Not So Slowth: Invertible VDF for Ethereum 2.0

Dankrad Feist, Dmitry Khovratovich, Mary Maller, Kelly Olson, Pratyush Ranjan Tiwari
Ethereum Foundation, Supranational, Johns Hopkins University
{dankrad, dmitry.khovratovich, mary.maller}@ethereum.org, kelly@supranational.net, pratyush@cs.jhu.edu

August 2021

Abstract

We present a VDF protocol that incorporate challenging requirements from the Ethereum ecosystem, which includes low-latency proof generation and VDF hardware that is both inexpensive and secure in post-quantum settings. Our protocol utilizes an aggressively optimized iterative algebraic transformation combined with the recent proof aggregation technique enabling the proof latency overhead of only a few minutes over the primary computation. The protocol is currently undergoing final scrutiny before entering the production stage at Ethereum 2.0.

1 Introduction

Distributed consensus protocols often require not only fast computations for efficiency, but also guaranteed slow computations for security. When we say guaranteed slow computations we mean that there is a lower bound on the time the computation takes even if one has access to an unlimited number of cores; i.e., the computation cannot be parallelized. Slow computations are required when a party is supposed to produce an output that will eventually benefit some other participants, so that the slowness of computation would guarantee unpredictability and thus fairness.

An example of this requirement in Ethereum 2.0 is the following. A group of 32 validators progressively builds a chain of collective randomness, with one value O generated per epoch $\mathcal{E}$. This randomness is used to select, for example, which validator can add a block onto the blockchain and reap the rewards of doing so. If the randomness is unbiased then the frequency with which a validator is selected depends on their stake in the system. However, if a malicious actor were to bias the randomness, then they can sample many different strings of randomness and select the one which benefits them most. Currently the randomness O is given by the reveals from a RANDAO commit-reveal scheme used to generate a random number where the commits are inputs produced by the validators during $\mathcal{E}$. However this commit-reveal scheme is biasable: a malicious validator that plays last can choose whether or not to reveal and the result will be different based on their choice. To transform O into a form of unbiasable randomness, one solution is to pass O through a *verifiable delay function* (VDF) which is guaranteed to be slow to compute, such that all validators must choose whether to reveal or not *before* they know the output of the VDF. Note that O can be evaluated by any VDF evaluator for each epoch $\mathcal{E}$. It is expected that the lowest execution time will be achieved on specialized hardware, and such hardware should be available to the participants. Thus the challenge becomes how can we build an efficient and secure VDF with affordable low-latency hardware?

A VDF is a tuple (F, Π) of function F and non-interactive protocol Π which works as follows: (1) Prover runs F on challenge I and produces output O; (2) Prover engages in Π and produces a proof π that $F(I) = O$; (3) Verifier obtains (I, O, π) and verifies π. To qualify for a VDF, the proof protocol should be *complete* and *sound*, the proof should be *succinct*, *non-malleable* (to prevent manipulation with the VDF output) and allow *fast verification*. A number of VDF constructions has been proposed

in the recent past [LW17, Wes19, BBBF18, Pie19, FMPS19, EFKP20, LV20, DGMV20] with their own advantages and limitations.

In this document we will first clarify what target properties we require from the VDF. We then propose a delay function that is difficult to parallelise. Next we suggest a pre-quantum SNARK based solution for making our delay function publicly verifiable such that anyone can quickly check that the VDF is computed correctly without computed the function for themselves. Finally we consider how to upgrade our proving system in the future after quantum attacks become feasible.

Our target

In addition to generic VDF properties as described above, a VDF protocol that solves the problem for Ethereum 2.0 should additionally possess the following features:

- **Tight latency bounds**: the minimal time needed to compute the VDF with arbitrary parallelism on modern hardware should be close to the best known algorithm to date.
- **Minimum hardware**: as a benign VDF computation brings no benefit to the executor, it should as cheap as possible so that parties could afford it.
- **Succinct proofs**: the VDF outputs should be widely available to all Ethereum nodes, and their verification should not expose a DoS attack vector.
- **Quantum upgrade**: when/if the quantum computers become powerful enough it should be possible to switch to a post-quantum version of the protocol while retaining the deployed hardware.
- **Delay granularity**: it should be possible to select the VDF expected running time with sufficient precision in seconds.

While this is not am immediate requirement, we expect that the resulting protocol would suit a large number of applications as well as other blockchains currently united as the VDF Alliance.

Our solution

Answering the challenges above, we present a VDF protocol called RootPack, which has latency bounds with tightness factor 5, a 200 KiB proof computable on a small GPU cluster or ASIC with a few minutes time increase, and so that the hardware can be still used in the post-quantum setting.

2 Requirements

At a high level, a protocol party, when outputting value O, also proves a statement of kind

$$\text{I have spent at least } t \text{ time to produce } O.$$

It is not important how exactly O is computed but it *is* important that the parameter t be close, or at least lower bound the actual time spent. The applications of VDF also yield separate requirements for hardware (where it is implemented), security properties, and statement parameters, which are all listed below.

A VDF was first formally defined by Boneh, Bonneau, Bünz and Fisch [BBBF18] as follows:

Definition 1 (Verifiable Delay Functions)**.** *A verifiable delay function (VDF) consists of the following algorithms:*

- $\mathsf{Gen}(1^\lambda, \Delta) \to \mathsf{pp}$*: The puzzle generation algorithm takes as input a security parameter* 1^λ *and a difficulty parameter* Δ *and outputs public parameters* pp *which fix the domain* $\mathcal{X}$ *and range* $\mathcal{Y}$ *of the puzzle and other information required to compute a puzzle or verify a puzzle solution*

- $\mathsf{Eval}(\mathsf{pp}, x) \rightarrow (y, \pi)$*: The puzzle evaluation algorithm takes as input the public parameters* pp*, an input from the domain* x*. It outputs a puzzle solution* y *and a proof* π*.*
- $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) \rightarrow 0/1$*: The puzzle verification algorithm takes as input the public parameters* pp*, an input from the domain* x*, an input from the range* y *and a proof* π*. It outputs either* 0 *or* 1*.*

Additionally, VDFs must satisfy the correctness, soundness and sequentiality definitions as defined below.

Definition 2 (Correctness)**.** *A verifiable delay function is correct if* $\forall \lambda, \Delta$*,* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \Delta)$*, and* $\forall x \in \mathcal{X}$ *if* $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ *then* $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$*.*

Definition 3 (Soundness)**.** *For soundness it is required that an adversary can not get a verifier to accept an incorrect VDF solution.*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1 & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \Delta) \\ y \neq \mathsf{Eval}(\mathsf{pp}, x) & (x, y, \pi) \leftarrow \mathcal{A}(1^\lambda, \Delta, \mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

VDF security is described using a sequentiality game played between a challenger and an adversary where first the challenger sends the public parameter pp to the adversary. There is a pre-processing phase where adversary $\mathcal{A}_0$ pre-processes the public-parameter and passes on an advice string z to adversary $\mathcal{A}_1$. In the challenge phase, the challenger sends input $x \in \mathcal{Y}$ to $\mathcal{A}_1$. The adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ wins the game if it can output a correct VDF solution y and proof π for the input x. The sequentiality guarantee is that no adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ can win the above game with probability greater than $\mathsf{negl}(\lambda)$ given $\mathcal{A}_0$ runs in total time $O(\mathsf{poly}(\Delta, \lambda))$ and $\mathcal{A}_1$ runs in parallel time polynomial in Δ with at most polynomial in Δ processors.

2.1 Delay statements

A delay statement is a statement of the form

```
delay(input, difficulty) == output
```

or equivalently

$$F(I, d) = O$$

where I and O are integer (tuples) and d is a non-negative integer. The difficulty parameter regulates the execution time. We need certain delay granularity, i.e. the difficulty can vary with precision in seconds. This naturally suggests that F is an iteration of some function f with sufficient granularity. A single call to f is then declared the unit of difficulty d, with $d \approx 2^{24}$ corresponding to 1 second for our proposal. We restrict ourselves to delay statements with:

- 15 sec to 3 days **delay range** ($2^{28} \leq d \leq 2^{42}$);
- 1 sec **delay granularity** – ($d = 0 \pmod{2^{24}}$).

2.2 Delay hardware

The delay hardware is supposed to be a USB stick fitted with an ASIC implementing the VDF with an additional requirement: every second an intermediate output is streamed to the host in real time. As the hardware is not yet manufactured, it can be simulated by precomputing every about 2^{24}-th round outputs and reading the appropriate output every $2^{24} \cdot 64$ ns ($\approx 1s$).

2.3 Delay proofs

Existing VDF constructions rely on a separate proof system that proves correctness of the VDF computation and thus ensures the delay statement. Clearly, the security of VDF relies on the one of the proof system. The following requirements are given:

1. First, a proof system should provide reasonable security level in a widely accepted security model. A random oracle model is chosen for its simplicity and versatility. There are two flavours of it: *programmable* and *non-programmable ROM*. The former assumes we can dynamically select RO outputs in a simulation for the security proof, which simplifies the proof but formally contrasts with real-world instantiations of RO with hash functions, whose functionality is pre-fixed. The latter model is somewhat closer to the reality but such security proofs are not always available. We thus *prefer* the non-programmable ROM but it is not a strong requirement. Regarding the security level, at least 120 bits of security is required (possibly under additional assumptions on some primitives used in the system).

2. Secondly, a proof of a delay statement, should be *post-quantum secure*. Concretely, no quantum attacks should exist on underlying primitives, and additionally the security in the random oracle model, if being employed, should translate to quantum oracles. More precisely, at least 80 bits of **provable soundness in a quantum random oracle model** is required, with slight preference for non-programmable one.

3. The proof should be **compact**. It is understood that the post-quantum security makes it difficult to be truly succinct (e.g. a few kilobytes), but it should be possible to contain the proof into 200 kBytes.

4. The proof should be verified within very short time (**succinct verification**). Concretely, we require at most 20ms verification time on a Raspberry Pi 4 Model B (single thread, no overclocking).

2.4 Prover hardware system

We have a separate requirement to a potential prover hardware system. We expect it to be a prover rig with prover software producing delay proofs for $1 \leq u \leq 8$ always-active delay hardware units. The requirement is to design and implement a prover system with

- **Commodity hardware** – the prover rig is built from off-the-shelf commodity hardware and requires little assembly;
- **Low latency** – delay proofs are produced with a latency (relative to the delay function output produced by the delay hardware) of:
  - at most 10% of the delay time if $d \geq 2^{36}$ (i.e. more than an hour);
  - at most 20% of the delay time if $d \geq 2^{32}$ (more than a few minutes);
  - at most 40% of the delay time if $d \geq 2^{28}$ (all other cases);
- **Permissive licensing** – the prover software is open-sourced under the Apache 2.0 license.

3 RootPack (MinRoot+SnarkPack) as an optimal design

We have considered three groups of VDF constructions:

- RSA-based VDF.[Wes19, Pie19] We rule out these promising designs due to the complicated and error-prone setup procedure [CHI+20, DDK+21] and difficulty to upgrage the construction for the post-quantum setting.

- Isogeny-based VDF [FMPS19]. Again, this setup succumbs to quantum attacks. Moreover, our internal estimate concludes that the existing construction require an unprecented memory bandwidth in order to achieve the lowest latency.
- Verifiable computation (VC) based VDF [BBBF18]. In this setting we see a VDF computation as a program whose correctness is asserted via a proof of knowledge(PoK). This construction is the most attractive as we can use a pre-quantum proof system nowadays and switch to a post-quantum proof at any time without compromising the setup. Additionally, most PoK constructions admit succinct verifiers. The downside of the approach is that the prover cost can be linear or even superlinear in the VDF computation, and the latency bounds on the prover cost are much less tight, so that we have to make sure it is parallelizable enough to yield little latency overhead.

The work closest to our proposed construction is the incremental verifiable computation (IVC) and the verifiable computation-based VDFs approach from [BBBF18]. The ideas in our work are similar in the sense that first we require a sequential function which is iteratively computed. And then for each the computed iterations, the prover computes SNARK proofs to prove correctness. We propose using SnarkPack to aggregate these proofs which are computed every iteration. We also propose an optimal design where the sequential function, the proof system and the aggregation protocol work in tandem to satisfy the VDF requirements as discussed in the previous section. Moreover, we provide realistic performance estimates for our proposed constuction where an ASIC evaluates the VDF and the proof work is done by GPUs.

Invertible VC-VDF In order to have delay granularity and to make the program description small, we select $F = f^D$ for some function f and D being some integer that determines the delay. The function f should be injective as otherwise any second preimage yields another VDF instance for free, which we want to avoid. Thus the inverse f^{-1} exists. For the proof purpose it does not matter if we prove $F(I) = O$ or $F^{-1}(O) = I$, and a prover can select the cheapest of the two. Thus it is convenient to have f easily invertible such that a circuit for f^{-1} is smaller. On the other hand, not every possible input I should be a valid input, as otherwise one can just start from arbitrary O in the faster backward direction. With restricting the input the VDF flow is as follows:

1. **Setup** – Prover learns the input domain $\mathcal{W}$ and input density, i.e. the (approximate) fraction of elements $\mathcal{W}$ that can be selected as a seed.
2. **Start** – Prover is given *input predicate* ϕ over $\mathcal{W}$.
3. **Computation** – Prover selects I such that $\phi(I) = true$. Then he computes $O = F(I)$.
4. **Proof** – if $\chi(O) = true$, Prover creates an IVC proof π and submits (I, O, π) to Verifier.
5. **Verification** – Verifier checks π.

MinRoot design There exist several candidates for f in the literature with VeeDo and Sloth++ [BBBF18] being the most prominent. In the core, both transform the state S to S' so that the pair satisfies a low-degree polynomial relation $G(S, S') = 0$. In more details, the forward direction is some inverse to a power function, where the exponent is selected to be an injective mapping. When $p \not\equiv 1 \pmod 3$ we have:

$$\text{Sloth++} : (x_{i+1}, y_{i+1}) \leftarrow \underbrace{(x_i, y_i)^{(p^2+1)/4}}_{\text{over } \mathbb{F}_{p^2}} + (c_1, c_2)$$

$$\text{VeeDo} : (x_{i+1}, y_{i+1}) \leftarrow A \cdot \underbrace{(x_i^{(2p-1)/3}, y_i^{(2p-1)/3})}_{\text{both over } \mathbb{F}_p} + (c_{1,i}, c_{2,i})$$

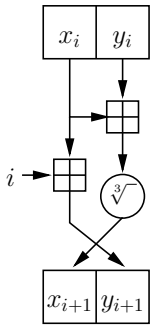

Figure 1: One round of MinRoot (cubic root version). Addition is in the field where the root is well defined.

Here (x_i, y_i) is the state at i-th step. We have found a better alternative to both, with only one exponentiation per round, which we call MinRoot (Figure 1):

$$\text{MinRoot} : (x_{i+1}, y_{i+1}) \leftarrow ((x_i + y_i)^{(2p-1)/3}, x_i) + (0, i) \tag{1}$$

If we have $p \equiv 1 \pmod 3$ but $p \not\equiv 1 \pmod 5$, which is the case for the primes which are prime group orders of popular BLS12-381 and BN254 curves, we use the fifth root as $(x_i + y_i)^{(4p-3)/5}$.

Hardware comparison We first observe that VeeDo uses two parallelizable exponentiations per round, whereas MinRoot does only one. Thus for the same delay time and same state size VeeDo requires twice the hardware, whereas the smaller state would reduce the security against precomputation attacks. Note also that VeeDo needs additional storage for constants (or hardware that computes those).

A comparison to Sloth++ is less straightforward. Assuming that both state elements in all designs are l-bit primes, we get that MinRoot uses one exponentiation to an l-bit exponent, VeeDo uses two, whereas Sloth++ computes a single $2l$-bit exponentiation of a $2l$-bit field element, which should yield twice longer step at CPU or an ASIC circuit twice as large. On the other hand, a hardware of the same size would imply twice smaller state, which succumbs to precomputation attacks. We conclude that MinRoot is more efficient in terms of hardware.

Actual performance

We assume that to compute a root function one uses a window-based square-and-multiply algorithm, which boils down to a sequence of modular multiplications. Our CPU benchmarks demonstrate that a naive 256-bit exponentiation takes about 10000 ns. An optimized ASIC implementation will be faster but given that at least 40 gates are needed for squaring [MOS20], 1μs is a reasonable estimate (i.e. 2^{20} calls to f per second and 2^{36} per day). A more conservative estimate would be to assume that the entire squaring can be done within a nanosecond cycle, so that one second constitutes to 2^{22} calls to f. More precise estimates will be made during the proof-of-concept stage.

3.1 SnarkPack

Design

In order to minimize the additional latency incurred by a VC proof it is desirable to (1) parallelize it and (2) start as early as possible. This makes the recursion schemes such as Halo [BDFG20] less suitable as they are inherently sequential and rather expensive. Instead, we opt for the recent SnarkPack design,

which combines the (fastest to-date) proofs [Gro16] with an aggregation protocol that can be efficiently pipelined and also has high concrete performance.

The SnarkPack protocol [GMN21, BMM+21] operates as follows:

1. Split the VDF computation into R chunks:

$$\widehat{F}(S_i) = S_{i+1} \tag{2}$$

   with $I = S_0$ and $O = S_R$.

2. Prepare a Groth16 proof π_i that (2) holds for each chunk.

3. Aggregate $\{S_i, \pi_i\}_i$ into a single proof π_{agg}. The cost is $O(R)$, the proof size and verification time both are $O(\log R)$.

Assuming each chunk consists of D/R calls to f, the SnarkPack *latency* is $O(D/R) + O(R)$, so that all but one Groth16 proofs can be prepared while the VDF function is still computed. Note that a *trusted setup* should be prepared for $\widehat{F}$ in the line of existing setup ceremonies.

Performance

Let us assume that a chunk lasts one minute, i.e. about 2^{26} calls to f. We plant to use the BLS12-381 curve, so the root function should be $x^{(4p-3)/5}$ and the inverse takes 3 constraints. Thus each MinRoot chunk requires $2^{27.5}$ constraints.

To understand the time it would take to prove such a large circuit, we can look at implementations used in practice by projects such as Filecoin. One of the most common SNARKs used on the Filecoin blockchain is a Proof-of-Replication proof, which requires the generation of 10 Groth16 proofs, each of size approximately 130 million, for a total of $\approx 2^{31}$ constraints. Recent benchmarks from the Filecoin community suggest that this proof can be created in less than 4 minutes using a modern CPU and GPU system, so our proof can be done in 15 seconds. A more conservative estimate (see above) with 2^{28} calls to f per chunk would imply that a GPU prover would spend as much time as the VDF evaluator. Since this VDF construction relies on generating many small proofs which are then aggregated, the proving operation can be easily parallelized by adding more proving hardware, such as a system with 4 GPUs, or by distributing the individual proof generation work across multiple 'workers'.

The aggregation is significantly cheaper and can be performed even on a single CPU for the entire VDF. Our own benchmarks show that aggregating ℓ proofs with Bellperson library takes approximately $\ell/2^6$ seconds, with the aggregated proof being $6 \log \ell$ KiB long and takes $10 \log \ell$ ms to verify. Therefore a 3-day aggregated proof for 2^{12} 1-minute chunks takes about 1 minute to generate, is 72 KiB long, and takes 0.1 sec to verify.

Quantum upgrade Even though the SnarkPack proof system is not quantum-resistant, there is no way to shortcut the computation of function F on a quantum computer. Some attacks become less expensive (see below) but not so significantly. Thus we can keep F and more importantly, the hardware computing it, and change the proof system only when the time comes. Of the post-quantum proof systems available today, STARKs [BBHR18] and its relatives Aurora [BCR+19] and Fractal [COS20] are suitable for this purpose. Concretely, we still prepare separate proofs for each chunk and then aggregate them using possibly the same proof system. With recent advance in the FFT computation [BSCKL21], the fact that the two proofs use different domains is no longer a problem.

3.2 Efficiency Improvements from Nova

The recently proposed Nova [KST21] proof-system, provides us an improvement over the Groth16 proofs. Nova does not require any FFTs for prover computation while also having the most optimal verifier circuit. The verifier circuit is constant sized and the cost mainly comes from two group

scalar multiplications. We implemented our invertible VC-VDFs over the Pasta curves [HBG20], with our MinRoot function and the Nova proof-system. The ASIC estimate for this design is 1ns per square/multiply. This means that the ASIC can takes 300ns per MinRoot iteration which gives us 3.3 million iterations/second. For the prover, each iteration is 3 constraints. To create a proof, a prover performs 2 multi-scalar multiplication/multi-exponentiation (MSMs) the size of the number of constraints. At 3.3 million iterations per second this *is* 2 MSMs of size 10^7 per second for 2×10^7 bases/second. We currently have GPU code that does 2×10^7 bases/second on a 384 bit curve and this should go up to about 4×10^7 bases/second when we move to256-bit curves. This means that a single GPU should be able to create SNARK proofs for 2 ASIC VDF evaluators.

4 Security of RootPack

4.1 Sequentiality (shortcut) attack

The sequentiality requirement to VDF exposes the primitive to a completely new class of attacks, where an adversary is able to compute F faster than expected. We formalize this requirement in the following definition.

Definition 4. *A VDF protocol* (F, Π) *is* (α, M, T)-sequential *if there is an adversary that (a) computes* M *calls to* F before *the Start step and (b) computes at most* M *calls to* F *within time* T *after the Start step – produces a valid proof* π *with probability* α*.*

We further say that a protocol has λ bits of *sequential security* if it is at most $(M2^{-\lambda}, M, l(F)/2)$-sequential, where $l(F)$ is the latency of F on common hardware. This means that after spending M calls to F before and after the start of the protocol, the adversary still has at most $M2^{-\lambda}$ chance to reduce the VDF computation cost by factor of 2.

Attack

A natural idea for a precomputation attack is to precompute some parts of F and then hope to meet those at some step. However, due to the use of a counter, each part can be met at only a certain step i. Assuming that the inverse of f costs only $\gamma < 1$ as much as the forward f, the attack proceeds as follows.

1. Compute $\frac{2M}{\gamma}$ chains of inverses of f of length $D/2$ starting with $i = D/2$ and ending with $i = 0$.
2. Store results as $\overline{\mathcal{W}} = \{(I', O')\}$.
3. When ϕ is known at the start phase, find M different values I such that $\phi(I) = true$ and see if there is a match with $\overline{\mathcal{W}}$. If found then use the entry to reduce the computing time by the factor 2.

The chance to meet a precomputed chain in the online phase is $\underbrace{\frac{2M}{\gamma}}_{\text{chains}} \cdot \underbrace{M}_{\text{starts}} \underbrace{2^{-2\ell}}_{\text{state size}}$, so F is at best

$$\left(\frac{2M^2}{\gamma 2^{-2\ell}}, M, l(F)/2\right)\text{-sequential.} \tag{3}$$

Let us estimate γ. We need 3 multiplications to compute x^3, 5 mults to compute x^5, and 1.25ℓ multiplications to compute either the cubic-root or the fifth root. Thus $\gamma > 2\ell$ so the VDF has λ bits of security for

$$\ell - \log \ell > \lambda$$

Thus we claim 110 bits of security for 128-bit primes and 240 bits of security for 256-bit primes.

4.2 Algebraic attacks

Algebraic attacks aim to find a shortcut in computing F by exploiting its algebraic properties. In order to speed up d rounds, i.e. computing f^d one should be able to solve equations of form

$$f^{-d}(I) = O \tag{4}$$

for given I.

If (4) is multivariate, then the best generic technique is applying Groebner basis algorithms [CLO97, Fau02]. These algorithms find a generating set for the ideal generated by (4) so that they can be solved as univariate equations. Little is known about the actual complexity of Groebner basis algorithms, but the available heuristic analysis suggests that it is exponential in d, thus quickly become impractical. Even in the unlucky case when a Groebner basis can be found for the system of equations that describe a VDF, we still face the problem of solving univariate equations for it.

Univariate equations over finite fields can be solved using Berlekamp or Cantor-Zassenhaus algorithms. It is known [vzGG13] that an equation of degree r can be solved using $O(r \log p)$ operations over elements of $\mathbb{F}_p$. This should be compared with $O(d \log p)$ operations needed to compute f^d. We conclude that, unless f^d has some structure that makes the univariate equations of its Groebner basis of very low degree (at most $\sqrt[3]{d}$), generic algebraic methods are not applicable even if f or $\widehat{f}$ has univariate representation.

4.3 Latency bounds

Currently the fastest exponentiation x^y algorithm that works with variable base is the windowing-ladder algorithm [CP05]. In the nutshell it splits the exponent y into windows of bitlength k, precomputes powers of x, and then applies the square-and-multiply method. For the 256-bit exponent the optimal k equals 5, which yields 256 squares and 66 multiplications. We are aware of no substantial improvement to this bound.

Going further down, the modulo squaring itself has been the subject of latency analysis in various models. For the gate depth model, the best available lower bounds [WW20] indicate that a squaring modulo ℓ-bit prime circuit has depth at least $2\log_2(\ell-1) - 2\log_2(\log_2(\ell-1)-1) - 4$, whereas the concrete upper bounds yield circuits of depth smaller than $5\log_2 \ell$ [MOS20]. We consider this gap to be small enough to prevent breakthroughs in circuit construction.

4.4 Quantum attacks

We also show that if we switch to a post-quantum proof system, the security of invertible VDF against a quantum computer is high enough with our parameters. The precomputation attacks utilize multi-target Grover's finds a pre-image to one of k targets in time $\sqrt{N/k}$. The probability to find the pre-image in t time grows as

$$f_{MG}(k, t, N) = kt^2/N \tag{5}$$

according to [BHMT02].

We consider precomputation based attacks described as follows:

1. Apply F^{-1} (i.e. invert F) to M/γ elements of $\mathcal{W}$ and create a proof for every inversion.
2. Store results as $\overline{\mathcal{W}} = \{(I, O, \pi)\}$.
3. When ϕ is known, check (using Grover's search) if there is I such that $(I, *, *) \in \overline{\mathcal{W}}$ and $\phi(I) = \mathit{true}$. If found then retrieve (O, π) from $\overline{\mathcal{W}}$.

We have to check each $\phi(I) = \mathit{true}$ individually so run Grover algorithm on the set $\overline{\mathcal{W}}$ of targets in the Computation phase, and in order to increase the attack probability we run M copies of Grover in

parallel. Let us assume that a call to ϕ costs as much as a call to f, so we can run Grover for time $D/2$. Then the chance to match any element of $\overline{W}$ can be derived from (5) as

$$P_{MG}(\beta, D, N) = \frac{M^2 D^2}{\gamma 2^{2-2l}} \tag{6}$$

Comparing to (3), and assuming that $D < 2^{40}$, we obtain that RootPack has λ bits of quantum security as long as

$$\ell - \log \ell > \lambda + 40.$$

Thus we get 70 bits of quantum security with 128-bit primes and 200 bits of security with 256-bit primes.

Practicality of Quantum Search

Grover's search is usually discussed for some oracle function $f(\cdot)$. However, several practical issues need to be considered when a specific construction of $f(\cdot)$ is being considered. The work of Viamontes, Markov and Hayes [VMH05] focuses on these issues. The first issue is that in order to query $f(\cdot)$ in superposition, we need a quantum hardware implementation of $f(\cdot)$. While a quantum circuit is usually similar in size to its classical counterpart, if the the circuit's maximum depth exceeds $\sqrt{N}$ then the evaluation of $f(\cdot)$ is the more cost consuming part of searching a size N database. This would diminish the quantum speedup by a considerable amount. Hence, to come up with resource estimates for quantum attacks we first come up with the quantum circuits for the VDF candidates. We omit the quantum circuits for brevity and keeping in mind the interests of the SBC audience. However, our estimates suggest that quantum computers which have the order of 1000 qubits would be required to make these attacks feasible. The state-of-art quantum machines have only reached 127-qubits [Bal21] at the time of writing.

4.5 MIMC reduction

Here we provide an observation that relates the sequentiality of invertible VDFs to the security of some well-known symmetric key primitives. For this we recall the MiMC bijective transformation over $\mathbb{F}_p^2$ [AGR+16]:

- Input $(x, y) \in \mathbb{F}_p^2$
- For $1 \leq i \leq R$ repeat:
    1. $(x, y) \leftarrow (y + x^3 + c_i, x)$ where c_i is a public constant that depends on i. In a keyed version, a key is also added mod$p$ with c_i.

The security of MiMC is based on the assumption that the resulting polynomial has large algebraic degree and it has no compact representation. Despite the years of public scrutiny [ACG+19, EGL+20], no counterargument to this assumption was found[1]. Computing Groebner basis has proven useless either. This hints that as long as the function f in the VDF design resembles that of MiMC, we should expect little to no algebraic structure in the polynomial describing the iteration of multiple rounds, which in turn implies that these polynomials are kind of "generic" and "random-looking". Thus intuitively there should be no sequentiality shortcut as it should not be for random functions.

We note that this argument holds as long as the polynomial describing the iteration has degree 3. It is known that certain quadratic polynomials yield short cycles and thus should not be used[VS04].

[1]There are certain cryptanalytic results for the binary field case and in the related key scenario, but they are irrelevant for our purpose.

5 Conclusion

We study and propose a new VDF protocol which satisfies the requirements for its use in Ethereum 2.0. We discuss the security and attacks possible on this protocol and provide estimates for an ASIC implementation for this VDF evalution. While we did not find any attacks that would suggest any crucial weaknesses in the protocol, we invite the VDF research community to scrutinize our work.

References

[ACG+19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of stark-friendly designs: Application to marvellous and mimc. In *ASIACRYPT (3)*, volume 11923 of *Lecture Notes in Computer Science*, pages 371–397. Springer, 2019.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[Bal21] Philip Ball. First quantum computer to pack 100 qubits enters crowded race, Nov 2021. https://www.nature.com/articles/d41586-021-03476-5.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT (1)*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. Technical report, Cryptology ePrint Archive, Report 2020/1536, 2020.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Asiacrypt*, 2021. https://eprint.iacr.org/2019/1177.pdf.

[BSCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ecfft) part i: Fast polynomial algorithms over all finite fields. *arXiv preprint arXiv:2107.08473*, 2021.

[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 Cryptol. ePrint Arch.*, 2020:374, 2020.

[CLO97] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793. Springer, 2020.

[CP05] Richard E. Crandall and Carl Pomerance. *Prime numbers: a computational perspective*. Springer, 2nd ed edition, 2005.

[DDK+21] Bernardo David, Justin Drake, Dmitry Khovratovich, Mary Maller, Hart Montgomery, Claudio Orlandi, Peter Scholl, Omer Shlomovits, and Riad Wahby. The red wedding: Playing attacker in mpc ceremonies. Real World Cryptography, 2021. https://iacr.org/submit/files/slides/2021/rwc/rwc2021/9/slides.pdf.

[DGMV20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, pages 65–84, 2020.

[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, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, pages 125–154, 2020.

[EGL+20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. An algebraic attack on ciphers with low-degree round functions: Application to full mimc. In *ASIACRYPT (1)*, volume 12491 of *Lecture Notes in Computer Science*, pages 477–506. Springer, 2020.

[Fau02] Jean Charles Faugere. A new efficient algorithm for computing gröbner bases without reduction to zero (f 5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, 2002.

[FMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *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*, pages 248–277, 2019.

[GMN21] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical SNARK aggregation. *IACR Cryptol. ePrint Arch.*, 2021:529, 2021.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[HBG20] Daira Hopwood, Sean Bowe, and Jack Grigg. The pasta curves for halo 2 and beyond, 2020. https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/.

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. *IACR Cryptol. ePrint Arch.*, page 370, 2021.

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to ppad-hardness and vdfs. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, pages 632–651, 2020.

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

[MOS20] Ahmet Can Mert, Erdinc Ozturk, and Erkay Savas. Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. Cryptology ePrint Archive, Report 2020/480, 2020. https://ia.cr/2020/480.

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In *ITCS*, volume 124 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[VMH05] George F. Viamontes, Igor L. Markov, and John P. Hayes. Is quantum search practical? *Comput. Sci. Eng.*, 7(3):62–70, 2005.

[VS04] Troy Vasiga and Jeffrey O. Shallit. On the iteration of certain quadratic maps over gf(p). *Discret. Math.*, 277(1-3):219–240, 2004.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407. Springer, 2019.

[WW20] Benjamin Wesolowski and Ryan Williams. Lower bounds for the depth of modular squaring. Cryptology ePrint Archive, Report 2020/1461, 2020. https://ia.cr/2020/1461.