(Verifiable) Delay Functions from Lucas Sequences

Charlotte Hoffmann[1], Pavel Hubáček[2,3], Chethan Kamath[4], and Tomáš Krňák[3]

[1] Institute of Science and Technology Austria, Austria
charlotte.hoffmann@ist.ac.at
[2] Institute of Mathematics, Czech Academy of Sciences, Czech Republic
[3] Charles University, Faculty of Mathematics and Physics, Czech Republic
hubacek@iuuk.mff.cuni.cz,tomas@krnak.cz
[4] Tel Aviv University, Israel
ckamath@protonmail.com

Abstract. Lucas sequences are constant-recursive integer sequences with a long history of applications in cryptography, both in the design of cryptographic schemes and cryptanalysis. In this work, we study the sequential hardness of computing Lucas sequences over an RSA modulus. First, we show that modular Lucas sequences are at least as sequentially hard as the classical delay function given by iterated modular squaring proposed by Rivest, Shamir, and Wagner (MIT Tech. Rep. 1996) in the context of time-lock puzzles. Moreover, there is no obvious reduction in the other direction, which suggests that the assumption of sequential hardness of modular Lucas sequences is strictly weaker than that of iterated modular squaring. In other words, the sequential hardness of modular Lucas sequences might hold even in the case of an algorithmic improvement violating the sequential hardness of iterated modular squaring. Second, we demonstrate the feasibility of constructing practically-efficient *verifiable* delay functions based on the sequential hardness of modular Lucas sequences. Our construction builds on the work of Pietrzak (ITCS 2019) by leveraging the intrinsic connection between the problem of computing modular Lucas sequences and exponentiation in an appropriate extension field.

Keywords: Delay functions · Verifiable delay functions · Lucas sequences.

1 Introduction

A verifiable delay function (VDF) $f : \mathscr{X} \to \mathscr{Y}$ is a function that satisfies two properties. First, it is a delay function, which means it must take a prescribed (wall) time T to compute f, irrespective of the amount of parallelism available. Second, it should be possible for anyone to quickly verify – say, given a short proof π – the value of the function (even without resorting to parallelism), where by quickly we mean that the verification time should be independent of or significantly smaller than T (e.g., logarithmic in T). If we drop either of the two requirements, then the primitive turns out trivial to construct. For instance,

for an appropriately chosen hash function h, the delay function $f(x) = h^T(x)$ defined by T-times iterated hashing of the input is a natural heuristic for an inherently sequential task which, however, seems hard to verify more efficiently than by recomputing. On the other hand, the identity function $f(x) = x$ is trivial to verify but also easily computable. Designing a simple function satisfying the two properties simultaneously proved to be a nontrivial task.

The notion of VDFs was introduced in [31] and later formalised in [9]. In principle, since the task of constructing a VDF reduces to the task of incrementally-verifiable computation [55, 9], constructions of VDFs could leverage succinct non-interactive arguments of knowledge (SNARKs): take any sequentially-hard function f (for instance, iterated hashing) as the delay function and then use the SNARK on top of it as the mechanism for verifying the computation of the delay function. However, as discussed in [9], the resulting construction is not quite practical since we would rely on a general-purpose machinery of SNARKs with significant overhead.

Efficient VDFs via algebraic delay functions. VDFs have recently found interesting applications in design of blockchains [17], randomness beacons [43, 52], proofs of data replication [9], or short-lived zero-knowledge proofs and signatures [3]. Since efficiency is an important factor there, this has resulted in a flurry of constructions of VDFs that are tailored with application and practicality in mind. They rely on more algebraic, structured delay functions that often involve iterating an *atomic operation* so that one can resort to custom proof systems to achieve verifiability. These constructions involve a range of algebraic settings like the RSA or class groups [42, 57, 8, 25, 5], permutation polynomials over finite fields [9], isogenies of elliptic curves [21, 54] and, very recently, lattices [15, 28]. The constructions in [42, 57] are arguably the most practical and the mechanism that underlies their delay function is the same: carry out iterated *squaring* in groups of unknown order, like RSA groups [47] or class groups [12]. What distinguishes these two proposals is the way verification is carried out, i.e., how the underlying "proof of exponentiation" works: while Pietrzak [42] resorts to an LFKN-style recursive proof system [35], Wesolowski [57] uses a clever linear decomposition of the exponent.

Iterated modular squaring and sequentiality. The delay function that underlies the VDFs in [42, 57, 25, 5] is the same, and its security relies on the conjectured *sequential* hardness of iterated squaring in a group of unknown order (suggested in the context of time-lock puzzles by Rivest, Shamir, and Wagner [48]). Given that the practically efficient VDFs all rely on the above single delay function, an immediate open problem is to identify additional sources of sequential hardness that are structured enough to support practically efficient verifiability.

1.1 Our Approach to (Verifiable) Delay Functions

In this work, we study an alternative source of sequential hardness in the algebraic setting and use it to construct efficient verifiable delay functions. The sequentiality

of our delay function relies on an atomic operation that is related to the computation of so-called Lucas sequences [34, 29, 59], explained next.

Lucas sequences. A Lucas sequence is a constant-recursive integer sequence that satisfies the recurrence relation

$$x_i = Px_{i-1} - Qx_{i-2}$$

for integers P and Q.[5] Specifically, the Lucas sequences of integers $(U_k(P,Q))_{k\in\mathbb{N}}$ and $(V_k(P,Q))_{k\in\mathbb{N}}$ of the first and second type (respectively) are defined recursively as

$$U_k(P,Q) = PU_{k-1}(P,Q) - QU_{k-2}(P,Q)$$

with $U_1(P,Q) = 1, U_0(P,Q) = 0$, and

$$V_k(P,Q) = PV_{k-1}(P,Q) - QV_{k-2}(P,Q)$$

with $V_1(P,Q) = P, V_0(P,Q) = 2$.

These sequences can be alternatively defined by the characteristic polynomial $x^2 - Px + Q$. Specifically, given the discriminant $D = P^2 - 4Q$ of the characteristic polynomial, one can alternatively compute the above sequences by performing operations in the extension field

$$\mathbb{Z}[\sqrt{D}] \simeq \mathbb{Z}[x]/(x^2 - D)$$

using the identities

$$U_i = \frac{\omega^i - \overline{\omega}^i}{\omega - \overline{\omega}} \quad \text{and} \quad V_i = \omega^i + \overline{\omega}^i,$$

where $\omega = (P + \sqrt{D})/2$ and its conjugate $\overline{\omega} = (P - \sqrt{D})/2$ are roots of the characteristic polynomial. Since conjugation and exponentiation commute in the extension field (i.e., $\overline{\omega^i} = \overline{\omega}^i$), computing the i-th terms of the two Lucas sequences over integers reduces to computing ω^i in the extension field, and vice versa.

The intrinsic connection between computing the terms in the Lucas sequences and that of exponentiation in the extension has been leveraged to provide alternative instantiations of public-key encryption schemes like RSA and ElGamal in terms of Lucas sequences [30, 7]. However, as we explain later, the corresponding underlying computational hardness assumptions are not necessarily equivalent.

Overview of our delay function. The delay function in [42, 57, 25, 5] is defined as the iterated squaring base x in a (safe) RSA group[6] modulo N:

$$f_N(x,T) := x^{2^T} \bmod N.$$

[5] Note that integer sequences like Fibonacci numbers and Mersenne numbers are special cases of Lucas sequences.

[6] The choice of modulus N is said to be safe if $N = pq$ for safe primes $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also prime.

Our delay function is its analogue in the setting of Lucas sequences:

$$f_N(P, Q, T) := (U_{2^T}(P, Q) \bmod N, V_{2^T}(P, Q) \bmod N).$$

As mentioned above, computing $f_N(P, Q, T)$ can be carried out equivalently in the extension field $\mathbb{Z}_N[\sqrt{D}]$ using the known relationship to roots of the characteristic polynomial of the Lucas sequence. Thus, the delay function can be alternatively defined as

$$f_N(P, Q, T) := \omega^{2^T} \quad \text{in} \quad \mathbb{Z}_N[\sqrt{D}].$$

Note that the atomic operation of our delay function is "doubling" the index of an element of the Lucas sequence modulo N (i.e., $V_i \mapsto V_{2i}$) or, equivalently, squaring in the extension field $\mathbb{Z}_N[\sqrt{D}]$ (as opposed to squaring in $\mathbb{Z}_N$). Using the representation of $\mathbb{Z}_N[\sqrt{D}]$ as $\{a+b\sqrt{D} \mid a, b \in \mathbb{Z}_N\}$, squaring in $\mathbb{Z}_N[\sqrt{D}]$ can be expressed as a combination of squaring, multiplication and addition modulo N, since

$$(a + b\sqrt{D})^2 = (a^2 + b^2 D) + 2ab\sqrt{D}. \tag{1}$$

Since $\mathbb{Z}_N[\sqrt{D}]$ is a group of unknown order (provided the factorization of N is kept secret), iterated squaring remains hard here. In fact, we show in Section 3.2 that iterated squaring in $\mathbb{Z}_N[\sqrt{D}]$ is *at least as hard as* iterated squaring for RSA moduli N. Moreover, we conjecture in Conjecture 1 that it is, in fact, strictly harder (also see discussion below on advantages of our approach).

Verifying modular Lucas sequence. To obtain a VDF, we need to show how to efficiently verify our delay function. To this end, we show how to adapt the interactive proof of exponentiation from [42] to our setting, which then – via the Fiat-Shamir Transform [22] – yields the non-interactive verification algorithm.[7] Thus, our main result is stated informally below.

Theorem 1 (Informally stated, see Theorem 2). *Assuming sequential hardness of modular Lucas sequence, there exists statistically-sound VDF in the random-oracle model.*

However, the modification of Pietrzak's protocol is not trivial and we have to overcome several hurdles that we face in this task, which we elaborate on in Section 1.2. We conclude this section with discussions about our results.

Advantage of our approach. Our main advantage is the reliance on a potentially weaker (sequential) hardness assumption while maintaining efficiency: we show in Section 3.2 that modular Lucas sequences are *at least* as sequentially-hard as the classical delay function given by iterated modular squaring [48]. Despite the linear recursive structure of Lucas sequences, there is no obvious reduction in the other direction, which suggests that the assumption of sequential hardness

[7] Further, using the ideas from [20, 14], it is possible to construct so-called *continuous* VDFs from Lucas sequences.

of modular Lucas sequences is strictly weaker than that of iterated modular squaring (Conjecture 1). In other words, the sequential hardness of modular Lucas sequences might hold even in the case of an algorithmic improvement violating the sequential hardness of iterated modular squaring. Even though both assumptions need the group order to be hidden, we believe that there is need for a nuanced analysis of sequential hardness assumptions in hidden order groups, especially because all current delay functions that provide sufficient structure for applications are based on iterated modular squaring. If the iterated modular squaring assumption is broken, our delay function is currently the only practical alternative in the RSA group.

Delay functions in idealised models. Recent works studied the relationship of group-theoretic (verifiable) delay functions to the hardness of factoring in idealised models such as the algebraic group model and the generic ring model [27, 50]. In the generic ring model, Rotem and Segev [50] showed the equivalence of straight-line delay functions in the RSA setting and factoring. Our construction gives rise to a straight-line delay function and, by their result, its sequentiality is equivalent to factoring *for generic algorithms*. However, their result holds only in the generic ring model and leaves the relationship between the two assumptions unresolved in the standard model.

Compare this with the status of the RSA assumption and factoring. On one hand, we know that *in the generic ring model*, RSA and factoring are equivalent [2]. Yet, it is possible to rule out certain classes of reductions from factoring to RSA in the standard model [11]. Most importantly, despite the equivalence in the generic ring model, there is currently no reduction from factoring to RSA in the standard model and it remains one of the major open problems in number theory related to cryptography since the introduction of the RSA assumption.

In summary, speeding up iterated squaring by a non-generic algorithm could be possible (necessarily exploiting the representations of ring elements modulo N), while such an algorithm may not lead to a speed-up in the computation of modular Lucas sequences despite the result of Rotem and Segev [50].

1.2 Technical Overview

Pietrzak's VDF. Let $N = pq$ be an RSA modulus where p and q are safe primes and let x be a random element from $\mathbb{Z}_N^*$. At its core, Pietrzak's VDF relies on the interactive protocol for the statement

"(N, x, y, T) satisfies $y = x^{2^T} \bmod N$".

The protocol is recursive and, in a round-by-round fashion, reduces the claim to a smaller statement by halving the time parameter. To be precise, in each round the (honest) prover sends the "midpoint" $\mu = x^{2^{T/2}}$ of the current statement to the verifier and they together reduce the statement to

"$(N, x', y', T/2)$ satisfies $y' = (x')^{2^{T/2}} \bmod N$",

where $x' = x^r\mu$ and $y' = \mu^r y$ for a random challenge r. This is continued till $(N, x, y, T = 1)$ is obtained at which point the verifier simply checks whether $y = x^2 \bmod N$ using a single modular squaring.

Since the challenges r are public, the protocol can be compiled into a non-interactive one using the Fiat-Shamir transform [22] and this yields a means to verify the delay function

$$f_N(x, T) = x^{2^T} \bmod N.$$

It is worth pointing out that the choice of safe primes is crucial for proving soundness: in case the group has easy-to-find elements of small order then it becomes easy to break soundness: see [10].

Adapting Pietrzak's protocol to Lucas sequences. For a modulus $N = pq$ and integers $P, Q, T \in \mathbb{N}$, recall that our delay function is defined as

$$f_N(T, P, Q) = (U_{2^T}(P, Q) \bmod N, V_{2^T}(P, Q) \bmod N),$$

or equivalently

$$f_N(T, P, Q) = \omega^{2^T} \quad \text{in} \quad \mathbb{Z}_N[\sqrt{D}],$$

for the discriminant $D = P^2 - 4Q$ of the characteristic polynomial $x^2 - Px + Q$. Towards building a verification algorithm for this delay function, the natural first step is to design an interactive protocol for the statement

"(N, P, Q, y, T) satisfies $y = \omega^{2^T}$ in $\mathbb{Z}_N[\sqrt{D}]$."

It turns out that the interactive protocol from [42] can be adapted for this purpose. However, we encounter two technicalities in this process.

Dealing with elements of small order. The main problem that we face while designing our protocol is avoiding elements of small order. In the case of [42], this was accomplished by moving to the setting of signed quadratic residues [26] in which the sub-groups are all of large order. It is not clear whether a corresponding object exists for our algebraic setting. However, in an earlier draft of Pietrzak's protocol [41], this problem was dealt with in a different manner: the prover sends a square root of μ, from which the original μ can be recovered easily (by squaring it) with a guarantee that the result lies in a group of quadratic residues QR_N. Notice that the prover knows the square root of μ, because it is just a previous term in the sequence he computed.

In our setting, we cannot simply ask for the square root of the midpoint as the subgroup of $\mathbb{Z}_N[\sqrt{D}]$ we effectively work in has a different structure. Nevertheless, we can use a similar approach: for an appropriately chosen small a, we provide an a-th root of ω (instead of ω itself) to the prover in the beginning of the protocol. The prover then computes the whole sequence for $\omega^{\frac{1}{a}}$. In the end, he has the a-th root of every term of the original sequence and he can recover any element of the original sequence by raising to the a-th power.

Sampling strong modulus. The second technicality is related to the first one. In order to ensure that we can use the above trick, we require a modulus where the small subgroups are reasonably small not only in the group $\mathbb{Z}_N$ but also in the extension $\mathbb{Z}_N[\sqrt{D}]$. Thus the traditional sampling algorithms that are used to sample strong primes (e.g., [46]) are not sufficient for our purposes. However, we show in Appendix A that sampling strong primes that suit our criteria can still be carried out efficiently.

Comparing our technique with [8, 25]. The VDFs in [8, 25] are also inspired by [42] and, hence, faced the same problem of low-order elements. In [8], this is dealt with by amplifying the soundness at the cost of parallel repetition and hence larger proofs and extra computation. In [25], the number of repetitions of [8] is reduced significantly by introducing the following technique: The exponent of the initial instance is reduced by some parameter q^C and at the end of an interactive phase, the verifier performs final exponentiation with q^C, thereby weeding out potential false low-order elements in the claim. This technique differs from the approach taken in our work in the following ways: The technique from [25] works in arbitrary groups but it requires the parameter q^C to be large and of a specific form. In particular, the VDF becomes more efficient when q^C is larger than 2^λ. In our protocol, we work in RSA groups whose modulus is the product of primes that satisfy certain conditions depending on a. This enables us to choose a parameter a that is smaller than a statistical security parameter and thereby makes the final exponentiation performed by the verifier much more efficient. Further, a can be any natural number, while q^C must be set as powers of all small prime numbers up a certain bound in [25].

1.3 More Related Work

Timed Primitives. The notion of VDFs was introduced in [31] and later formalised in [9]. VDFs are closely related to the notions of time-lock puzzles [48] and proofs of sequential work [36]. Roughly speaking, a time-lock puzzle is a delay function that additionally allows efficient sampling of the output via a trapdoor. A proof of sequential work, on the other hand, is a delay "multi-function", in the sense that the output is not necessarily unique. Constructions of time-lock puzzles are rare [48, 6, 38], and there are known limitations: e.g, that it cannot exist in the random-oracle model [36]. However, we know how to construct proofs of sequential work in the random-oracle model [36, 16, 1, 19].

Since VDFs have found several applications, e.g., in the design of resource-efficient blockchains [17], randomness beacons [43, 52] and proof of data replication [9], there have been several constructions. Among them, the most notable are the iterated-squaring based construction from [42, 57, 8, 25], the permutation-polynomial based construction from [9], the isogenies-based construction from [21, 54, 13] and the construction from lattice problems [28, 15]. The constructions in [42, 57] are quite practical (see the survey [10]) and the VDF deployed in the cryptocurrency Chia is basically their construction adapted to the algebraic setting of class groups [17]. This is arguably the closest work to ours. On the

other hand, the constructions from [21, 54], which work in the algebraic setting of isogenies of elliptic curves where no analogue of square and multiply is known, simply rely on "exponentiation". Although, these constructions provide a certain form of quantum resistance, they are presently far from efficient. Recently, Freitag et al. [23] constructed VDFs from any sequential hard function and polynomial hardness of learning with errors, the first from standard assumptions. Very recently, the works of Cini, Lai and Malavolta [28, 15] constructed the first VDF from lattice-based assumptions and conjectured it to be post-quantum secure.

Several variants of VDFs have also been proposed. A VDF is said to be unique if the proof that is used for verification is unique [42]. Recently, Choudhuri et al. [5] constructed unique VDFs from the sequential hardness of iterated squaring in *any* RSA group and polynomial hardness of LWE. A VDF is tight [18] if the gap between simply computing the function and computing it *with* a proof is small. Yet another extension is a continuous VDF [20]. The feasibility of time-lock puzzles and proofs of sequential works were recently extended to VDFs. It was shown [51] that the latter requirement, i.e., working in a group of unknown order, is inherent in a black-box sense. It was shown in [18, 37] that there are barriers to constructing tight VDFs in the random-oracle model.

VDFs also have surprising connection to complexity theory [14, 20, 33].

Work related to Lucas sequences. Lucas sequences have long been studied in the context of number theory: see for example [45] or [44] for a survey of its applications to number theory. Its earliest application to cryptography can be traced to the $(p + 1)$ factoring algorithm [58]. Constructive applications were found later thanks to the parallels with exponentiation. Several encryption and signature schemes were proposed, most notably the LUC family of encryption and signatures [30, 39]. It was later shown that some of these schemes can be broken or that the advantages it claimed were not present [7]. Other applications can be found in [32].

2 Preliminaries

2.1 Interactive Proof Systems

Interactive protocols. An interactive protocol consists of a pair (P, V) of interactive Turing machines that are run on a common input x. The first machine P is the prover and is computationally unbounded. The second machine V is the verifier and is probabilistic polynomial-time.

In an ℓ-round (i.e., $(2\ell - 1)$-message) interactive protocol, in each round $i \in [1, \ell]$, first P sends a message $\alpha_i \in \Sigma^a$ to V and then V sends a message $\beta_i \in \Sigma^b$ to P, where Σ is a finite alphabet. At the end of the interaction, V runs a (deterministic) Turing machine on input $\{x, (\beta_1, \ldots, \beta_\ell), (\alpha_1, \ldots, \alpha_\ell)\}$. The interactive protocol is *public-coin* if β_i is a uniformly distributed random string in Σ^b.

Interactive proof systems. The notion of an interactive proof for a language L is due to Goldwasser, Micali and Rackoff [24].

Definition 1. *For a function $\epsilon : \mathbb{N} \to [0,1]$, an interactive protocol (P, V) is an* ϵ-statistically-sound *interactive proof system for L if:*

- ***Completeness:*** *For every $x \in L$, if V interacts with P on common input x, then V accepts with probability 1.*
- ***Soundness:*** *For every $x \notin L$ and every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$, the verifier V accepts when interacting with $\widetilde{\mathsf{P}}$ with probability less than $\epsilon(|x|)$, where ϵ is called the* soundness error*.*

2.2 Verifiable Delay Functions

We adapt the definition of verifiable delay functions from [9] but we decouple the verifiability and sequentiality properties for clarity of exposition of our results. First, we present the definition of a delay function.

Definition 2. *A* delay function $\mathtt{DF} = (\mathtt{DF.Setup}, \mathtt{DF.Gen}, \mathtt{DF.Eval})$ *consists of a triple of algorithms with the following syntax:*

$\mathtt{pp} \leftarrow \mathtt{DF.Setup}(1^n)$*:*
On input a security parameter 1^n, the algorithm $\mathtt{DF.Setup}$ *outputs public parameters* $\mathtt{pp}$*.*

$x \leftarrow \mathtt{DF.Gen}(\mathtt{pp}, T)$*:*
On input public parameters $\mathtt{pp}$ *and a time parameter $T \in \mathbb{N}$, the algorithm* $\mathtt{DF.Gen}$ *outputs a challenge x.*

$y \leftarrow \mathtt{DF.Eval}(\mathtt{pp}, (x, T))$*:*
On input a challenge pair (x, T), the (deterministic) algorithm $\mathtt{DF.Eval}$ *outputs the value y of the delay function in time T.*

The security property required of a delay function is sequential hardness as defined below.

Definition 3 (Sequentiality). *We say that a delay function* $\mathtt{DF}$ *satisfies the* sequentiality *property, if there exists an $\epsilon \in (0,1)$ such that for all $T(\lambda) \in \mathrm{poly}(\lambda)$ and for every adversary $A = (A_0, A_1)$, where A_1 uses $\mathrm{poly}(\lambda)$ processors and runs in time $O(T^{\epsilon}(\lambda))$, there exists a negligible function μ such that*

$$\Pr\left[\begin{array}{r} A_1(\mathtt{pp}, \mathtt{state}, (x, T(\lambda))) = y \\ \textit{where} \\ \mathtt{pp} \leftarrow \mathtt{DF.Setup}(1^n) \\ \mathtt{state} \leftarrow A_0(\mathtt{pp}) \\ x \leftarrow \mathtt{DF.Gen}(\mathtt{pp}, T(\lambda)) \\ y \leftarrow \mathtt{DF.Eval}(\mathtt{pp}, (x, T(\lambda))) \end{array}\right] \leq \mu(\lambda).$$

A few remarks about our definition of sequentiality are in order:

1. We require computing $\texttt{DF.Eval}(\texttt{pp}, (x, T))$ to be hard in less than T sequential steps even using any polynomially-bounded amount of parallelism and precomputation. Note that it is necessary to bound the amount of parallelism, as an adversary could otherwise break the underlying hardness assumption (e.g. hardness of factorization). Analogously, T should be polynomial in λ as, otherwise, breaking the underlying hardness assumptions becomes easier than computing $\texttt{DF}(x, T)$ itself for large values of T.
2. Another issue is what bound on the number of sequential steps of the adversary should one impose. For example, the delay function based on T repeated modular squarings can be computed in sequential time $O(T/\text{loglog } T)$ using polynomial parallelism [4]. Thus, one cannot simply bound the sequential time of the adversary by $o(T)$. Similarly to [38], we adapt the $O(T^\epsilon)$ bound for $\epsilon \in (0, 1)$ which, in particular, is asymptotically smaller than $O(T/\text{loglog } T)$.
3. Without loss of generality, we assume that the size of $\texttt{pp}$ is at least linear in n and the adversary A does not have to get the unary representation of the security parameter 1^n as its input.

The definition of *verifiable* delay function extends a delay function with the possibility to compute publicly-verifiable proofs of correctness of the output value.

Definition 4. *A delay function* $\texttt{VDF} = (\texttt{VDF.Setup}, \texttt{VDF.Gen}, \texttt{VDF.Eval})$ *is a* verifiable delay function *if it is equipped with two additional algorithms* $\texttt{VDF.Prove}$ *and* $\texttt{VDF.Verify}$ *with the following syntax:*

$(y, \pi) \leftarrow \texttt{VDF.Prove}(\texttt{pp}, (x, T))$*:*
On input public parameters and a challenge pair (x, T)*, the* $\texttt{VDF.Prove}$ *algorithm outputs* (y, π)*, where* π *is a proof that the output* y *is the output of* $\texttt{VDF.Eval}(\texttt{pp}, (x, T))$.

$\{\texttt{accept/reject}\} \leftarrow \texttt{VDF.Verify}(\texttt{pp}, (x, T), (y, \pi))$*:*
On input public parameters, a challenge pair (x, T)*, and an output/proof pair* (y, π)*, the (deterministic) algorithm* $\texttt{VDF.Verify}$ *outputs either* $\texttt{accept}$ *or* $\texttt{reject}$*.*

In addition to sequentiality (inherited from the underlying delay function), the $\texttt{VDF.Prove}$ and $\texttt{VDF.Verify}$ algorithms must together satisfy correctness and (statistical) soundness as defined below.

Definition 5 (Correctness). *A verifiable delay function* $\texttt{VDF}$ *is* correct *if for all* $T \in \mathbb{N}$

$$\Pr\left[\begin{array}{r} \texttt{VDF.Verify}(\texttt{pp}, (x, T), (y, \pi)) = \texttt{accept} \\ \textit{where} \\ \texttt{pp} \leftarrow \texttt{VDF.Setup}(1^n) \\ x \leftarrow \texttt{VDF.Gen}(\texttt{pp}, T) \\ (y, \pi) \leftarrow \texttt{VDF.Prove}(\texttt{pp}, (x, T)) \end{array}\right] = 1.$$

Definition 6 (Statistical soundness). *A verifiable delay function* $\texttt{VDF}$ *is* statistically sound *if for every (computationally unbounded) malicious prover* P^* *there exists*

a negligible function $\mu(\lambda)$ such that

$$\Pr\left[\begin{array}{r} \texttt{VDF.Verify}(\texttt{pp},(x,T),(\tilde{y},\tilde{\pi})) = \texttt{accept} \\ \textit{and} \quad y \neq \tilde{y} \\ \textit{where} \\ \texttt{pp} \leftarrow \texttt{VDF.Setup}(1^n) \\ x \leftarrow \texttt{VDF.Gen}(\texttt{pp},T) \\ y \leftarrow \texttt{VDF.Eval}(\texttt{pp},(x,T)) \\ (\tilde{y},\tilde{\pi}) \leftarrow P^*(\texttt{pp},(x,T)) \end{array}\right] \leq \mu(\lambda).$$

3 Delay Functions from Lucas Sequences

In this section, we propose a delay function based on Lucas sequences and prove its sequentiality assuming that iterated squaring in a group of unknown order is sequential (Section 3.1). Further, we conjecture (Section 3.2) that our delay function candidate is even more robust than its predecessor proposed by Rivest, Shamir, and Wagner [48]. Finally, we turn our delay function candidate into a *verifiable* delay function (Section 4).

3.1 The Atomic Operation

Our delay function is based on subsequences of Lucas sequences, whose indexes are powers of two.

Definition 7. *For $P, Q \in \mathbb{Z}$ and $t \in \mathbb{N} \cup \{0\}$, we define subsequences (u_t) and (v_t) of Lucas sequences by*

$$u_t := U_{2^t}(P,Q) \quad \textit{and} \quad v_t := V_{2^t}(P,Q). \tag{2}$$

Although the value of (u_t, v_t) depends on parameters (P, Q), we omit (P, Q) from the notation because these parameters will be always obvious from the context.

The underlying atomic operation our delay function is

$$f_N(T,P,Q) = (u_T \bmod N, v_T \bmod N).$$

There are several ways to compute (u_t, v_t) in T sequential steps, and we describe two of them below.

An approach based on squaring in a suitable extension ring. To compute the value $f_N(T, P, Q)$, we can use the extension ring $\mathbb{Z}_N[\sqrt{D}]$, where $D := P^2 - 4Q$ is the discriminant of the characteristic polynomial $f(z) = z^2 - Pz + Q$ of the Lucas sequence. The characteristic polynomial $f(z)$ has a root $\omega := (P + \sqrt{D})/2 \in \mathbb{Z}_N[\sqrt{D}]$, and it is known that, for all $i \in \mathbb{N}$, it holds that

$$\omega^i = \frac{V_i + U_i\sqrt{D}}{2} \quad \left(\text{i.e.,} \quad \omega^{2^t} = \frac{v_t + u_t\sqrt{D}}{2}\right).$$

Thus, by iterated squaring of ω, we can compute terms of our target subsequences. To get a better understanding of squaring in the extension ring, consider the representation of the root $\omega = a + b\sqrt{D}$ for some $a, b \in \mathbb{Z}_N$. Then,

$$(a + b\sqrt{D})^2 = (a^2 + b^2 D) + 2ab\sqrt{D}.$$

Then, the atomic operation of our delay function can be interpreted as $g\colon \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N \times \mathbb{Z}_N$, defined for all $a, b \in \mathbb{Z}_N$ as

$$g : (a, b) \mapsto (a^2 + b^2 D, 2ab) \quad g^t(\left(\frac{P}{2}, \frac{1}{2}\right)) = \left(\frac{v_t}{2}, \frac{u_t}{2}\right). \tag{3}$$

An approach based on known identities. Many useful identities for members of modular Lucas sequences are known, such as

$$U_{j+i} = U_j V_i - Q^i U_{j-i}, \quad \text{and} \quad V_{j+i} = V_j V_i - Q^i V_{j-i}. \tag{4}$$

Setting $j = i$ we get

$$U_{2i} = U_i V_i, \quad \text{and} \quad V_{2i} = V_i^2 - 2Q^i. \tag{5}$$

The above identities are not hard to derive (see, e.g., Lemma 12.5 in [40]). Indexes are doubled on each of application of the identities in Equation (5), and, thus, for $t \in \mathbb{N} \cup \{0\}$, we define an auxiliary sequence (q_t) by $q_t := Q^{2^t}$. Using the identities in Equation (5), we get recursive equations

$$u_{t+1} = u_t v_t, \quad v_{t+1} = v_t^2 - 2q_t \quad \text{and} \quad q_{t+1} = q_t^2. \tag{6}$$

Then, the atomic operation of our delay function can be interpreted as $g\colon \mathbb{Z}_N \times \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N \times \mathbb{Z}_N \times \mathbb{Z}_N$, defined for all $u, v, q \in \mathbb{Z}_N$ as

$$g : (u, v, q) \mapsto (uv, v^2 - 2q, q^2), \quad g^t((1, P, Q)) = (u_t, v_t, Q^{2^t}). \tag{7}$$

After a closer inspection, the reader may have an intuition that an auxiliary sequence q_t, which introduces a third state variable, is redundant. This intuition is indeed right. In fact, there is another easily derivable identity

$$q_t = \frac{v_t^2 - u_t^2 D}{4}, \tag{8}$$

which can be found, e.g., as Lemma 12.2 in [40]. On the other hand, Equation (8) is quite interesting because it allows us to compute large powers of an element $Q \in \mathbb{Z}_N$ using two Lucas sequences. We use this fact in the security reduction in Section 3.2. Our construction of a delay function, denoted `LCS`, is given in Figure 1.

$\texttt{LCS.Setup}(1^n)$: Samples two n-bit primes p and q and outputs $N := p \cdot q$.
$\texttt{LCS.Gen}(N, T)$: Samples P and Q independently from the uniform distribution on $\mathbb{Z}_N$, sets $D := P^2 - 4Q$ and outputs (P, D, T).
$\texttt{LCS.Eval}(N, (P, D, T))$: Sets

$$g \colon \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N \times \mathbb{Z}_N$$
$$(a, b) \mapsto (a^2 + b^2 D, 2ab),$$

computes

$$\left(\frac{v_T}{2}, \frac{u_T}{2}\right) := g^T \left(\frac{P}{2}, \frac{1}{2}\right)$$

and outputs (u_T, v_T).

Fig. 1. Our delay function candidate LCS based on a modular Lucas sequence.

On the discriminant D. Notice that whenever D is a quadratic residue modulo N, the value $\sqrt{D}$ is an element of $\mathbb{Z}_N$ and hence $\mathbb{Z}_N[\sqrt{D}] = \mathbb{Z}_N$. By definition, LCS.Gen generates a parameter D that is a quadratic residue with probability $1/4$, so it might seem that in one fourth of the cases there is another approach to compute (u_t, v_t): find the element $\sqrt{D}$ and then perform n sequential squarings in the group $\mathbb{Z}_N$. However, it is well known that finding square roots of uniform elements in $\mathbb{Z}_N$ is equivalent to factoring the modulus N, so this approach is not feasible. We can therefore omit any restrictions on the discriminant D in the definition of our delay function LCS.

3.2 Reduction from RSW Delay Function

In order to prove the sequentiality property (Definition 3) of our candidate LCS, we rely on the standard conjecture of the sequentiality of the RSW time-lock puzzles, implicitly stated in [48] as the underlying hardness assumption.

Definition 8 (RSW delay function). *The RSW delay function is defined as follows:*

$\texttt{RSW.Setup}(1^n)$*: Samples two n-bit primes p and q and outputs $N := p \cdot q$.*
$\texttt{RSW.Gen}(N, T)$*: Outputs an x sampled from the uniform distribution on $\mathbb{Z}_N^*$.*
$\texttt{RSW.Eval}(N, (x, T))$*: Outputs $y := x^{2^T} \mod N$.*

Theorem 2. *If the RSW delay function has the sequentiality property, then the LCS delay function has the sequentiality property.*

Proof. Suppose there exists an adversary (A_0, A_1) who contradicts the sequentiality of LCS, where A_0 is a precomputation algorithm and A_1 is an online algorithm. We construct an adversary (B_0, B_1) who contradicts the sequentiality of RSW as follows:

- The algorithm B_0 is defined identically to the algorithm A_0.

– On input $(N, \texttt{state}, (x \in \mathbb{Z}_N^*, T))$, B_1 picks a P from the uniform distribution on $\mathbb{Z}_N$, sets

$$Q := x, \quad D := P^2 - 4Q$$

and it runs $A_1(N, \texttt{state}, (P, D, T))$ to compute (u_T, v_T). The algorithm B_1 computes $y = x^{2^T} = Q^{2^T} = q_T$ using the identity in Equation (8).

Note that the input distribution for the algorithm A_1 produced by B_1 differs from the one produced by `LCS.Gen`, because the `LCS` generator samples Q from the uniform distribution on $\mathbb{Z}_N$ (instead of $\mathbb{Z}_N^*$). However, this is not a problem since the size of $\mathbb{Z}_N \setminus \mathbb{Z}_N^*$ is negligible compared to the size of $\mathbb{Z}_N$, so the statistical distance between the distribution of D produced by B_1 and the distribution of D sampled by `LCS.Gen` is negligible in the security parameter. Thus, except for a negligible multiplicative loss, the adversary (B_0, B_1) attains the same success probability of breaking the sequentiality of `RSW` as the probability of (A_0, A_1) breaking the sequentiality of `LCS` – a contradiction to the assumption of the theorem. □

We believe that the converse implication to Theorem 2 is not true, i.e., that breaking the sequentiality of `RSW` does not necessarily imply breaking the sequentiality of `LCS`. Below, we state it as a conjecture.

Conjecture 1. Sequentiality of `LCS` cannot be reduced to sequentiality of `RSW`.

One reason why the above conjecture might be true is that, while the `RSW` delay function is based solely only on multiplication in the group $\mathbb{Z}_N^*(\cdot)$, our `LCS` delay function uses the full arithmetic (addition and multiplication) of the commutative ring $\mathbb{Z}_N$.

One way to support the conjecture would be to construct an algorithm that speeds up iterated squaring but is not immediately applicable to Lucas sequences. By [49] we know that this cannot be achieved by a generic algorithm. A non-generic algorithm that solves iterated squaring in time $O(T/\text{loglog}(T))$ is presented in [4]. The main tool of their construction is the Explicit Chinese Remainder Theorem modulo N. However, a similiar theorem exists also for univariate polynomial rings, which suggests that a similar speed-up can be obtained for our delay function by adapting the techniques in [4] to our setting.

4 VDF from Lucas sequences

In Section 3.1 we saw different ways of computing the atomic operation of the delay function. Computing (u_t, v_t) in the extension field seems to be the more natural and time and space effective approach. Furthermore, writing the atomic operation $g(a, b) = (a^2 + b^2 D, 2ab)$ as $\omega \mapsto \omega^2$ is very clear, and, thus, we follow this approach throughout the rest of the paper.

4.1 Structure of $\mathbb{Z}_N[x]/(x^2 - Px + Q)$

To construct a VDF based on Lucas sequences, we use an algebraic extension

$$\mathbb{Z}_N[x]/(x^2 - Px + Q), \tag{9}$$

where N is an RSA modulus and $P, Q \in \mathbb{Z}_N$. In this section, we describe the structure of the algebraic extension given in Expression (9). Based on our understanding of the structure of the above algebraic extension, we can conclude that using modulus N composed of safe primes (i.e., for all prime factors p of N, $p-1$ has a large prime divisor) is necessary but not sufficient condition for security of our construction. We specify some sufficient conditions on factors of N in the subsequent Section 4.2.

First, we introduce some simplifying notation for quotient rings.

Definition 9. *For $m \in \mathbb{N}$ and $f(x) \in \mathbb{Z}_m[x]$, we denote by $\mathbb{Z}_{m,f}$ the quotient ring $\mathbb{Z}[x]/(m, f(x))$, where $(m, f(x))$ denotes the ideal of the ring $\mathbb{Z}[x]$ generated by m and $f(x)$.*

Observation 1, below, allows us to restrict our analysis only to the structure of $\mathbb{Z}_{p,f}$ for prime $p \in \mathbb{P}$.

Observation 1 *Let $p, q \in \mathbb{P}$ be distinct primes, $N := p \cdot q$ and $f(x) \in \mathbb{Z}_N[x]$. Then*

$$\mathbb{Z}_{N,f} \simeq \mathbb{Z}_{p,f} \times \mathbb{Z}_{q,f} \ .$$

Proof. Using the Chinese reminder theorem, we get

$$\mathbb{Z}_{N,f} \simeq \frac{\mathbb{Z}[x]/(f(x))}{(N)} \simeq \frac{\mathbb{Z}[x]/(f(x))}{(p)} \times \frac{\mathbb{Z}[x]/(f(x))}{(q)} \simeq \mathbb{Z}_{p,f} \times \mathbb{Z}_{q,f}$$

as claimed. □

The following lemma characterizes the structure of $\mathbb{Z}_{p,f}$ with respect to the discriminant of f. We use $\left(\frac{a}{p}\right)$ to denote the standard Legendre symbol.

Lemma 1. *Let $p \in \mathbb{P} \setminus \{2\}$ and $f(x) \in \mathbb{Z}_p[x]$ be a polynomial of degree 2 with the discriminant D. Then*

$$\mathbb{Z}^*_{p,f}(\cdot) \simeq \begin{cases} \mathbb{Z}_{p^2-1}(+) & \left(\frac{D}{p}\right) = -1 \\ \mathbb{Z}_{p-1}(+) \times \mathbb{Z}_p(+) & \left(\frac{D}{p}\right) = 0 \\ \mathbb{Z}_{p-1}(+) \times \mathbb{Z}_{p-1}(+) & \left(\frac{D}{p}\right) = 1 \end{cases} .$$

Proof. We consider each case separately:

- If $\left(\frac{D}{p}\right) = -1$, then $f(x)$ is irreducible over $\mathbb{Z}_p$ and $\mathbb{Z}_{p,f}$ is a field with p^2 elements. Since $\mathbb{Z}_{p,f}$ is a finite field, $\mathbb{Z}^*_{p,f}$ is cyclic and contains $p^2 - 1$ elements.

– If $\left(\frac{D}{p}\right) = 0$, then $D = 0$ and f has some double root α and it can be written as $\beta(x-\alpha)^2$ for some $\beta \in \mathbb{Z}_p^*$. Since the ring $\mathbb{Z}_{p,\beta(x-\alpha)^2}$ is isomorphic to the ring $\mathbb{Z}_{p,x^2}$ (consider the isomorphism $g(x) \mapsto g(x+\alpha)$), we can restrict ourselves to describing the structure of $\mathbb{Z}_{p,x^2}$.

We will prove that the function ψ,

$$\begin{aligned}\psi &: \mathbb{Z}_p^*(\cdot) \times \mathbb{Z}_p(+) \to \mathbb{Z}_{p,x^2}^*(\cdot),\\ \psi &: (a,b) \mapsto a \cdot (1+x)^b,\end{aligned}$$

is an isomorphism. First, the polynomial $a + cx \in \mathbb{Z}_{p,x^2}$ is invertible if and only if $a \neq 0$ (inverse is $a^{-1} - a^{-2}cx$). For the choice $b = a^{-1}c$, we have

$$\psi(a,b) = a(1+x)^b \equiv a(1+bx) \equiv a(1+a^{-1}cx) \equiv a + cx \mod (p, x^2).$$

Thus ψ is onto. Second, ψ is, in fact, a bijection, because

$$|\mathbb{Z}_{p,x^2}^*(\cdot)| = p^2 - p = (p-1)\cdot p = |\mathbb{Z}_p^*(\cdot) \times \mathbb{Z}_p(+)|. \tag{10}$$

Finally, ψ is a homomorphism, because

$$\psi(a_1,b_1)\cdot\psi(a_2,b_2) = a_1a_2(1+x)^{b_1+b_2} = \psi(a_1a_2, b_1+b_2).$$

If $\left(\frac{D}{p}\right) = 1$, then $f(x)$ has two roots $\beta_1, \beta_2 \in \mathbb{Z}_p$. We have an isomorphism

$$\begin{aligned}\psi &: \mathbb{Z}_p[x]/(f(x)) \to \mathbb{Z}_p \times \mathbb{Z}_p\\ \psi &: g(x) + (f(x)) \mapsto (g(\beta_1), g(\beta_2))\end{aligned}$$

and $(\mathbb{Z}_p \times \mathbb{Z}_p)^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_p^* \simeq \mathbb{Z}_{p-1}(+) \times \mathbb{Z}_{p-1}(+)$. □

4.2 Strong Groups and Strong Primes

To achieve the verifiability property of our construction, we need $\mathbb{Z}_{p,f}^*$ to contain a strong subgroup (defined next) of order asymptotically linear in p. We remark that our definition of strong primes is stronger than the one by Rivest and Silverman [46].

Definition 10 (Strong groups). *For $\lambda \in \mathbb{N}$, we say that a non-trivial group $\mathbb{G}$ is λ-strong, if the order of each non-trivial subgroup of $\mathbb{H}$ is greater than 2^λ.*

Observation 2 *If $\mathbb{G}_1$ and $\mathbb{G}_2$ are λ-strong groups, then $\mathbb{G}_1 \times \mathbb{G}_2$ is a λ-strong group.*

It can be seen from Lemma 1 that $\mathbb{Z}_{p,f}^*$ always contains groups of small order (e.g. $Z_2(+)$). To avoid these, we descend into the subgroup of a-th powers of elements of $\mathbb{Z}_{p,f}^*$. Below, we introduce the corresponding notation.

Definition 11. *For an Abelian group $\mathbb{G}$ and $a \in \mathbb{N}$, we define the subgroup $\mathbb{G}^{(a)} := \{x^a \mid x \in \mathbb{G}\}$ of $\mathbb{G}$ in the multiplicative notation and $a\mathbb{G} := \{ax \mid x \in \mathbb{G}\}$ in the additive notation.*

Further, we show in Lemma 2 below that (λ, a)-strong primality (defined next) is a sufficient condition for $(\mathbb{Z}^*_{p,f})^{(a)}$ to be a λ-strong group.

Definition 12 (Strong primes). *Let $p \in \mathbb{P}$ and $\lambda, a \in \mathbb{N}$. We say that p is a (λ, a)-strong prime, if $\lambda > a$ and there exists $W \in \mathbb{N}$, $W > 1$, such that $p^2 - 1 = aW$ and every prime factor of W is greater than 2^λ.*

Since a is a public parameter in our setup, super-polynomial a could reveal partial information about the factorization of N. However, we could allow a to be polynomial in λ while maintaining hardness of factoring N.[8] For the sake of simplicity of Definition 12, we rather use stronger condition $a < \lambda$. The following simple observation will be useful for proving Lemma 2.

Observation 3 *For $\forall m, n \in \mathbb{N} : n\mathbb{Z}_m \simeq \mathbb{Z}_{m/gcd(m,n)}$.*

Lemma 2. *Let p be a (λ, a)-strong prime and $f(x) \in \mathbb{Z}_p[x]$ be a quadratic polynomial. Then, $(\mathbb{Z}^*_{p,f})^{(a)}$ is a λ-strong group.*

Proof. From definition of the strong primes, there exists $W \in \mathbb{N}$, whose factors are bigger than 2^λ and $p^2 - 1 = aW$. We denote $W^- := \gcd(p-1, W)$ a factor of W. Applying Observation 3 to Lemma 1, we get

$$(\mathbb{Z}^*_{p,f})^{(a)} \simeq \begin{cases} a\mathbb{Z}_{p^2-1}(+) \simeq a\mathbb{Z}_{aW}(+) \simeq \mathbb{Z}_W(+) & \left(\frac{D}{p}\right) = -1 \\ a\mathbb{Z}_p(+) \times a\mathbb{Z}_{p-1}(+) \simeq \mathbb{Z}_p(+) \times \mathbb{Z}_{W^-}(+) & \left(\frac{D}{p}\right) = 0 \\ a\mathbb{Z}_{p-1}(+) \times a\mathbb{Z}_{p-1}(+) \simeq \mathbb{Z}_{W^-}(+) \times \mathbb{Z}_{W^-}(+) & \left(\frac{D}{p}\right) = 1. \end{cases}$$

In particular, we used above the fact that Observation 2 implies that $a\mathbb{Z}_{p-1}(+) \simeq \mathbb{Z}_{W^-}$ as explained next. Since $(p-1)(p+1) = aW$, all divisors of $p-1$ are divisors of aW. By definition of a and W in Definition 12, we also have that $\gcd(a, W) = 1$, which implies that any factor of $p-1$ divides either a or W, but *not both*. When we divide $p-1$ by all the common divisors with a, only the common divisors with W are left, which implies $(p-1)/\gcd(a, p-1) = \gcd(W, p-1) = W^-$. The proof of the lemma is now completed by Observation 2.

Corollary 1. *Let p be a (λ, a_p)-strong prime, q be a (λ, a_q)-strong prime, $N = p \cdot q$, $a = \mathrm{lcm}(a_p, a_q)$, $P, Q \in \mathbb{Z}_N$ and $f(x) = x^2 - Px + Q$. Then $(Z^*_{N,f})^{(a)}$ is λ-strong.*

[8] Since we set a to be at most polynomial in λ, its is possible to go over all possible candidate values for a in time polynomial in λ. Thus, any algorithm that could factor N using the knowledge of a can be efficiently simulated even without the knowledge of a.

4.3 Our Interactive Protocol

Our interactive protocol is formally described in Figure 3. To understand this protocol, we first recall the outline of Pietrzak's interactive protocol from Section 1.2 and then highlight the hurdles. Let $N = p \cdot q$ be an RSA modulus where p and q are strong primes and let x be a random element from $\mathbb{Z}_N^*$. The interactive protocol in [42] allows a prover to convince the verifier of the statement

"(N, x, y, T) satisfies $y = x^{2^T} \bmod N$".

The protocol is recursive and in a round-by-round fashion reduces the claim to a smaller statement by halving the time parameter. To be precise, in each round the (honest) prover sends the "midpoint" $\mu = x^{2^{T/2}}$ of the current statement to the verifier and they together reduce the statement to

"$(N, x', y', T/2)$ satisfies $y' = (x')^{2^{T/2}} \bmod N$",

where $x' = x^r \mu$ and $y' = \mu^r y$ for a random challenge r. This is continued until $(N, x, y, T = 1)$ is obtained at which point the verifier simply checks whether $y = x^2 \bmod N$.

The main problem, we face while designing our protocol is ensuring that the verifier can check whether μ sent by prover lies in an appropriate subgroup of $\mathbb{Z}_N[\sqrt{D}]$. In the first draft of Pietrzak's protocol[41], prover sends a square root of μ, from which the original μ can be recovered easily (by simply squaring it) with a guarantee, that the result lies in a group of quadratic residues QR_N. Notice that the prover knows the square root of μ, because it is just a previous term in the sequence he computed.

Using Pietrzak's protocol directly for our delay function would require computing a-th roots in RSA group for some arbitrary a. Since this is a computationally hard problem, we cannot use the same trick. In fact, the VDF construction of Wesolowski [56] is based on similar hardness assumption.

While Pietrzak shifted from QR_N to the group of signed quadratic residues QR_N^+ in his following paper [42] to get unique proofs, we resort to his old idea of 'squaring a square root' and generalise it.

The high level idea is simple. First, on input ω, prover computes the sequence $(\omega, \omega^2, \ldots, \omega^{2^T})$. Next, during the protocol, verifier maps all elements sent by the prover by homomorphism

$$\psi : \mathbb{Z}_{N,f}^* \to (\mathbb{Z}_{N,f}^*)^{(a)}, \quad \psi(x) = x^a \tag{11}$$

into the target strong group $(\mathbb{Z}_{N,f}^*)^{(a)}$. This process is illustrated in Figure 2. Notice that the equality $y = \omega^{2^T}$ for the original sequence implies the equality $y^a = (\omega^a)^{2^T}$ for the mapped sequence $(\omega^a, \omega^{2a}, \ldots, \omega^{a2^T})$.

Restriction to Elements of $(\mathbb{Z}_{N,f}^*)^{(a)}$. Mapping Equation (11) introduces a new technical difficulty. Since ψ is not injective, we narrow the domain inputs, for which the output of our VDF is verifiable, from $\mathbb{Z}_{N,f}^*$ to $(\mathbb{Z}_{N,f}^*)^{(a)}$. Furthermore,

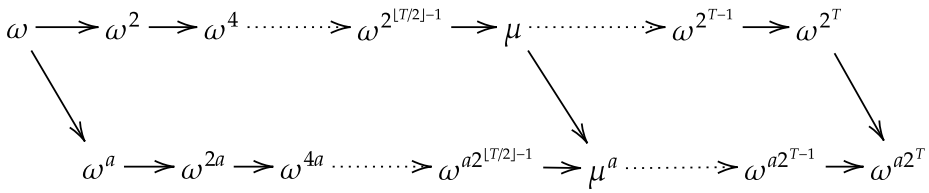

Fig. 2. Illustration of our computation of the iterated squaring using the a-th root of ω. Horizontal arrows are $x \mapsto x^2$ and diagonal arrows are $x \mapsto x^a$.

the only way to verify that a certain x is an element of $(\mathbb{Z}^*_{N,f})^{(a)}$ is to get an a-th root of x and raise it to the ath power. So we have to represent elements of $(\mathbb{Z}^*_{N,f})^{(a)}$ by elements of $\mathbb{Z}^*_{N,f}$ anyway. To resolve these two issues, we introduce a non-unique representation of elements of $(\mathbb{Z}^*_{N,f})^{(a)}$.

Definition 13. *For $a \in \mathbb{N}$ and $x \in \mathbb{Z}^*_{N,f}$, we denote x^a (an element of $(\mathbb{Z}^*_{N,f})^{(a)}$) by $[x]$. Since this representation of x^a is not unique, we define an equality relation by*

$$[x] = [y] \overset{def}{\leftrightarrow} x^a = y^a$$

We will denote by tilde ($\tilde{x}$) the elements that were already powered to the a by a verifier (i.e. $\tilde{x} = x^a$). Thus tilded variables verifiably belong to the target group $(\mathbb{Z}^*_{N,f})^{(a)}$.

In the following text, the goal of the brackets notation in Definition 13 is to distinguish places where the equality means the equality of elements of $Z^*_{N,f}$ from those places, where the equality holds up to $\mathrm{Ker}(\psi)$. A reader can also see the notation in Definition 13 as a concrete representation of elements of a factor group $\mathbb{Z}_{N,f}/\mathrm{Ker}(\psi)$.

Our security reduction 2 required the delay function to operate everywhere on $\mathbb{Z}_N$. This is not a problem if the `LCS.Setup` algorithm is modified to output the set $\mathrm{Ker}(\psi)$.

4.4 Security

Recall here that $(\mathbb{Z}^*_{N,f})^{(a)}$ is λ-strong group, so there exist $p_1, \ldots, p_m \in \mathbb{P} \cap (2^\lambda, \infty)$ and $k_1, \ldots, k_m \in \mathbb{N}$ such that

$$(\mathbb{Z}^*_{N,f})^{(a)} \simeq \mathbb{Z}_{p_1^{k_1}}(+) \times \cdots \times \mathbb{Z}_{p_m^{k_m}}(+) \tag{12}$$

Definition 14. *For $z \in (\mathbb{Z}^*_{N,f})^{(a)}$ and $i \in [m]$, we define z_i as i-th coordinate of $\psi(z)$, where ψ is the isomorphism given by Equation* (12).

Setting

$n \in \mathbb{N}$ computational security parameter
$\lambda \in \mathbb{N}$ statistical security parameter
$a \in \mathbb{N}$ exponentiation parameter
$N \in \mathbb{N}$ a product of n-bit (λ, a_p)-safe prime p and
(λ, a_q)-safe prime q such that $a = \mathrm{lcm}(a_p, a_q)$
$f(x) = x^2 - Px + Q$ for some $P, Q \in \mathbb{Z}_N$
$((N, a), [\omega], T)$ a challenge tuple
$((N, a), [\omega], T, [y])$ a solution tuple

The Interactive Protocol

1. Prover and verifier get a challenge tuple $((N, a), [\omega], T)$ as a common input.
2. Prover computes the sequence
$$\omega \to \omega^2 \to \omega^4 \to \ldots \to \omega^{2^T}$$
and sends its last element $[y] := [\omega^{2^T}]$ to the verifier.
3. Prover and verifier repeat the halving protocol, initiated with solution tuple $((N, a), [\omega], T, [y])$, until verifier either `accept` or `reject`.

The Halving Protocol

1. Prover and verifier get solution tuple $((N, a), [\omega], T, [y])$ as common input.
2. If $T = 1$, then the verifier computes $(\tilde{\omega}, \tilde{y}) = (\omega^a, y^a)$ and it outputs `accept` provided that $\tilde{y} = \tilde{\omega}^2$ or `reject` otherwise.
3. Prover sends $[\mu] := [\omega^{2^{\lfloor T/2 \rfloor}}]$ to verifier.
4. If $\mu \notin \mathbb{Z}^*_{N,f}$, then verifier output `reject`.
5. Verifier picks a random r from uniform distribution on $\mathbb{Z}_{2^\lambda}$ and he sends r to the prover.
6. Finally prover and verifier merge solution tuples
$$((N, a), [\omega], \lfloor T/2 \rfloor, [\mu]) \qquad \text{and}$$
$$\begin{cases} ((N, a), [\mu], T/2, [y]) & \text{for even } T \\ ((N, a), [\mu], \lceil T/2 \rceil, [y^2]) & \text{for odd } T \end{cases}$$
into the new solution tuple
$$\begin{cases} ((N, a),\ [\omega^r \mu], T/2, [\mu^r y]) & \text{for even } T \\ \big((N, a),\ [\omega^r \mu], \lceil T/2 \rceil, [(\mu^r y)^2]\big) & \text{for odd } T. \end{cases}$$

Fig. 3. Our Interactive Protocol for `LCS`.

Lemma 3. *Let $T \in \mathbb{N}$ and $\omega, \mu, y \in (\mathbb{Z}^*_{N,f})^{(a)}$. If $y \neq \omega^{2^T}$, then*

$$\Pr\left[\begin{array}{r} y' = (\omega')^{2^{T/2}} \\ \textit{where} \\ r \leftarrow \mathbb{Z}_{2^\lambda} \\ \omega' := \omega^r \mu \\ y' := \mu^r y \end{array}\right] < \frac{1}{2^\lambda}. \tag{13}$$

Proof. Fix ω, μ and y. Let some $r \in \mathbb{Z}_{2^\lambda}$ satisfy

$$\mu^r y = (\omega^r \mu)^{2^{T/2}}. \tag{14}$$

Using notation from Definition 14, we rewrite Equation (14) as a set of equations

$$\begin{aligned} r\mu_1 + y_1 &\equiv 2^{T/2}(r\omega_1 + \mu_1) \mod p_1^{k_1}, \\ &\vdots \\ r\mu_m + y_m &\equiv 2^{T/2}(r\omega_m + \mu_m) \mod p_m^{k_m}. \end{aligned}$$

For every $j \in [m]$, by reordering the terms, the j-th equation becomes

$$r(2^{T/2}\omega_j - \mu_j) + (2^{T/2}\mu_j - y_j) \equiv 0 \mod p_j^{k_j} \tag{15}$$

If $\forall j \in [m] : 2^{T/2}\omega_j - \mu_j \equiv 0 \bmod p_j^{k_j}$, then $\mu = \omega^{2^{T/2}}$. Further for every $j \in [m] : 2^{T/2}\mu_j - y_j \equiv 0 \bmod p_j^{k_j}$. It follows that $y = \mu^{2^{T/2}}$. Putting these two equations together gives us $y = \omega^{2^T}$, which contradicts our assumption $y \neq \omega^{2^T}$.

It follows that there exists $j \in [m]$ such that

$$2^{T/2}\omega_j - \mu_j \not\equiv 0 \mod p_j^{k_j}. \tag{16}$$

Thereafter there exists $k < k_j$ such that p_j^k divides $(2^{T/2}\omega_j - \mu_j)$ and

$$(2^{T/2}\omega_j - \mu_j)/p_j^k \not\equiv 0 \mod p_j. \tag{17}$$

Furthermore, from Equation (15), p_j^k divides $(2^{T/2}\mu_j - y_j)$. Finally, dividing eq. Equation (15) by p_j^k, we get that r is determined uniquely (mod p_j),

$$r \equiv -\frac{(2^{T/2}\mu_j - y_j)/p_j^k}{(2^{T/2}\omega_j - \mu_j)/p_j^k} \mod p_j.$$

Using the fact that $2^\lambda < p_j$, this uniqueness of r upper bounds number of $r \in \mathbb{Z}_{2^\lambda}$, such that Equation (14) holds, to one. It follows that the probability that Equation (14) holds for r chosen randomly from the uniform distribution over $\mathbb{Z}_{2^\lambda}$ is less than $1/2^\lambda$. □

Corollary 2. *The halving protocol will turn an invalid input tuple (i.e. $[y] \neq [\omega^{2^T}]$) into a valid output tuple (i.e. $[y'] = [(\omega')^{2^{T/2}}]$) with probability less than $1/2^\lambda$.*

Theorem 3. *For any computationally unbounded prover who submits anything other than $[y]$ such that $[y] = [\omega^{2^T}]$ in phase 2 of the protocol, the soundness error is upper-bounded by $\log(T)/2^\lambda$*

Proof. In each round of the protocol, T decreases to $\lceil T/2 \rceil$. It follows that the number of rounds of the halving protocol before reaching $T = 1$ is upper bounded by $\log T$.

If the verifier accepts the solution tuple $((N, a), [\omega''], 1, [y''])$ in the last round, then the equality $[y''] = [(\omega'')^2]$ must hold. It follows that the initial inequality must have turned into equality in some round of the halving protocol. By Lemma 3, the probability of this event is bounded by $1/2^\lambda$. Finally, using the union bound for all rounds, we obtain the upper bound $(\log T)/2^\lambda$. □

4.5 Our VDF

Analogously to the VDF of Pietrzak [42], we compile our public-coin interactive proof given in Figure 3 into a VDF using the Fiat-Shamir heuristic. The complete construction is given in Figure 4. For ease of exposition, we assume that the time parameter T is always a power of two.

As discussed in Section 4.3, it is crucial for the security of the protocol that the prover computes a sequence of powers of the a-th root of the challenge and the resulting value (as well as the intermediate values) received from the prover is lifted to the appropriate group by raising it to the a-th power. We use the tilde notation in Figure 4 in order to denote elements on the sequence relative to the a-th root.

Note that, by the construction, the output of our VDF is the $a \cdot 2^T$-th power of the root of the characteristic polynomial for Lucas sequence with parameters P and Q. Therefore, the value of the delay function implicitly corresponds to the $a \cdot 2^T$-th term of the Lucas sequence.

Theorem 4. *Let λ be the statistical security parameter. The* LCS *VDF defined in Figure 4 is correct and statistically-sound with a negligible soundness error if* hash *is modelled as a random oracle, against any adversary that makes $o(2^\lambda)$ oracle queries.*

Proof. The correctness follows directly by construction.

To prove its statistical soundness, we proceed in a similar way to [42]. We cannot apply Fiat-Shamir transformation directly, because our protocol does not have constant number of rounds, thus we use Fiat-Shamir heuristic to each round separately.

First, we use a random oracle as the hash function. Second, if a malicious prover computed a proof accepted by verifier for some tuple $((N, a), ([\omega], T), [y])$ such that

$$[y] \neq [\omega^{2^T}], \tag{19}$$

then he must have succeeded in turning inequality from Equation (19) into equality in some round. By Lemma 3, probability of such a flipping is bounded by $1/2^\lambda$. Every such an attempt requires one query to random oracle. Using a union bound, it follows that the probability that a malicious prover who made q queries to random oracle succeeds in flipping initial inequality into equality in some round is upper-bounded by $q/2^\lambda$.

`LCS.Setup`$(1^n, 1^\lambda)$**:** Runs a strong primes generator on input $1^n, 1^\lambda$ to get an n-bit (λ, a_p)-strong prime p and an n-bit (λ, a_q)-strong prime q, such that $N = p \cdot q$ is a Blum integer. Then it chooses a hash function

$$\text{hash} : \mathbb{Z} \times \mathbb{Z}_N^3 \to \mathbb{Z}_{2^\lambda}.$$

and it outputs the public parameters

$$(N := p \cdot q, a := \text{lcm}(a_p, a_q), \text{hash}).$$

`LCS.Gen`(N, T)**:** Samples P and Q independently from the uniform distribution on $\mathbb{Z}_N$, sets

$$D := P^2 - 4Q \quad \text{and} \quad \omega := \frac{P + z}{2},$$

where z is a formal variable satisfying $z^2 = D$, and outputs (ω, T).

`LCS.Eval`$(N, (\omega, T))$**:** Computes the sequence

$$\omega \to \omega^2 \to \omega^4 \to \omega^8 \to \ldots \to \omega^{2^{T-1}} \to \omega^{2^T}$$

and outputs its last term ω^{2^T}.

`LCS.Prove`$((N, a, \text{hash}), ([\omega], T))$**:** Computes

$$y := [\omega^{2^T}]$$

and sets $(\omega_1, y_1) = (\omega, y)$. For $i = 1, \ldots, t$ computes

$$\begin{aligned} \mu_i &:= \omega_i^{2^{T/2^i}}, \\ r_i &:= \text{hash}(T/2^{i-1}, \omega_i^a, y_i^a, \mu_i^a), \\ \omega_{i+1} &:= \omega_i^{r_i} \mu, \\ y_{i+1} &:= \mu^{r_i} y_i. \end{aligned}$$

It outputs $([y], \pi = ([\mu_1], \ldots, [\mu_t]))$.

`LCS.Verify`$((N, a, \text{hash}), ([\omega], T), ([y], \pi))$**:** Sets $\tilde{\omega}_1 = \omega^a$, $\tilde{y}_1 = y^a$ and for each $i = 1, \ldots, t$, computes

$$\begin{aligned} \tilde{\mu}_i &:= \mu_i^a, \\ r_i &:= \text{hash}(T/2^{i-1}, \tilde{\omega}_i, \tilde{y}_i, \tilde{\mu}_i), \\ \tilde{\omega}_{i+1} &:= \tilde{\omega}_i^{r_i} \tilde{\mu}_i, \\ \tilde{y}_{i+1} &:= \tilde{\mu}_i^{r_i} \tilde{y}_i. \end{aligned}$$

It outputs `accept` if

$$\tilde{y}_{t+1} = \tilde{\omega}_{t+1}^2, \tag{18}$$

otherwise it outputs `reject`.

Fig. 4. `VDF` based on Lucas sequences

Since q is $o(2^\lambda)$, $q/2^\lambda$ is a negligible function and thus the soundness error is negligible. □

5 Open Problems

In this work, we constructed verifiable delay functions from modular Lucas sequences, which are linear recurrences of order two, and gave a reduction from the sequentiality of iterated squaring to the sequentiality of our delay function. A natural question to ask is if our approach can be generalised to linear recurrences of higher order. In Appendix C we show that in certain groups one can soundly verify linear recurrences of order $k > 2$. However, not all linear recurrences satisfy the sequentiality property. Constructing instances of higher-order linear recurrences that are provably sequential remains an open problem.

Acknowledgements. We thank Alon Rosen and Krzysztof Pietrzak for several fruitful discussions. We also thank the anonymous reviewers of SCN 2022 and TCC 2023 for valuable suggestions.

Chethan Kamath is supported by Azrieli International Postdoctoral Fellowship, by the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101042417, acronym SPP), and by ISF grant 1789/19.

References

1. Abusalah, H., Kamath, C., Klein, K., Pietrzak, K., Walter, M.: Reversible proofs of sequential work. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 11477, pp. 277–291. Springer (2019)
2. Aggarwal, D., Maurer, U.: Breaking RSA generically is equivalent to factoring. IEEE Trans. Inf. Theory **62**(11), 6251–6259 (2016). https://doi.org/10.1109/TIT.2016.2594197, https://doi.org/10.1109/TIT.2016.2594197
3. Arun, A., Bonneau, J., Clark, J.: Short-lived zero-knowledge proofs and signatures. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 487–516. Springer (2022). https://doi.org/10.1007/978-3-031-22969-5‟17, https://doi.org/10.1007/978-3-031-22969-5_17
4. Bernstein, D., Sorenson, J.: Modular exponentiation via the explicit chinese remainder theorem. Mathematics of Computation **76**, 443–454 (2007). https://doi.org/10.1090/S0025-5718-06-01849-7
5. Bitansky, N., Choudhuri, A.R., Holmgren, J., Kamath, C., Lombardi, A., Paneth, O., Rothblum, R.D.: PPAD is as hard as LWE and iterated squaring. IACR Cryptol. ePrint Arch. p. 1072 (2022)
6. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: ITCS. pp. 345–356. ACM (2016)

7. Bleichenbacher, D., Bosma, W., Lenstra, A.K.: Some remarks on lucas-based cryptosystems. In: CRYPTO. Lecture Notes in Computer Science, vol. 963, pp. 386–396. Springer (1995)
8. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: CRYPTO (4). Lecture Notes in Computer Science, vol. 12828, pp. 123–152. Springer (2021)
9. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer (2018)
10. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. IACR Cryptology ePrint Archive **2018**, 712 (2018)
11. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring. In: Nyberg, K. (ed.) Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding. Lecture Notes in Computer Science, vol. 1403, pp. 59–71. Springer (1998). https://doi.org/10.1007/BFb0054117, https://doi.org/10.1007/BFb0054117
12. Buchmann, J., Williams, H.C.: A key-exchange system based on imaginary quadratic fields. J. Cryptol. **1**(2), 107–118 (1988)
13. Chávez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In: SAC. Lecture Notes in Computer Science, vol. 13203, pp. 441–460. Springer (2021)
14. Choudhuri, A.R., Hubáček, P., Kamath, C., Pietrzak, K., Rosen, A., Rothblum, G.N.: PPAD-hardness via iterated squaring modulo a composite. IACR Cryptology ePrint Archive **2019**, 667 (2019)
15. Cini, V., Lai, R.W.F., Malavolta, G.: Lattice-based succinct arguments from vanishing polynomials. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 72–105. Springer Nature Switzerland, Cham (2023)
16. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 10821, pp. 451–467. Springer (2018)
17. Cohen, B., Pietrzak, K.: The Chia network blockchain. Tech. rep. (2019), https://www.chia.net/assets/ChiaGreenPaper.pdf, Accessed: 2022-07-29
18. Döttling, N., Garg, S., Malavolta, G., Vasudevan, P.N.: Tight verifiable delay functions. In: SCN. Lecture Notes in Computer Science, vol. 12238, pp. 65–84. Springer (2020)
19. Döttling, N., Lai, R.W.F., Malavolta, G.: Incremental proofs of sequential work. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 11477, pp. 292–323. Springer (2019)
20. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 12107, pp. 125–154. Springer (2020)
21. Feo, L.D., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 11921, pp. 248–277. Springer (2019)
22. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
23. Freitag, C., Pass, R., Sirkin, N.: Parallelizable delegation from LWE. IACR Cryptol. ePrint Arch. p. 1025 (2022)

24. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
25. Hoffmann, C., Hubáček, P., Kamath, C., Klein, K., Pietrzak, K.: Practical statistically sound proofs of exponentiation in any group. In: CRYPTO. Lecture Notes in Computer Science, vol. 13508, pp. 1–30 (2022)
26. Hofheinz, D., Kiltz, E.: The group of signed quadratic residues and applications. In: CRYPTO. Lecture Notes in Computer Science, vol. 5677, pp. 637–653. Springer (2009)
27. Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: Pass, R., Pietrzak, K. (eds.) Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12552, pp. 390–413. Springer (2020). https://doi.org/10.1007/978-3-030-64381-2"14, https://doi.org/10.1007/978-3-030-64381-2_14
28. Lai, R.W.F., Malavolta, G.: Lattice-based timed cryptography. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 782–804. Springer Nature Switzerland, Cham (2023)
29. Lehmer, D.H.: An extended theory of Lucas' functions. Annals of Mathematics **31**(3), 419–448 (1930), http://www.jstor.org/stable/1968235
30. Lennon, M.J.J., Smith, P.J.: LUC: A new public key system. In: Douglas, E.G. (ed.) Ninth IFIP Symposium on Computer Security. p. 103–117. Elsevier Science Publishers (1993)
31. Lenstra, A.K., Wesolowski, B.: Trustworthy public randomness with sloth, unicorn, and trx. IJACT **3**(4), 330–343 (2017)
32. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 2894, pp. 398–415. Springer (2003)
33. Lombardi, A., Vaikuntanathan, V.: Fiat-shamir for repeated squaring with applications to ppad-hardness and vdfs. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 12172, pp. 632–651. Springer (2020)
34. Lucas, E.: Théorie des fonctions numériques simplement périodiques. American Journal of Mathematics **1**(4), 289–321 (1878), http://www.jstor.org/stable/2369373
35. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)
36. Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: ITCS. pp. 373–388. ACM (2013)
37. Mahmoody, M., Smith, C., Wu, D.J.: A note on the (im)possibility of verifiable delay functions in the random oracle model. IACR Cryptology ePrint Archive **2019**, 663 (2019)
38. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 11692, pp. 620–649. Springer (2019)
39. Müller, W.B., Nöbauer, W.: Some remarks on public-key cryptosystems. Studia Sci. Math. Hungar. **16**, 71–76 (1981)
40. P., C., Bressoud, D.M.: Factorization and primality testing. Mathematics of Computation **56**(193), 400 (1991)
41. Pietrzak, K.: Simple verifiable delay functions. IACR Cryptology ePrint Archive **2018**, 627 (2018), https://eprint.iacr.org/2018/627/20180720:081000
42. Pietrzak, K.: Simple verifiable delay functions. In: ITCS. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)

43. Rabin, M.O.: Transaction protection by beacons. J. Comput. Syst. Sci. **27**(2), 256–267 (1983)
44. Ribenboim, P.: My Numbers, My Friends: Popular Lectures on Number Theory. Springer-Verlag New York (2000)
45. Riesel, H.: Prime numbers and computer methods for factorization (1985)
46. Rivest, R., Silverman, R.: Are 'strong' primes needed for rsa. Cryptology ePrint Archive, Report 2001/007 (2001), https://eprint.iacr.org/2001/007
47. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems (reprint). Commun. ACM **26**(1), 96–99 (1983)
48. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., Massachusetts Institute of Technology (1996)
49. Rotem, L., Segev, G.: Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 481–509. Springer International Publishing, Cham (2020)
50. Rotem, L., Segev, G., Shahaf, I.: Generic-group delay functions require hidden-order groups. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12107, pp. 155–180. Springer (2020)
51. Rotem, L., Segev, G., Shahaf, I.: Generic-group delay functions require hidden-order groups. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 12107, pp. 155–180. Springer (2020)
52. Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.R.: Randrunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In: 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society (2021)
53. Seres, I.A., Burcsi, P.: A note on low order assumptions in RSA groups (2020), https://eprint.iacr.org/2020/402
54. Shani, B.: A note on isogeny-based hybrid verifiable delay functions. IACR Cryptology ePrint Archive **2019**, 205 (2019)
55. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC. Lecture Notes in Computer Science, vol. 4948, pp. 1–18. Springer (2008)
56. Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 11478, pp. 379–407. Springer (2019)
57. Wesolowski, B.: Efficient verifiable delay functions. J. Cryptol. **33**(4), 2113–2147 (2020), https://doi.org/10.1007/s00145-020-09364-x
58. Williams, H.C.: A $p + 1$ method of factoring. Math. Comput. **39**(159), 225–234 (1982)
59. Williams, H.C.: Édouard lucas and primality testing. The Mathematical Gazette **83**, 173 (1999)

A Generating Strong Primes

We propose two algorithms for sampling strong primes necessary for security of our verifiable delay function. Recall that if p is a (λ, a)-strong prime, then $p^2 - 1$ has only a few small factors and all remaining factors are larger than 2^λ. From

$p^2 - 1 = (p-1)(p+1)$ and $gcd(p-1, p+1) = 2$, it follows that $p^2 - 1$ always has at least two large factors.

Naive algorithm. To sample a random strong prime, we can simply pick some random n-bit pseudo-prime and check if it is a strong prime. First, the algorithm finds a small factors of $p-1$ by trial division and then it checks whether the co-factor $p^- := (p-1)/a^-$ is a prime (where a^- denotes the product of all small factors). Identical test is done for $p+1$. If p succeeds in both tests, then p is a $(\lambda, a^- a^+)$-strong prime.

Algorithm 1 (Naive strong primes generator)
Input: *Security parameters* $1^n, 1^\lambda$.
Output: *Integer tuple* (p, a) *such that* p *is* (λ, a)*-strong prime.*

1. *Pick an* n*-bit pseudo-prime* p.
2. *Factorize* $p-1$ *to product* $a^- p^-$*, where factors of* a^- *are smaller than* λ*. If* p^- *is not a pseudo-prime, goto Item 1.*
3. *Factorize* $p+1$ *to product* $a^+ p^+$*, where factors of* a^+ *are smaller than* λ*. If* p^+ *is not a pseudo-prime, goto Item 1.*
4. *Output* $(p, a^- a^+)$.

Enhanced algorithm. Note that for every p output by the naive algorithm (Algorithm 1), both $p-1$ and $p+1$ have exactly one large prime divisor. We can increase the efficiency of our strong primes generator by allowing $p-1$ and $p+1$ to have two large prime factors. Our enhanced algorithm first fixes p^-, a large prime divisor of $p-1$, and p^+, a large prime divisor of $p+1$. Using the Chinese Remainder Theorem (CRT), the algorithm then computes a p such that

$$\begin{aligned} p &\equiv +1 \mod p^-, \\ p &\equiv -1 \mod p^+ \end{aligned}$$

and it adds $(p^- p^+)$ to p until p is a prime.

During the next step, the algorithm checks whether $(p-1)/(a^- p^-)$ (resp. $(p+1)/(a^+ p^+)$) is a prime, where a^- (resp. a^+) is the product of all small factors of $(p-1)/b^-$ (resp. $(p+1)/b^+$) found by trial division. If at least one of these tests fails, then the algorithm keeps on adding $(p^- p^+)$ to p to get get another prime.

Algorithm 2 (Enhanced strong primes generator)
Input: *Security parameters* $1^n, 1^\lambda$.
Output: *Integer tuple* (p, a) *such that* p *is* (λ, a)*-strong prime.*

1. *Choose two random* n*-bits pseudo-primes* p^+ *and* p^-.
2. *Using CRT compute* p *s.t.*

$$p \equiv +1 \ mod\ p^- \qquad (i.e.,\ \exists a^- \ : p-1 = b^- p^-) \quad and$$
$$p \equiv -1 \ mod\ p^+ \qquad (i.e.,\ \exists a^+ \ : p+1 = b^+ p^+).$$

3. *While* p *is not a pseudo-prime:* $p \leftarrow p + (p^- p^+)$.
4. *Set* $b^+ := (p-1/p^-)$ *and* $b^+ := (p+1)/p^+$.
5. *Find all factors of* b^- *(resp.* b^+*) smaller than* λ. *We denote product of these small factors* a^- *(resp.* a^+*).*
6. *If* b^-/a^- *is not a prime or* b^+/a^+ *is not a prime,*
   - *then* $p \leftarrow p + p^+ p^-$ *and goto step. Item 3.*
   - *Otherwise return* (p, a)*, where* $a = a^- a^+$.

In practice, the primality tests performed both in Algorithm 1 and Algorithm 2 would be implemented via a probabilistic primality test such as the Rabin-Miller test. As a proof of concept, we provide some strong primes generated using these two algorithms in the subsequent Appendix A.1.

A.1 Strong Primes

Here we provide two examples of 1000-bit strong primes p and q which were respectively generated by the naive algorithm (Algorithm 1) and the enhanced algorithm (Algorithm 2). Both primes were generated on `Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz` in order of minutes using a non-optimized implementation in Python language.

$p = 164085922465767264569699321791946741063666556217835106048358267877577579355432286287370821698233693249787730659261218535109656726892396444183489757576699371496058498738776381121912968714343938103223781485730129446100301122864628024924538827741957556943380929448283532844233385323309124719640161469841327$

To enable one to verify strong primality of q, we attach factorization of $q-1$ and $q+1$.

$q = 234310875683355859353017539445184202322643928469640122820578995172828328102897933036376469140764836174213983164443102567714128700069899255085953490924420980521750963534770027053835151857348809030770591135971774823388233385706974253990681529145758352001482553018067333107608724264088480643904786550719593786 03$

$q-1 = 6 \times 11822526874984019332382339925306971459701892301154737945668930164788495715146587969779448349380662375233290499508686850487804475801126211387024059597783077341 \times 330316970532963411704221820987300359 3$

14773977698446822810166305718575766464335772692853747888099647462896338169747895947003092333681615154510579518718 7

$q+1 = 4\times$ 96514525260340898538356635691712174068423026703959658983005506958628390006670917098005964661744495469121308078474608858407983642377471303252973183552673640 7 $\times$ 60693163814285815334827546554817051090377133240560566613715642950225042432228983797102438335390019190970282785334261596189115445805977677305365442284 93

B Safe primes modulus

In this section we explore how reduce requirements on modulus at the expense of losing statistical soundness. Let N be product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$. Let P be sampled from uniform distribution on $\mathbb{Z}_N$, D be sampled from uniform distribution on

$$\{D \in \mathbb{Z}_N \mid \left(\frac{D}{N}\right) = 1\}$$

and set $Q := \frac{1}{4}(P^2 - D)$. Since $\left(\frac{D}{N}\right) = 1$, there are two possible cases:

1. $\left(\frac{D}{p}\right) = -1$ and $\left(\frac{D}{q}\right) = -1$ or
2. $\left(\frac{D}{p}\right) = 1$ and $\left(\frac{D}{q}\right) = 1$

Similarly to proof of Lemma 2,

$$(\mathbb{Z}^*_{N,f})^{(a)} \simeq \begin{cases} a\mathbb{Z}_{p^2-1}(+) \times a\mathbb{Z}_{q^2-1}(+) & \text{in case 1} \\ (a\mathbb{Z}_{p-1}(+))^2 \times (a\mathbb{Z}_{q-1}(+))^2 & \text{in case 2} \end{cases},$$

Since the modulus N is composed of safe primes instead of strong primes, group $(\mathbb{Z}^*_{N,f})^{(a)}$ can contain subgroups of low order in case 1. We show that the protocol remains computationally secure assuming that every polynomial adversary has only negligable advantage in the following decisional problem:

Problem 1 (Quadratic residuocity problem). Given an RSA modulus N and a sampled from uniform distribution on $\mathbb{Z}_N$ such that $\left(\frac{a}{N}\right) = 1$, decide whether a is a quadratic residue mod N.

Definition 15. *Let $n \in \mathbb{N}$ be a security parameter and $N = \Theta(2^n)$. We say that τ is a low-order element of $\mathbb{Z}_N$, if there exists a $e = poly(\lambda)$ such that $\tau^e = 1$.*

Claim. If there is exists a polynomial malicious prover P^* who is able break the protocol, then there exists a polynomial algorithm A for computing low-order elements in $(\mathbb{Z}^*_{N,f})^{(a)}$.

Proof (Sketch). As far as y and x^{2^T} differ in in some large component, it is statistically impossible that we will get an equality $y' = (x')^{2^T/2}$ at the end of the

halving protocol. Therefore y and x^{2^T} can differ only in some small component and the desired low-order element can be extracted as $y^{-1}x^{2^T}$.

Formal version of this claim with a complete proof can be found in [10]. There is an enhanced version of this claim in [53], which shows a sub-exponential lower-bound on order of element returned by extractor.

Claim. If there exists a polynomial algorithm A for computing low-order elements in $(\mathbb{Z}^*_{N,f})^{(2)}$, then there exists a polynomial algorithm B solving the Quadratic residuocity problem for safe primes RSA modulus.

Proof. For given N, D, algorithm B samples a random P and compute $Q := \frac{1}{4}(P^2 - D) \bmod N$. B runs algorithm A for group $(\mathbb{Z}^*_{N,f})^{(2)}$ getting element τ. Algorithm B verifies output of A by verifying $\tau^r = 1$ for random $r \leftarrow \mathbb{Z}_{2^\lambda}$. Let us denote order of τ by e. If e is polynomial in λ, then this test is true positive with non-negligible probability $1/e$ (=1/poly(λ)). Otherwise if e is superpolynomial, then this test is false negative with only negligible probability $1/e$.

If τ passes the low-orderity test, B output 0 (D is not quadratic residuo mod N). Otherwise B output 1. We split the computation of success rate of B into two cases.

In case 1, B's success rate is equal to A's success rate times success rate of the low-orderity test (=$1/e$).

In case 2, there are no low-order elements in $(\mathbb{Z}^*_{N,f})^{(2)}$ and thus output of A cannot be valid. It follows that success rate of B in this case is equal to true negativeness of the low-orderity check, which is $1 - \text{negl}(\lambda)$.

Since these two cases occurs with the same probability, this computation concludes in

$$\frac{1}{2}(1 - \text{negl}(\lambda)) + \frac{1}{2}\text{Succ}(A)\frac{1}{\text{poly}(\lambda)} = \frac{1}{2} + \frac{1}{\text{poly}(\lambda)}$$

success rate of B, where Succ(A) denotes (non-negligible) success rate of A.

C On Linear Recurrences of Higher Order

In this chapter, we discuss a possible extension of our approach to linear recurrences with order greater then two. In particular we show that one can soundly verify higher-order linear recurrences in $\mathbb{Z}_N$ if all prime factors of N are so called *generalized strong primes.*

Definition 16. *Let $\boldsymbol{R}$ be a ring and $k \in \mathbb{N}$. The linear recurrence $(s_i)_{i\in\mathbb{N}_0}$ of order k is given by set of initial conditions $b_0, \ldots, b_{k-1} \in \boldsymbol{R}$ s.t. $\forall i \in 0, \ldots, k-1 :$ $s_i = b_i$ and coefficients of linear recurrence $a_0, \ldots, a_{k-1} \in \boldsymbol{R}$ s.t.*

$$\forall i > k : \; s_i = a_{k-1}s_{i-1} + \ldots + a_0 s_{i-k}$$

Definition 17. *We define the characteristic polynomial of linear recurrence*

$$f(x) := x^k - a_{k-1}x^{k-1} - a_{k-2}x^{k-2} + \ldots + a_0.$$

Linear recurrences modulo p are known as linear-feedback shift registers (LSFR) and they are well studied.

Proposition 1 (Galois representation of LSFR). *Let $(s_i)_{i\in\mathbb{N}_0}$ be linear recurrence and $f(x)$ be its characteristic polynomial. There exists a polynomial $g^{(0)}(x)$ of degree $k-1$ s.t. for every $i \in \mathbb{N}_0$ the leading coefficient of*

$$g^{(i)}(x) := x^i g^{(0)}(x) \mod f(x)$$

is equal to s_i.

Corollary 3. *The n-th element of linear recurrence can be computed in $O(\log(n))$ sequential steps.*

Fact 1 *Let $p \in \mathbb{P}$, $f(x) \in \mathbb{Z}_p[x]$ be a monic polynomial of degree k, $r_1(x) \cdot \ldots \cdot r_\ell(x)$ be the factorization of $f(x)$ to irreducible factors. We denote $d_i = deg(r_i(x))$. Then*

$$\mathbb{Z}_{p,f} \simeq \bigtimes_{i=1}^{\ell} \mathbb{Z}_{p,r_i} \simeq \bigtimes_{i=1}^{\ell} \mathbb{Z}_{p^{d_i}-1}(+).$$

We derive sufficient conditions on factors of N to be able to use our protocol with recurrences of higher order.

First of all, notice that the fact that f has degree 2 is used for proving strength of $(\mathbb{Z}^*_{p,f})^{(a)}$, but it is no longer needed for protocol security proof. It follows that if we are able to find p and a, $a = poly(\lambda)$, such that $(\mathbb{Z}^*_{p,f})^{(a)}$ is λ-strong group for all polynomials of degree d over $\mathbb{Z}_p$, then we are able to apply our protocol to recurrences of order d. Fact 1 gives us characterization of primes that satisfy this property.

If we restrict ourselves to recurrences of order d with irreducible characteristic polynomial, then p must be (λ, a, d)-strong prime for some $a \in \mathbb{N}$ to achieve λ-strength of $(\mathbb{Z}^*_{p,f})^{(a)}$ and thus λ-bit security, where (λ, a, d)-strong primality is defined as follows:

Definition 18 (Generalized strong primes). *Let $p \in \mathbb{P}$ and $\lambda, a, d \in \mathbb{N}$. We say that p is a (λ, a, d)-strong prime, if $\lambda > a$ and there exists $W \in \mathbb{N}$ such that $p^d - 1 = aW$ and every factor of W (excluding number 1) is greater than 2^λ.*

If we want to compute recurrences of order d for any characteristic polynomial, then p must be a (λ, a_i, d_i)-strong prime for every $d_i \in [d]$.

Generalized strong primes can be generated by analog of naive algorithm (algorithm 1).