer r. Since this q is almost as big as 2^T it cannot be computed by the client itself. However, it is not sufficient to blind x and then send q as the new exponent to the server since q contains information about the verifier challenge ℓ and hence needs to be blinded as well in some way. Since ℓ is a prime number and not just a uniform element this seems to be challenging.

2. The second open problem is to find a blind VDF that is at the same time non-interactive and has client complexity of $O(\log T)$. In this work we only present constructions that satisfy one of those properties. Since in practice we cannot count on the server always being online and always prioritizing our client's request, it is desirable to remove interaction between the server and the client.

Acknowledgements. This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85.

References

[1] Bernstein, D., Sorenson, J.: Modular exponentiation via the explicit chinese remainder theorem. Mathematics of Computation 76, 443–454 (2007) 8

[2] Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part IV. Lecture Notes in Computer Science, vol. 12828, pp. 123–152. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021) 5

[3] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 19–23, 2018) 2, 5, 6

[4] Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions using proof of exponentiation. IACR Communications in Cryptology (CiC) 1(1), 7 (2024) 5, 8

[5] Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology – CRYPTO'82. pp. 199–203. Plenum Press, New York, USA, Santa Barbara, CA, USA (1982) 4

[6] Chaum, D.: Blind signature system. In: Chaum, D. (ed.) Advances in Cryptology – CRYPTO'83. p. 153. Plenum Press, New York, USA, Santa Barbara, CA, USA (1983) 3

[7] Chavez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In: AlTawy, R., Hülsing, A. (eds.) Selected Areas in Cryptography. pp. 441–460. Springer International Publishing, Cham (2022) 5

[8] Cini, V., Lai, R.W.F., Malavolta, G.: Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 72–105. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023) 5

[9] De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 248–277. Springer, Cham, Switzerland, Kobe, Japan (Dec 8–12, 2019) 5

[10] Döttling, N., Garg, S., Malavolta, G., Vasudevan, P.N.: Tight verifiable delay functions. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20: 12th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 12238, pp. 65–84. Springer, Cham, Switzerland, Amalfi, Italy (Sep 14–16, 2020) 6

[11] Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part III. Lecture Notes in Computer Science, vol. 12107, pp. 125–154. Springer, Cham, Switzerland, Zagreb, Croatia (May 10–14, 2020) 6

[12] Fischlin, R., Schnorr, C.P.: Stronger security proofs for RSA and Rabin bits. Journal of Cryptology 13(2), 221–244 (Mar 2000) 6

[13] Freitag, C., Pass, R., Sirkin, N.: Parallelizable delegation from LWE. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022: 20th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 13748, pp. 623–652. Springer, Cham, Switzerland, Chicago, IL, USA (Nov 7–10, 2022) 5

[14] Hoffmann, C., Hubácek, P., Ivanova, S.: Practical batch proofs of exponentiation. IACR Commun. Cryptol. 2(3), 9 (2025), https://doi.org/10.62056/abvur-iuc 5

[15] Hoffmann, C., Hubácek, P., Kamath, C., Klein, K., Pietrzak, K.: Practical statistically-sound proofs of exponentiation in any group. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part II. Lecture Notes in Computer Science, vol. 13508, pp. 370–399. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022) 5, 7

[16] Hoffmann, C., Hubácek, P., Kamath, C., Pietrzak, K.: Certifying giant nonprimes. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13940, pp. 530–553. Springer, Cham, Switzerland, Atlanta, GA, USA (May 7–10, 2023) 6

[17] Hoffmann, C., Hubáček, P., Kamath, C., Krňák, T.: (verifiable) delay functions from lucas sequences. In: Rothblum, G.N., Wee, H. (eds.) Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14372, pp. 336–362. Springer (2023), https://doi.org/10.1007/978-3-031-48624-1_13 5

[18] Hoffmann, C., Pietrzak, K.: Watermarkable and zero-knowledge verifiable delay functions from any proof of exponentiation. In: Jager, T., Pan, J. (eds.) PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 15674, pp. 36–66. Springer, Cham, Switzerland, Røros, Norway (May 12–15, 2025) 5, 9

[19] Hofheinz, D., Kiltz, E.: The group of signed quadratic residues and applications. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 637–653. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009) 6

[20] Lai, R.W.F., Malavolta, G.: Lattice-based timed cryptography. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part V. Lecture Notes in Computer Science, vol. 14085, pp. 782–804. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023) 5

[21] Lombardi, A., Vaikuntanathan, V.: Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 632–651. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020) 5

[22] Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: Kleinberg, R.D. (ed.) ITCS 2013: 4th Innovations in Theoretical Computer Science. pp. 373–388. Association for Computing Machinery, Berkeley, CA, USA (Jan 9–12, 2013) 1

[23] Osadnik, M., Kaviani, D., Cini, V., Lai, R., Malavolta, G.: Papercraft : Lattice-based verifiable delay function implemented. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) Proceedings - 46th IEEE Symposium on Security and Privacy, SP 2025. pp. 1603–1621. Proceedings - IEEE Symposium on Security and Privacy, IEEE, United States (2025), publisher Copyright: © 2025 IEEE.; IEEE Symposium on Security and Privacy, SP ; Conference date: 12-05-2025 Through 15-05-2025 5

[24] Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) ITCS 2019: 10th Innovations in Theoretical Computer Science Conference. vol. 124, pp. 60:1–60:15. Leibniz International Proceedings in Informatics (LIPIcs), San Diego, CA, USA (Jan 10–12, 2019) 2, 4, 5, 7, 8, 18

[25] Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., Massachusetts Institute of Technology (1996) 1, 2, 5, 8

[26] Rotem, L.: Simple and efficient batch verification techniques for verifiable delay functions. In: Nissim, K., Waters, B. (eds.) TCC 2021: 19th Theory of Cryptography Conference, Part III. Lecture Notes in Computer Science, vol. 13044, pp. 382–414. Springer, Cham, Switzerland, Raleigh, NC, USA (Nov 8–11, 2021) 5

[27] Shani, B.: A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Paper 2019/205 (2019), https://eprint.iacr.org/2019/205, https://eprint.iacr.org/2019/205 5

[28] Sokolik, Y., Nassar, M., Rottenstreich, O.: Age-aware fairness in blockchain transaction ordering for reducing tail latency. IEEE Trans. Netw. 33(2), 807–822 (2025), https://doi.org/10.1109/TNET.2024.3503758 2

[29] Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 379–407. Springer, Cham, Switzerland, Darmstadt, Germany (May 19–23, 2019) 2, 5, 19

[30] Zhang, H., Yeo, M., Estrada-Galiñanes, V., Ford, B.: Zeroauction: Zero-deposit sealed-bid auction via delayed execution. In: Budurushi, J., Kulyk, O., Allen, S., Diamandis, T., Klages-Mundt, A., Bracciali, A., Goodell, G., Matsuo, S. (eds.) Financial Cryptography and Data Security. FC 2024 International Workshops - Voting, DeFI, WTSC, CoDecFin, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14746, pp. 170–188. Springer (2024), https://doi.org/10.1007/978-3-031-69231-4_12 2