FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Tomáš Krňák

Verifiable Delay Functions from Lucas sequences

Computer Science Institute of Charles University

Supervisor of the master thesis:	Mgr. Pavel Hubáček, Ph.D.
Study programme:	Mathematics
Study branch:	Mathematics for Information Technologies

Prague 2022

I would like to thank my advisor, Mgr. Pavel Hubáček, Ph.D., for his guidance support throughout this thesis. I really appreciate he "took me under his wings" during my studies and he brought many interesting opportunities to my academic journey. I would also like to dedicate a special thanks to Chethan Kamath, Ph.D. for his assistance and consultations during writing this thesis.

Title: Verifiable Delay Functions from Lucas sequences

Author: Bc. Tomáš Krňák

Department: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

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 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: Lucas sequences, verifiable delay function, modular squaring, strong primes

Contents

1. Introduction

1.1 Overview

A Verifiable Delay Function (VDF) is a function that satisfies two properties. First, it is a delay function, which means it must take a prescribed 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 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 iterated hashing 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 [LW17] and later formalised in [BBBF18]. In principle, since the task of constructing a VDF reduces to the task of incrementally-verifiable computation [Val08, BBBF18], constructions of VDFs could leverage succinct non-interactive arguments of knowledge (SNARKs): take any sequentially-hard function f (for instance the 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 [BBBF18], the resulting construction is not quite practical since we would rely on a general-purpose machinery of SNARKs with significant overhead. Moreover, the VDF construction would inherit the non-standard knowledge-of-exponent assumption on which the current constructions of SNARKs rely on.

Efficient VDFs via algebraic delay functions VDFs have recently found interesting applications in the area of cryptocurrency [CP19b], where efficiency is an important factor. This has resulted in a flurry of constructions that are tailored with the 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 simpler machinery than SNARKs to achieve verifiability. These constructions involve a range of algebraic setups like the RSA modulus [Pie19, Wes19], permutation polynomials over finite fields [BBBF18], and isogenies of elliptic curves [FMPS19, Sha19].

The constructions in [Pie19, Wes19] are arguably the most practical and the mechanism that underlies their delay function is the same: carry out iterated *squaring* in the RSA modulus or, more generally, any group of unknown order (e.g. class groups of an imaginary quadratic fields). What distinguishes these two proposals is the way verification is carried out: while Pietrzak [Pie19] resorts to an LFKN-style recursive proof system [LFKN92], Wesolowski [Wes19] uses a clever linear decomposition of the exponent. The constructions from [FMPS19, Sha19], on the other hand, work in the algebraic setting of isogenies of elliptic curves. Since no analogue of square and multiply is known for this setting, these constructions simply rely on "exponentiation". That said, these constructions are far from being practically efficient.

Squaring and sequentiality Since the delay function that underlies the VDF in both [Pie19] and [Wes19] is the same, they also end up relying on the same hardness assumption, that iterated squaring is inherently sequential in a group of unknown order (suggested in the context of time-lock puzzles by Rivest, Shamir, and Wagner [RSW96]). As shown in [BS07], modular exponentiation, and hence modular repeated squaring, is amenable to parallelism when carried out in batches. Therefore, it would be prudent to have efficient VDFs based on other atomic operations than modular squaring. This constitutes one of the main motivations of our work and, as we show, it is indeed possible *without* having to sacrifice efficiency by a large margin.

1.2 Our Approach to VDFs

In this work, we construct verifiable delay functions that are of a different flavour compared to the known constructions. The sequentiality of our delay function relies on an atomic operation that is related to the computation of so-called Lucas sequences [Luc78, Leh30], explained next. Integer sequences like Fibonacci numbers and Mersenne numbers are special cases of Lucas sequences.

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

$$x_n = Px_{n-1} - Qx_{n-2}$$

for integers P and Q. Specifically, we focus on Lucas sequences U_k and V_k defined bellow.

Definition 1. *For $P, Q \in \mathbb{Z}$ we define Lucas sequences of integers $(U_k(P,Q))_{k\in\mathbb{N}}$ and $(V_k(P,Q))_{k\in\mathbb{N}}$ recursively by*

$$\begin{aligned} &V_0 = 2,\ V_1 = P, V_k = PV_{k-1} - QV_{k-2} \quad \textit{and} \\ &U_0 = 0,\ U_1 = 1, U_k = PU_{k-1} - QU_{k-2}. \end{aligned}$$

The sequences can be alternatively defined by the characteristic polynomial $z^2 - Pz + Q$. Specifically, given the discriminant $D = P^2 - 4Q$ of the characteristic polynomial, one can alternatively compute the sequences in the extension field

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

using the identities

$$U_n = \frac{\omega^n - \overline{\omega}^n}{\omega - \overline{\omega}} \quad \text{and} \quad V_n = \omega^n + \overline{\omega}^n, \tag{1.1}$$

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., $\overline{\omega^n} = \overline{\omega}^n$), computing the n-th terms of the two Lucas sequences over integers reduces to computing ω^n 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 [LS93, BBL95]. We briefly discuss the analogue of RSA encryption system. Henceforth, we focus on the V_n sequence, but our observations mostly apply to U_n too.

LUC encryption scheme When the Lucas sequence is computed modulo an integer N chosen as in the RSA encryption scheme, it gives rise to an interesting cycle structure. The exact structure depends on the choice of N, P and Q but, with high probability the cycle is of length $\Psi(N)$, where $\Psi(\cdot)$ is the generalised totient function for Lucas sequences [Leh30]. In particular, for any $k \in \mathbb{N}$,

$$V_{k\Psi(N)+1} = V_1 \bmod N. \tag{1.2}$$

This allows defining an analogue of the RSA encryption scheme in the setting of the Lucas sequence modulo N. The key pair is (e, d) such that $ed = 1 \bmod \Psi(N)$ and the encryption of a message m works by simply computing $c = V_e(m, 1)$, the e-th term in the Lucas sequence defined by $P = m$ and $Q = 1$. The decryption can be carried out by computing $V_d(c, 1)$, the d-th term in the Lucas sequence defined by $P = c$ and $Q = 1$. The correctness is guaranteed by the identity in eq.1.2.

Overview of our VDF Our VDF builds on the construction of Pietrzak [Pie19] and Wesolowski [Wes19], where the delay function is defined as the iterated squaring base x in a "safe" RSA modulus N:

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

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 the 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 already observed, computing this function can be carried out equivalently in the extension field $\mathbb{Z}_N[\sqrt{D}]$ using

$$f_N(P, Q, T) := \omega^{2^T} \bmod (N, z^2 - D).$$

Note that the atomic operation of our VDF is "doubling" the index of an element of the Lucas sequence modulo N (i.e., $(U_n, V_n) \mapsto (U_{2n}, V_{2n})$), 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

$$(a + b\sqrt{D})^2 = (a^2 + b^2 D) + 2ab\sqrt{D}. \tag{1.3}$$

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 3.3 that iterated squaring in $\mathbb{Z}_N[\sqrt{D}]$ is at least as hard as iterated squaring modulo N.

As for the second property of a VDF, i.e., efficient verifiability, we show how to adapt the interactive protocol from [Pie19] to our setting which then – via the Fiat-Shamir Transform [FS86] – yields the non-interactive verification algorithm. 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 the next section.

Advantages of our approach.

- Our main advantage is the reliance of potential weaker (sequential) hardness assumption: we show in section 3.3 that modular Lucas sequences are *at least* as sequentially-hard as the classical delay function given by iterated modular squaring [RSW96]. 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 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, shared generation of RSA modulus is cost-intensive (see [CCD$^+$20] and the references therein) and therefore a modulus, say in blockchain applications, is usually generated for long-term usage and reusability in mind. Once the modulus is fixed, the operations are also inadvertently fixed. This unfortunately would allow optimising dedicated hardware for the atomic operation. However, in our case there is another free parameter D that is yet to be set (see eq. 1.3) which then determines the atomic operation even after the modulus N has been fixed. This parameter could be varied from protocol execution to protocol execution. Therefore, we believe that our setting might offers some level of mitigation against both these issues. Note that this comes at a slight price of working in the extension field, where the size of the elements and the cost of an atomic operation is, roughly speaking, twice that of working in the multiplicative group modulo N.

1.3 Our Techniques

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 from the uniform distribution over $\mathbb{Z}_{2^\lambda}$. This is continued until $(N, x, y, T = 1)$ is obtained, at which point, the verifier simply checks whether $y = x^2 \bmod N$.

Since the challenges r are public, the protocol can be compiled into a non-interactive one using the Fiat-Shamir transform [FS86] 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 of the above protocol.

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

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

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 [Pie19] 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 [Pie19], this was accomplished by moving to the setting of so-called signed quadratic residues [HK09] in which the sub-groups are all of large order. It is not clear a corresponding object exists for our algebraic setting. However, in an earlier draft of Pietrzak's protocol[Pie18], the issue of low-order elements 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.

Square roots. 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. Prover then computes 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 powering to a.

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 modulus $\mathbb{Z}_N$ but also in the extension $\mathbb{Z}_N[\sqrt{D}]$. Thus, the traditional sampling algorithms that are used to sample strong primes (e.g., [RS01]) are not sufficient for our purposes. However, we show in Section 4.4 that sampling strong primes that suit our criteria can still be carried out efficiently.

Extensions First, we show that our construction can be extended to other linear recurrences than Lucas sequence in Chapter 5. Second, we prove in Appendix A that if the adaptive root assumption (introduced in [Wes19]) holds in RSA group then it also holds for Lucas sequences and thus Weselowski's protocol can be used for verification of our delay function. Third, we show in Appendix B that the strong primality requirement (introduced in section 4.2) can be relaxed to safe primality, assuming computational hardness of the Quadratic residuocity problem.

1.4 Related Work

Related work to VDFs The notion of VDFs was introduced in [LW17] and later formalised in [BBBF18]. VDFs are closely-related to the notions of time-lock puzzles [RSW96] and proofs of sequential work [MMV13]. 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 "map", in the sense that the output is not necessarily unique. Constructions of time lock puzzles are rare [RSW96, BGJ$^+$16], and there are known limitations: e.g, that it cannot exist in the random-oracle model [MMV13]. However, we know how to construct proofs of sequential work in the random-oracle model [MMV13, CP18, AKK$^+$19, DLM19].

Since VDFs have recently found important applications in the area of cryptocurrency [CP19b], there have been several candidate constructions. Among them, the most notable are the iterated-squaring based construction from [Pie19, Wes19], the permutation-polynomial based construction from [BBBF18] and the isogenies-based construction from [FMPS19, Sha19]. Isogenies-base constructions, where no analogue of square and multiply is known, simply rely on "exponentiation". Although, these constructions provide certain form of quantum resistance, they are presently far from efficient. On the other hand, constructions in [Pie19, Wes19] based on iterated-squaring are quite practical (see the survey [BBF18]) and the VDF currently being used for consensus in the cryptocurrency Chia[1] is basically their construction adapted to the algebraic setting of class groups [CP19b].

There are several other interesting use cases of VDFs. Servers can use VDFs to mitigate DoS attacks by challenging each new user with a VDF instance [RG21]. Publicly verifiable randomness beacons can be constructed via VDFs [SJH$^+$20, EFKP20]. So-called short-live signatures and short-live zero-knowledge proofs are constructed via VDFs in [ABC22]. This short-liveness means that after a specified period of time, the proof (resp. the signature) is no longer convincing.

[1]In particular, Chia blockchain uses this VDF to avert long range attacks, where an adversary tries to generate a chain that forked (from the chain seen by the honest parties) many blocks in the past [CP19a]. In the Chia network, it is difficult to extend a forked chain quickly, because every new block must be finalized by solving a VDF challenge.

As VDFs are becoming building blocks of many protocols, there also already exists whole spectrum of newly required features. A VDF is said to be unique if the proof that is used for verification is unique [Pie19]. A VDF is tight [DGMV19] if the gap between simply computing the function and computing it *with* a proof is small. Yet another extension is a continuous VDF, where all steps of the computation (i.e. $f(t')$ for $t' < t$) are publicly and continuously verifiable [EFKP20]. Cooperative VDF (coVDF) are designed to be computed by multiple parties in either serial or parallel manner[MQ21].

The feasibility of time-lock puzzles and proofs of sequential works were recently extend to VDFs. It was shown [RSS20] that the latter requirement, that is working in a group of unknown order is inherent in a black-box sense. It was shown in [DGMV19, MSW19] that there are barriers to constructing tight VDFs in the random-oracle model.

VDFs have also have surprising connection to complexity theory [CHK$^+$19, EFKP20].

Work related to Lucas sequence Lucas sequences have long been studied in the context of number theory: see for example [Rie85] or [Rib00] for a survey of its applications to number theory. Its earliest application to cryptography can be traced to the $(p+1)$ factoring algorithm [Wil82]. 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 [LS93, MN81]. It was later shown that some of these schemes can be broken or that the advantages it claimed were not present [BBL95]. Statistical Zero-Knowledge Arguments are constructed using Lucas sequences in [Lip03].

Lucas-Lehmer primality test [Leh27], based on Lucas sequences, is currently used to search for the biggest primes the human-kind knows [GIM18].

1.5 Organisation

We start with the prerequisite basic definitions (VDFs and interactive protocols) in Chapter 2. Then, we define a delay function based on Lucas sequences and discuss its sequentiality in Chapter 3. We describe how to turn this delay function into verifiable delay function in Chapter 4. Then, in Chapter 5, we show how the ideas can be extended to linear recurrences other than Lucas sequences. We end with some concluding remarks and further research directions in Chapter 6.

2. Preliminaries

2.1 Verifiable Delay Functions

We adapt the definition of verifiable delay functions from [BBBF18] 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*** `DF` *consists of a triple of algorithms* (`DF.Setup`, `DF.Gen`, `DF.Eval`) *such that*

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

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

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

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

Definition 3 (**Sequentiality**). *Let* `DF` *be a delay function. We say that* `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 $A = (A_0, A_1)$ using* poly(λ) *processors and running in time $O(T^\epsilon(\lambda))$, there exists a negligible function μ such that*

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

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

1. We require computing $\mathtt{DF}(x, T)$ 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 $\mathtt{DF}(x, T)$ 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 $O(T/\mathrm{loglog}\, T)$

using polynomial parallelism [BS07]. Thus, one cannot simply bound the sequential time of the adversary by $o(T)$. Similarly to [MT19], 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. Further, tuples of parameters (λ, T) cannot be arbitrary (e.g. consider constant T and infinitely growing λ). We force T to scale with growing λ by defining T as a polynomial in λ.

4. Without loss of generality, we assume that the size of pp is at least linear in λ and the adversary A does not have to get the unary representation of the security parameter 1^{λ} 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* verifiable delay function `VDF` *is a delay function* (`VDF.Setup`, `VDF.Gen`, `VDF.Eval`) *with two additional algorithms* `VDF.Prove` *and* `VDF.Verify` *such that*

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

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

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

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

Definition 6 (**Statistical soundness**)**.** *A verifiable delay function* `VDF` *is* statistically sound *if for every (potentially computationally unbounded) malicious prover* P^* *there exists a negligible function* $\mu(\lambda)$ *such that*

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

2.2 Proof Systems

An interactive protocol consists of a pair (P, V) of interactive Turing machines that are run on a common input I. The first machine is called the prover and is denoted by P, and the second machine, which is probabilistic, is called the verifier and is denoted by V.

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 $\{I, (\beta_1 1 = , \ldots 1 = , \beta_\ell), (\alpha_1 1 = , \ldots 1 = , \alpha_\ell)\}$. The interactive protocol is *public-coin* if β_i is a uniformly distributed random string in Σ^b.

The notion of an interactive proof for a language L is due to Goldwasser, Micali and Rackoff [GMR89].

Definition 7. *An interactive protocol* (P, V) *is a* ϵ-sound *interactive proof (IP) for L if:*

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

A non-interactive proof system involves the prover sending a single message to the verifier. We work in the CRS model where we assume that both prover and verifier have access to a common reference string (CRS). When the CRS is simply a uniformly random string from some domain $\mathscr{R}$, it is referred to as a *common random string.* We focus on *adaptive* proof systems where a cheating prover gets to see the CRS *before* forging a proof for a statement of its choice.

Definition 8. *A pair of machines* (P, V) *is an* ϵ-adaptively-sound *non-interactive proof system for a language L if* V *is probabilistic polynomial-time and the following three properties hold:*

- ***Completeness:*** *For every $I \in L$,*

$$\Pr_{R \leftarrow \mathscr{R}} [\mathsf{V}(I, \mathsf{P}(I, R), R) = 1] = 1,$$

where the probability is over the random choice of the CRS $R \in \mathscr{R}$.

- ***Soundness:*** *For every (potentially computationally unbounded) cheating prover strategy* $\widetilde{\mathsf{P}}$*,*

$$\Pr_{\substack{R \leftarrow \mathscr{R} \\ (I, \widetilde{\pi}) \leftarrow \widetilde{\mathsf{P}}(R)}} [\mathsf{V}(I, \widetilde{\pi}, R) = 1 \wedge I \notin L] \leq \epsilon(|I|).$$

3. DF 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. Further, we put forward a conjecture that our delay function candidate is even more robust that its predecessor proposed by Rivest, Shamir, and Wagner [RSW96]. In 4, we turn our delay function candidate into a *verifiable* delay function.

3.1 Atomic operation

Our DF is based on subsequences of the Lucas sequences, whose indexes are powers of two.

Definition 9. *For $P, Q \in \mathbb{Z}$ and $t \in \mathbb{N} \cup \{0\}$, we define subsequences (u_t) and (u_t) of Lucas sequences by*

$$u_t := U_{2^t}(P, Q) \quad \textit{and} \quad v_t := V_{2^t}(P, Q). \tag{3.1}$$

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

What is the underlying atomic operation our DF

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

There are several ways how to compute (u_t, v_t) in t sequential steps. In the following text, we describe two of them.

Extension ring based approach We can use the extension ring $\mathbb{Z}_N[\sqrt{D}]$, where $D := P^2 - 4Q$ is the discriminant of the characteristic polynomial $f(x) = x^2 - Px + Q$. The polynomial $f(x)$ has a root $\omega := (P + \sqrt{D})/2 \in \mathbb{Z}_N[\sqrt{D}]$. For $\forall n \in \mathbb{N}$ it holds that

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

Thus by iterated squaring of ω we can compute terms of our target subsequences. But what does the squaring in the extension ring means exactly? Let $\omega = a + b\sqrt{D}$ for some $a, b \in \mathbb{Z}_N$. Then

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

We end up with atomic operation

$$g : (a, b) \mapsto (a^2 + b^2 D, 2ab), \quad g^t(\left(\frac{P}{2}, \frac{1}{2}\right)) = \left(\frac{u_t}{2}, \frac{v_t}{2}\right). \tag{3.2}$$

Identities based approach There exists many useful identities for members of Lucas sequences, among of them

$$U_{m+n} = U_m V_n - Q^n U_{m-n}, \quad \text{and} \quad V_{m+n} = V_m V_n - Q^n V_{m-n}. \tag{3.3}$$

Setting $m = n$ we get

$$U_{2n} = U_n V_n, \quad \text{and} \quad V_{2n} = V_n^2 - 2Q^n. \tag{3.4}$$

These identities are not hard to derive and a reader can find a proof in [PB91] Lemma 12.5. Indexes are doubled on each of application of equations 3.4.

Definition 10. *For $t \in \mathbb{N} \cup \{0\}$, we define an auxiliary sequence $q_t := Q^{2^t}$.*

Using identities 3.4, we get recursive equations

$$u_{t+1} = u_t v_t, \quad v_{t+1} = v_t^2 - 2q_t \quad \text{and} \quad q_{t+1} = q_t^2. \tag{3.5}$$

We end up with the atomic operation

$$g : (u, v, q) \mapsto (uv, v^2 - 2q, q^2), \quad g^t((1, P, Q)) = (u_t, v_t, q_t). \tag{3.6}$$

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

$$q_t = \frac{v_t^2 - u_t^2 D}{4}, \tag{3.7}$$

which can be found as lemma 12.2 in [PB91]. On the other side, identity 3.7 is quite interesting, because it allows us to compute large powers of element $Q \in \mathbb{Z}_N$ only using Lucas sequences. We use this fact in the security reduction.

Conclusion Computing (u_t, v_t) in the extension field seems to be the most natural and time and space effective approach. Furthermore, writing the atomic operation as $\omega \mapsto \omega^2$ is very clear. In the rest of this thesis, we will follow this approach.

3.2 Construction

Construction 1 (`DF` from Lucas sequences).

`LCS.Setup`(1^λ): *Samples two λ-bit primes p and q and outputs $N := p \cdot q$.*

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

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

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

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

$$\omega \to \omega^2 \to \omega^4 \to \omega^8 \to \ldots \to \omega^{2^{T-1}} \to \omega^{2^T} \tag{3.8}$$

and returns its last term $y := \omega^{2^T}$.

3.3 Reduction to RSW puzzles

In order to prove sequentiality (def. 3) of `LCS`, we use `RSW` puzzles sequentiality, implicitly stated in [RSW96], as underlying hardness assumption.

Definition 11 (**`RSW` delay function**). *We define* `RSW` *delay function as follows*
`RSW.Setup`(1^λ) *outputs an RSA modulus* N *of bit length* λ.
`RSW.Gen`(N, T) *outputs a random element* x *from uniform distribution on* $\mathbb{Z}_N^*$.
`RSW.Eval`$(N, (x, T))$ *outputs* $y := x^{2^T} \mod N$.

Theorem 1. *If* `RSW` *delay function has the sequentiality property, then* `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

$$Q := x, \quad D := P^2 - 4Q, \quad \omega := (P + z)/2,$$

  where z is a formal variable satisfying $z^2 = D$, and it runs $A_1(N, \texttt{state}, (\omega, T))$ 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 (3.7).

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 $\mathbb{Z}_N$, and the distribution of ω produced by B_1 is statistically indistinguishable from the distribution of ω sampled by `LCS.Gen`. 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 opposite implication in theorem 1 is not true, i.e. breaking sequentiality `RSW` does not necessarily means breaking sequentiality of `LCS`. Let us state this conjecture.

Conjecture 1.1. *Sequentiality of* `LCS` *cannot be reduced to sequentiality of* `RSW`.

A clue to this conjecture is that while `RSW` is based only on multiplication in the group $\mathbb{Z}_N^*(\cdot)$, the `LCS` uses full arithmetic (addition and multiplication) of the commutative ring $\mathbb{Z}_N$.

Yet another nice feature of our construction is that the result from [BS07], where $O(T/\text{loglog}(T))$ algorithm for iterated squaring is proposed, does not apply to our delay function, because our atomic operation differs.

4. VDF from Lucas sequences

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

To construct a VDF from Lucas sequnces, we use an algebraic extension

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

where N is a RSA modulus and $P, Q \in \mathbb{Z}_N$. In this section, we describe the structure of 4.1.

From the knowledge of structure of 4.1 we conclude that using modulus N composed of safe primes (i.e., $p - 1$ has a large prime divisor) is necessary but not sufficient condition for security of our construction. We specify the sufficient conditions on factors of N in the following Section 4.2.

First, we introduce a simplifying notation for quotient rings.

Definition 12. *For $n \in \mathbb{N}$ and $f(x) \in \mathbb{Z}_n$, we denote by $\mathbb{Z}_{n,f}$ the quotient ring $\mathbb{Z}[x]/(n, f(x))$, where $(n, f(x))$ denotes the ideal of the ring $\mathbb{Z}[x]$ generated by n and $f(x)$.*

We denote the set of all primes by $\mathbb{P}$. 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 $p, q \in \mathbb{P}$, $N := p \cdot q$ 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

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

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 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 have some double root α. Therefore $f(x)$ 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 structure of $\mathbb{Z}_{p,x^2}$.

We will prove that the function ψ,

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

is an isomorphism. First, polynom $a + cx \in \mathbb{Z}_{p,x^2}$ is invertible iff $a \neq 0$ (inverse is $a^{-1} - a^{-2}cx$). For 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

$$|\mathbb{Z}_{p,x^2}^*(\cdot)| = p^2 - p = (p-1)\cdot p = |\mathbb{Z}_p^*(\cdot) \times \mathbb{Z}_p(+)|. \tag{4.2}$$

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

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

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

For sake of our construction, we need $\mathbb{Z}_{p,f}^*$ to contain a strong subgroup of order asymptotically linear in p. We mind a reader that our definition of strong primes is stronger, than definition by Rivest and Silverman in [RS01].

Definition 13 (**Strong groups**). *For $\lambda \in \mathbb{N}$, we say that a non-trivial group $\mathbb{G}$ is λ-strong, if the order of each subgroup of $\mathbb{H}$ (excluding the trivial subgroup) 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 $\mathbb{Z}_{p,f}^*$ will always contain groups of small order (e.g. $Z_2(+)$). To get rid of these, we descend into the subgroup of a-th powers of elements of $\mathbb{Z}_{p,f}^*$ using the following notion.

Definition 14. *For Abelian group $\mathbb{G}$ and $a \in \mathbb{N}$, we define $\mathbb{G}^{(a)}$, subgroup of $\mathbb{G}$, as $\{x^a \mid x \in \mathbb{G}\}$ in multiplicative notion, and $a\mathbb{G} := \{ax \mid x \in \mathbb{G}\}$ in additive notion.*

Further, we show (in lemma 2 bellow) that (λ, a)-strong primality (definition 15 bellow) is sufficient condition for $(\mathbb{Z}_{p,f})^{(a)}$ to be a λ-strong group.

Definition 15 (**Strong primes**). *Let $p \in \mathbb{P}$ and $\lambda, a \in \mathbb{N}$. We say that p is the (λ, a)-strong prime, if $\lambda > a$ and there exists $W \in \mathbb{N}$, $W > 1$, such that $p^2 - 1 = aW$ and every 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 factorization of N. However, we could allow a to be polynomial in λ. For the sake of simplicity of the definition 15, we rather use stronger condition $a < \lambda$.

Lemma 2. *Let p be a (λ, a)-strong prime and $f(x) \in \mathbb{Z}_p[x]$ be a polynomial of degree 2. 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.

It is a simple observation that for $\forall m, n \in \mathbb{N} : n\mathbb{Z}_m \simeq \mathbb{Z}_{m/\gcd(m,n)}$. We use this observation to conclude that every branch from lemma 1 gives a product of λ-strong groups. According to observation 2, these products are λ-strong groups.

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

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

4.3 Interactive protocol

We recall the outline of Pietrzak's interactive protocol from Section 1.3. 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 [Pie19] 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 till $(N, x, y, T = 1)$ 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[Pie18], 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.

$$\begin{tikzcd}
\omega \arrow[r] \arrow[dr] & \omega^2 \arrow[r] & \omega^4 \arrow[r, dotted] & \omega^{2^{\lfloor T/2\rfloor -1}} \arrow[r] & \mu \arrow[r, dotted] \arrow[dr] & \omega^{2^{T-1}} \arrow[r] & \omega^{2^T} \arrow[dr] & \\
& \omega^a \arrow[r] & \omega^{2a} \arrow[r] & \omega^{4a} \arrow[r, dotted] & \omega^{a2^{\lfloor T/2\rfloor -1}} \arrow[r] & \mu^a \arrow[r, dotted] & \omega^{a2^{T-1}} \arrow[r] & \omega^{a2^T}
\end{tikzcd}$$

Figure 4.1: 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 $x \mapsto x^a$.

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 [Wes19] 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 [Pie19] to get a proof uniqueness, 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

$$\psi : \mathbb{Z}_{N,f}^*(\cdot) \to (\mathbb{Z}_{N,f}^*)^{(a)}(\cdot), \quad \psi(x) = x^a \tag{4.3}$$

into the target strong group $(\mathbb{Z}_{N,f}^*)^{(a)}$. This process is illustrated in fig. 4.1. Notice that the equality $y = \omega^{2^T}$ for the original sequence implies the equeality $y^a = (\omega^a)^{2^T}$ for the mapped sequence $(\omega^a, \omega^{2a}, \ldots, \omega^{a2^T})$.

4.3.1 Restriction to elements of $(\mathbb{Z}_{N,f}^*)^{(a)}$

Mapping 4.3 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, the only way how 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 power it to the a. So we have to represent elements of $(\mathbb{Z}_{N,f}^*)^{(a)}$ by elements of $\mathbb{Z}_{N,f}^*$ anyway. To resolve these issues, we introduce a non-unique representation of elements of $(\mathbb{Z}_{N,f}^*)^{(a)}$.

Definition 16. *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*

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

In the following text, the goal of the brackets notion 16 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 notion 16 as a concrete representation of elements of a factor group $\mathbb{Z}_{N,f}/\mathrm{Ker}(\psi)$. We will denote by tilde $(\tilde{x})$ the elements that were already powered to the a by a verifier (i.e. $\tilde{x} = x^a$). Therefore tilded variables verifiably belong to the target group $(\mathbb{Z}_{N,f}^*)^{(a)}$.

Our security reduction 1 requires the DF to operate everywhere on $\mathbb{Z}_N$. This problem can be bypassed by modifying `LCS.Setup` to attach the auxiliary set $\mathrm{Ker}(\psi)$ to its output.

4.3.2 Description of interactive protocol for our VDF

Setup

$\lambda \in \mathbb{N}$	security parameter
$a \in \mathbb{N}$	exponentiation parameter
$N \in \mathbb{N}$	a product of (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 $P, Q \in \mathbb{Z}_N$
$((N, a, \mathrm{hash}), [\omega], T)$	a challenge tuple
$((N, a, \mathrm{hash}), [\omega], T, [y])$	a solution tuple

The Interactive Protocol

1. Prover and verifier get a challenge tuple $((N, a, \mathrm{hash}), [\omega], T)$ 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 $((N, a, \mathrm{hash}), [\omega], T, [y])$, until verifier either `accept` or `reject`.

The Halving Protocol

1. Prover and verifier get solution tuple $((N, a, \mathrm{hash}), [\omega], T, [y])$ 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

$$((N, a, \mathrm{hash}), [\omega], \lfloor T/2 \rfloor, [\mu]) \qquad \text{and}$$
$$\begin{cases} ((N, a, \mathrm{hash}), [\mu], T/2, [y]) & \text{for even } T \\ ((N, a, \mathrm{hash}), [\mu], \lceil T/2 \rceil, [y^2]) & \text{for odd } T \end{cases}$$

into the new solution tuple

$$\begin{cases} ((N, a, \mathrm{hash}),\ [\omega^r \mu], T/2, [\mu^r y]) & \text{for even } T \\ ((N, a, \mathrm{hash}),\ [\omega^r \mu], \lceil T/2 \rceil, [(\mu^r y)^2]) & \text{for odd } T \end{cases}$$

and they output this tuple.

4.3.3 Security proof

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

$$(\mathbb{Z}^*_{N,f})^{(a)} \simeq \mathbb{Z}_{p_1^{k_1}}(+) \times \cdots \times \mathbb{Z}_{p_n^{k_n}}(+) \tag{4.4}$$

Definition 17. *For $z \in (\mathbb{Z}^*_{N,f})^{(a)}$ and $i \in [n]$, we define z_i as i-th coordinate of $\psi(z)$, where ψ is the isomorphism given by eq.* (4.4)*.*

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

$$\Pr_{r \leftarrow \mathbb{Z}_{2^\lambda}} \begin{bmatrix} y' = (\omega')^{2^{T/2}} \\ \hfill \textit{where} \\ \hfill \omega' := \omega\mu^r \\ \hfill y' := \mu^r y \end{bmatrix} < \frac{1}{2^\lambda}. \tag{4.5}$$

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

$$\mu^r y = (\omega^r \mu)^{2^{T/2}}. \tag{4.6}$$

Using notion from def. 17, we rewrite eq. (4.6) as a set of equations

$$\begin{aligned} r\mu_1 + y_1 &\equiv 2^{T/2}(r\omega_1 + \mu_1) \mod p_1^{k_1}, \\ &\vdots \\ r\mu_n + y_n &\equiv 2^{T/2}(r\omega_n + \mu_n) \mod p_n^{k_n}. \end{aligned}$$

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

$$r(2^{T/2}\omega_j - \mu_j) + (2^{T/2}\mu_j - y_j) \equiv 0 \mod p_j^{k_j} \tag{4.7}$$

If $\forall j \in [n] : 2^{T/2}\omega_j - \mu_j \equiv 0 \bmod p_j^{k_j}$, then $\mu = \omega^{2^{T/2}}$. Further for every $j \in [n] : 2^{T/2}\mu_j - y_j \equiv 0 \bmod p_j^{k_j}$. It folows 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 $j \in [n]$ such that

$$2^{T/2}\omega_j - \mu_j \not\equiv 0 \mod p_j^{k_j}. \tag{4.8}$$

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

$$(2^{T/2}\omega_j - \mu_j)/p_j^k \not\equiv 0 \mod p_j, \tag{4.9}$$

Furthermore, from eq. (4.7), p_j^k divides $(2^{T/2}\mu_j - y_j)$. Finally, dividing eq. 4.7 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 eq. (4.6) holds, to one. It follows that the probability the eq. 4.6 holds for r is chosen randomly from uniform distribution over $\mathbb{Z}_{2^\lambda}$ is less then $1/2^\lambda$. □

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 then* $1/2^\lambda$*.*

Theorem 2. *For any computationally unbounded prover who submits anything else then* $[y]$ *such that* $[y] = [\omega^{2^T}]$ *in the phase 2 of the protocol, the probability that verifier accepts is upper-bounded by*

$$\frac{\log T}{2^\lambda}.$$

Proof. In the 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 $((N, a, \text{hash}), [\omega''], 1, [y''])$ 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 $(\log T)/2^\lambda$. □

4.4 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 λ-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 parameter* 1^λ*.*

Output: *Integer tuple* (p, a) *such that* p *is* (λ, a)*-strong prime.*

1. *Pick a* λ*-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 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 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 parameter* 1^λ.

Output: *Integer tuple* (p, a) *such that* p *is* (λ, a)*-strong prime.*

```
1. Choose two random $\lambda$-bits pseudo-primes $p^+$ and $p^-$.
2. Using CRT compute $p$ s.t.
       $p \equiv +1$ mod $p^-$     (i.e., $\exists a^- : p - 1 = b^- p^-$)   and
       $p \equiv -1$ mod $p^+$     (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 then $\lambda$. 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. 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 Appendix C.

4.5 Construction

Analogously to the VDF of Pietrzak [Pie19], we compile our public-coin interactive proof into a VDF using the Fiat-Shamir heuristic. The complete construction is given in Construction 2. For ease of exposition, we assume that the time parameter T is always a power of two.

Construction 2 (**`VDF` based on Lucas sequences**).

`LCS.Setup`(1^λ)**:** *Runs a strong primes generator on input* 1^λ *to get a* (λ, a_p)*-strong prime* p *and a* (λ, a_q)*-strong prime* q *and it sets* $N = p \cdot q$*. Then it chooses a hash function*

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

and it outputs the public parameters

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

`LCS.Gen`$((N, a, \mathrm{hash}), T)$ *is identical to* `LCS.Gen` *from construction 1.*

`LCS.Eval`$((N, a, \mathrm{hash}), (\omega, T))$**:** *is identical to* `LCS.Eval` *from construction 1.*

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

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

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

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

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

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

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

It outputs `accept` *if*

$$\tilde{y}_{t+1} = \tilde{\omega}_{t+1}^2, \tag{4.10}$$

otherwise it outputs `reject`*.*

Theorem 3. `LCS` *VDF from construction 2 is correct and statistically sound in random oracle model.*

Proof. The correctness follows directly by construction.

To prove a statistical soundness, we proceed in the way similar to [Pie19]. We cannot apply Fiat-Shamir transformation directly, because our protocol does not have constant number or 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

$$[y] \neq [\omega^{2^T}], \tag{4.11}$$

then he must have succeeded in turning inequality from eq. (4.11) 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 union bound, it follows that the probability that a malicious prover who made q queries to random oracle succeed in flipping initial inequality into equality in some round is upper-bounded by $q/2^\lambda$.

Since q is polynomial in λ, $q/2^\lambda$ is a negligible function and thus our protocol is statistically sound. $\square$

5. Linear recurrences of higher order

In this chapter we discuss extension of our approach to linear recurrences with order greater then 2.

Definition 18. *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 $b_0, \dots, b_{k-1} \in \boldsymbol{R}$ s.t. $\forall i \in 0, \dots, k-1 : s_i = b_i$ and coefficients of linear recurrence $a_0, \dots, a_{k-1} \in \boldsymbol{R}$ s.t.*

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

Definition 19. *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 of elements of a finite field $\mathbb{F}$ and $f(x) \in \mathbb{F}[x]$ be its characteristic polynomial. There exists a polynomial $g^{(0)}(x) \in \mathbb{F}[x]$ of degree $k-1$ such that 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 $a < \lambda$, 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 (defined bellow) for some $a \in \mathbb{N}$ to achieve λ-strength of $(\mathbb{Z}_{p,f}^*)^{(a)}$ and thus λ-bit security.

Definition 20 (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 is greater then 2^λ.*

If we want to compute recurrences of order d for any characteristic polynomial f and prove values of their terms by protocol 2 for group λ-strong $(Z^*_{p,f})^{(a)}(\cdot)$ (for some $a \in \mathbb{N}$), then p must be a (λ, a_i, d_i)-strong prime for every $d_i \in \{1, \ldots, d\}$.

Generalized strong primes can be generated by analogue of naive algorithm (algorithm 1). To test (λ, a, d) primality, number $p^d - 1$ is factorized to $p - 1$ and $p^{d-1} + p^{d-2} + \ldots + p + 1$. Each of these factors is cleaned from small factors and then its primality is checked.

6. Conclusion

In this thesis we have seen how VDFs can be based on alternative operation than iterated squaring without sacrificing efficiency a lot. There are several natural directions that could be further explored. Here we list a few:

1. Can our VDF be instantiated solely in the RSA modulus avoiding the extension field altogether? This would require a deeper understanding of the properties of Lucas sequences, in particular whether there exists an analogue of signed quadratic residues for Lucas sequence. This is also related to the question of whether our proofs can be made unique since Pietrzak manages to achieve uniqueness by moving to the algebraic setting of signed quadratic residues.

2. There have been recent progress [CCD$^+$20, CHI$^+$20] in efficient shared generation of the standard RSA modulus [BF01]. It is an interesting question whether any of these techniques can be used to sample strong primes required for our purposes or for that matter strong primes that are required in [Pie19].

3. Since Lucas sequences have a lot of structure, it is worth exploring whether this can be exploited to construct VDFs with additional properties like batching or aggregation.

Bibliography

[ABC22] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. Cryptology ePrint Archive, Report 2022/190, 2022. `https://ia.cr/2022/190`.

[AKK$^+$19] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible Proofs of Sequential Work. In *EUROCRYPT (2)*, volume 11477 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2019.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A Survey of Two Verifiable Delay Functions. *IACR Cryptology ePrint Archive*, 2018:712, 2018.

[BBL95] Daniel Bleichenbacher, Wieb Bosma, and Arjen K. Lenstra. Some Remarks on Lucas-Based Cryptosystems. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 386–396. Springer, 1995.

[BF01] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

[BGJ$^+$16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-Lock Puzzles from Randomized Encodings. In *ITCS*, pages 345–356. ACM, 2016.

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

[CCD$^+$20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and Abhi Shelat. Multiparty Generation of an RSA Modulus. *IACR Cryptology ePrint Archive*, 2020:370, 2020.

[CHI$^+$20] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthuramakrishnan Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. *IACR Cryptology ePrint Archive*, 2020:374, 2020.

[CHK$^+$19] Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via Iterated Squaring Modulo a Composite. *IACR Cryptology ePrint Archive*, 2019:667, 2019.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple Proofs of Sequential Work. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467. Springer, 2018.

[CP19a] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. 2019. `https://www.chia.net/assets/ChiaGreenPaper.pdf`.

[CP19b] Bram Cohen and Krzysztof Pietrzak. The Chia Network Blockchain, 2019.

[DGMV19] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight Verifiable Delay Functions. *IACR Cryptology ePrint Archive*, 2019:659, 2019.

[DLM19] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental Proofs of Sequential Work. *IACR Cryptology ePrint Archive*, 2019:650, 2019.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous Verifiable Delay Functions. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2020, Proceedings*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, 2020.

[FMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable Delay Functions from Supersingular Isogenies and Pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277. Springer, 2019.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GIM18] GIMPS. GIMPS Project Discovers Largest Known Prime Number: $2^{82,589,933}-1$, December 2018. `https://www.mersenne.org/primes/press/M82589933.html`.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[HK09] Dennis Hofheinz and Eike Kiltz. The Group of Signed Quadratic Residues and Applications. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653. Springer, 2009.

[Leh27] D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bulletin of the American Mathematical Society*, 33(3):327 – 340, 1927.

[Leh30] D. H. Lehmer. An Extended Theory of Lucas' Functions. *Annals of Mathematics*, 31(3):419–448, 1930.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic Methods for Interactive Proof Systems. *J. ACM*, 39(4):859–868, 1992.

[Lip03] Helger Lipmaa. On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2003.

[LS93] M. J. J. Lennon and P. J. Smith. LUC: A new public key system. In E. G. Douglas, editor, *Ninth IFIP Symposium on Computer Security*, page 103–117. Elsevier Science Publishers, 1993.

[Luc78] Edouard Lucas. Théorie des Fonctions Numériques Simplement Périodiques. *American Journal of Mathematics*, 1(4):289–321, 1878.

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

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In *ITCS*, pages 373–388. ACM, 2013.

[MN81] W. B. Müller and W. Nöbauer. Some remarks on public-key cryptosystems. *Studia Sci. Math. Hungar.*, 16:71–76, 1981.

[MQ21] Liam Medley and Elizabeth A. Quaglia. Collaborative Verifiable Delay Functions. In Yu Yu and Moti Yung, editors, *Information Security and Cryptology - 17th International Conference, Inscrypt 2021, Virtual Event, August 12-14, 2021, Revised Selected Papers*, volume 13007 of *Lecture Notes in Computer Science*, pages 507–530. Springer, 2021.

[MSW19] Mohammad Mahmoody, Caleb Smith, and David J. Wu. A Note on the (Im)possibility of Verifiable Delay Functions in the Random Oracle Model. *IACR Cryptology ePrint Archive*, 2019:663, 2019.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic Time-Lock Puzzles and Applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649. Springer, 2019.

[PB91] C. P. and David M. Bressoud. Factorization and Primality Testing. *Mathematics of Computation*, 56(193):400, 1991.

[Pie18] Krzysztof Pietrzak. Simple Verifiable Delay Functions. *IACR Cryptology ePrint Archive*, 2018:627, 2018.

[Pie19] Krzysztof Pietrzak. Simple Verifiable Delay Functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[RG21] Mayank Raikwar and Danilo Gligoroski. Non-Interactive VDF Client Puzzle for DoS Mitigation. In *European Interdisciplinary Cybersecurity Conference*, EICC, page 32–38, New York, NY, USA, 2021. Association for Computing Machinery.

[Rib00] Paulo Ribenboim. *My Numbers, My Friends: Popular Lectures on Number Theory.* Springer-Verlag New York, 2000.

[Rie85] Hans Riesel. Prime numbers and computer methods for factorization. 1985.

[RS01] Ron Rivest and Robert Silverman. Are 'Strong' Primes Needed for RSA. Cryptology ePrint Archive, Report 2001/007, 2001.

[RSS20] Lior Rotem, Gil Segev, and Ido Shahaf. Generic-Group Delay Functions Require Hidden-Order Groups. *IACR Cryptology ePrint Archive*, 2020:225, 2020.

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

[SB20] István András Seres and Péter Burcsi. A Note on Low Order Assumptions in RSA groups. 2020. `https://eprint.iacr.org/2020/402`.

[Sha19] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:205, 2019.

[SJH$^+$20] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. Cryptology ePrint Archive, Report 2020/942, 2020. `https://ia.cr/2020/942`.

[Val08] Paul Valiant. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[Wes19] Benjamin Wesolowski. Efficient Verifiable Delay Functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407. Springer, 2019.

[Wil82] Hugh C. Williams. A $p + 1$ method of factoring. *Math. Comput.*, 39(159):225–234, 1982.

A. Analogue for Wesolowski's construction

Since Weselowski's VDF [Wes19] can be instantiated over an arbitrary group of unknown order, we can construct its analogue for Lucas sequences while using arithmetic of $\mathbb{Z}[\sqrt{D}]$. Furthemore, it is possible to reduce the adaptive root game assumption (defined in [Wes19]) for $\mathbb{Z}_N^*[\sqrt{D}]$ to adaptive root game assumtion for $\mathbb{Z}_N^*$.

Theorem 4. *Adaptive root game for $\mathbb{Z}_N[\sqrt{D}]$ is not easier then adaptive root game for $\mathbb{Z}_N$.*

Proof. We will proceed similarly to the proof of theorem 1. Assume that there exists an adversary A with non-negligible advantage in the adaptive root game in $\mathbb{Z}_N[\sqrt{D}]$. We construct an adversary A' who has the same advantage in the adaptive root game in $\mathbb{Z}_N$.

We construct A' according to the diagram bellow.

$$\begin{tikzcd}
A & & A' & & \text{Oraculum} \\
& \xleftarrow{N} & & \xleftarrow{N} & N \leftarrow \mathtt{Gen}(1^\lambda) \\
& \xrightarrow{\omega := \frac{1}{2}(V_1 + U_1\sqrt{D})} & & \xrightarrow{Q := \frac{1}{4}(V_1^2 + DU_1^2)} & \\
& \xleftarrow{\ell} & & \xleftarrow{\ell} & \ell \leftarrow \mathbb{P} \cap [0, 2^\lambda] \\
& \xrightarrow{\epsilon := (V_k + U_k\sqrt{D})/2} & & \xrightarrow{C := \frac{1}{4}(V_k^2 + DU_k^2)} &
\end{tikzcd}$$

The algorithm A uses the identity 3.7 twice. First, when computing deriving Q from (U_1, V_1) and second, when deriving C from (U_k, V_k). If (ω, ϵ) returned by A satisfies $\epsilon^\ell = \omega$, then (Q, C) returned by A' satisfies $C^\ell = Q$. Thus success rate of A' in Adaptive root game for $\mathbb{Z}_N[\sqrt{D}]$ is equal to success rate of A in Adaptive root game for $\mathbb{Z}_N$.

□

B. Safe primes modulus

In this section we explore how reduce requirements on modulus at the expense of loosing 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 negligible chance to solve the following decision problem:

Definition 21 (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 22. *Let $\lambda \in \mathbb{N}$ be a security parameter and $N = \Theta(2^\lambda)$. 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 4.1. *If there is exists a polynomial malicious prover P^* who is able to break the halving protocol, then there exists a polynomial algorithm A for computing low-order elements in $(\mathbb{Z}_{N,f}^*)^{(a)}$.*

Proof sketch. As long 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 $y^{-1}x^{2^T}$. $\square$

Formal version of the claim 4.1 with a complete proof can be found in [BBF18]. There is an enhanced version of this claim in [SB20], which shows a sub-exponential lower-bound on order of element returned by extractor.

Claim 4.2. *Let N be a safe primes modulus and $D \in \mathbb{Z}_N$ is such that $\left(\frac{D}{N}\right) = 1$. If there exists a polynomial algorithm A for computing low-order elements in $(\mathbb{Z}[\sqrt{D}]^*)^{(2)}$, then there exists a polynomial algorithm B solving the Quadratic residuocity problem.*

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 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 $1 - \mathrm{negl}(\lambda)$.

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

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

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

C. Strong primes example

Here we provide two examples of 1000-bit strong primes as a proof of concept. Prime p was generated by the naive algorithm 1 and prime q was generated by the enhanced 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.

$p = 16408592246576726456969932179194674106366655621783510604835826787757579355432286287370821698233693249787730659261218535109656726892396444183489757576699371496058498738776381121912968714343938103223781485730129446100301122864628024924538827741957556943380929448283532844233385323309124719640161469841327$

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

$q = 23431087568335585935301753944518420232264392846964012282057899517282832810289793303637646914076483617421398316444310256771412870006989925508595349092442098052175096353477002705383515185734880903077059113597177482338823338570697425399068152914575835200148255301806733310760872426408848064390478655071959378603$

$q-1 = 6\times 118225268749840193323823399253069714597018923011547379456689301647884957151465879697794483493806623752332904995086885048780447580112621138702405959778307734 1 \times 3303169705329634117042218209873003593147739776984468228101663057185757664643357726928537478880996474628963381697478959470030923336816151545105795187187$

$q+1 = 4\times 96514525260340898538356635691712174068423026703959658983005506958628390006670917098005964661744495469121308078474608858407983642377471303252973183552673640 7 \times 6069316381428581533482754655481705109037713324056056661371564295022504243222898379710243833539001919097028278533426159618911544580597767730536544228493$