Simple Verifiable Delay Functions

Krzysztof Pietrzak[*]
IST Austria
`pietrzak@ist.ac.at`

June 22, 2018

Abstract

We construct a verifable delay function (or unique publicly verifiable proof of sequential work) by showing how the Rivest-Shamir-Wagner time-lock puzzle can be made publicly verifiable.

Concretely, we give a statistically sound public-coin protocol to prove that a tuple (N, x, T, y) satisfies $y = x^{2^T} \pmod N$ where the prover doesn't know the factorization of N and its running time is dominated by solving the puzzle, that is, compute x^{2^T}, which is conjectured to require T sequential squarings.

The motivation for this work comes from the Chia blockchain design, which uses a VDF as a key ingredient. For typical parameters, our proofs are of size around $10KB$ and verification cost around three RSA exponentiations.

1 introduction

The RSW time-lock puzzle [RSW96] is defined as follows

The puzzle is a tuple (N, x, T) where $N = p \cdot q$ is an RSA modulus, $x \in \mathbb{Z}_N^*$ is a random generator and $T \in \mathbb{N}$ is a time parameter.

The solution to the puzzle is $y = x^{2^T} \mod N$. It can be computed making two exponentiations by the party who generates the puzzle (and thus knows the group order $\phi(N) = (p-1)(q-1)$) as

$$e = 2^T \mod \phi(N) \quad , \quad y = x^e \mod N \qquad (1)$$

but is conjectured to require T sequential squarings if the group order (or equivalently, the factorization of N) is not known

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to \ldots \to x^{2^T} \mod N \qquad (2)$$

To be more precise, the conjecture here is that T sequential steps are necessary to compute x^{2^T} (mod N) even if one can use large parallelism.

As an application, [RSW96] show how to "encrypt to the future": sample a puzzle (N, x, T) together with its solution y, then derive key k_y from y and encrypt a message m into a ciphertext $c = \mathsf{ENC}(k_y, m)$. Given $(N, x, T), c$ one can recover the message m in time required to compute T squarings sequentially, but (under the above conjecture) not faster.

Proofs of sequential work (PoSW) are objects closely related to time-lock puzzles. PoSW were introduced in [MMV13], and informally are proof systems where on input a random challenge x and time parameter T one can generate a *publicly verifable* proof making T sequential computations, but it's hard to come up with an accepting proof in less than T sequential steps.

The PoSW constructed in [MMV13] is not very practical (at least for large T) as the prover needs not only T time, but also linear in T space to compute a proof. Recently [CP18] constructed a very simple and practical PoSW in the random oracle model. They were interested in PoSW as they serve as a key ingredient in the Chia blockchain design (`chia.net`).

The main open problem left open in [CP18] was to construct PoSW that is *unique*, in the sense that one cannot compute two accepting proofs on the same challenge. The existing PoSW all allow to generate

---

[*] This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 682815/TOCNeT).

many accepting proofs at basically the same cost as honestly computing the proof. Unfortunately such PoSW cannot be used for blockchains as they would allow for so called grinding attacks.

Verifable delay functions (VDF) were recently introduced by Boneh, Bonneau, Bünz and Fisch [BBBF18]. A VDF can be seen as a relaxation of unique PoSW which still suffice for blockchain applications (see [BBBF18] for other applications). In a VDF the proof on challenge (x,T) has two parts (y,π), where y is a deterministic function of x that needs T sequential time to compute, and π is a proof that y was correctly computed. It must be possible to compute π with low parallelism and such that π can be output almost at the same time as y. In [BBBF18] this is achieved using incrementally verifiable computation [Val08]. The (very high level) idea is to compute a hash chain $y = \underbrace{h(h(\ldots h(x)\ldots))}_{T \text{ times}}$ and at the same time use incrementally verifiable computation to compute the proof π, so the proof will be ready shortly after y is computed. To make this generic approach actually practical [BBBF18] use very particular algebraic functions (permutation polynomials) for h and proof systems that can exploit that algebraic structure.

A VDF from RSW. The RSW time-lock puzzle looks like a promising starting point for constructing a VDF. The main difficulty one needs to solve is achieving public verifiability: to efficiently verify $y \overset{?}{=} x^{2^T} \pmod N$ one needs the group order of Z_N^* (or equivalently, the factorization of N). But the factorization cannot be public as otherwise also computing y becomes easy.

One idea to solve this issue is to somehow obfuscate the group order so it can only be used to efficiently verify if a given solution is correct, but not to speed up its computation. There currently is no known instantiation to this approach.

In this work we give a different solution. We construct a protocol where a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ it computed the correct solution $y = x^{2^T} \pmod N$ without either party knowing the factorization (or any other hard to compute function) of N. Our protocol is public-coin, and thus can be made non-interactive – and thus give a VDF – via the Fiat-Shamir transformation.

Our protocol is inspired by the IP=PSPACE proof [LFKN90, Sha90]. The key idea of the proof is very simple. Assume $\mathcal{P}$ wants to convince $\mathcal{V}$ that a tuple (x,y) satisfies $y = x^{2^T} \pmod N$. For this, $\mathcal{P}$ first sends $\mu = x^{2^{T/2}}$ to $\mathcal{V}$. Now $\mu = x^{2^{T/2}}$ together with $y = \mu^{2^{T/2}}$ imply $y = x^{2^T}$. The only thing we have achieved at this point is to reduce the time parameter from T to $T/2$ at the cost of having two instead just one statement to verify. We then show that the verifier can merge those two statements in a randomized way into a single statement $(x',y') = (x^r \cdot \mu, \mu^r \cdot y)$ that satisfies $y' = x'^{2^{T/2}}$ if the original statement $y = x^{2^T}$ was true, but is almost certainly wrong (over the choice of the random exponent r) if the original statement was wrong, no matter what μ the malicious prover did send. This subprotocol is repeated $\log(T)$ times – each time halving the time parameter – until $\mathcal{V}$ can trivially verify correctness of the claim.

The VDF we get has short proofs and is efficiently verifiable. For typical parameters (2048 bit modulus and $T \le 40$) a proof is about $10KB$ large and the cost for verification is around three full exponentiations (for comparison, a standard RSA decryption or RSA signature computation requires one full exponentiation).

Outline. We present the protocol in §2 and the security proof in §3. In §4 we define VDFs, and in §5 we discuss how the protocol is turned into a VDF and discuss several efficiency and security issues.

Notation. For a set $\mathcal{X}$, $x \overset{\$}{\leftarrow} \mathcal{X}$ means x is assigned a random value from $\mathcal{X}$. For a randomized algorithm alg we denote with $x \overset{\$}{\leftarrow} \mathsf{alg}$ that x is assigned the output of alg on fresh random coins, if alg is deterministic we just write $x \leftarrow \mathsf{alg}$.

2 the protocol

Our protocol, where $\mathcal{P}$ convinces $\mathcal{V}$ it solved an RSW puzzle, goes as follows:

- The verifier $\mathcal{V}$ and prover $\mathcal{P}$ have common input an RSW puzzle (N,x,T) and a statistical security parameter λ.

- $\mathcal{P}$ solves the puzzle by computing $y = x^{2^T} \bmod N$ (making T sequential squarings), and sends y to $\mathcal{V}$.
- Now $\mathcal{P}$ and $\mathcal{V}$ iterate the "halving protocol" below. In this subprotocol, on common input (N,x,T,y) the output is either of the form $(N,x',\lceil T/2\rceil,y')$, in which case it is used as input to the next iteration of the halving subprotocol, or the protocol has stopped with verifier output in $\{\mathsf{reject},\mathsf{accept}\}$.

2.1 the halving subprotocol

On common input (N,x,T,y)

1. If $T=2$ then $\mathcal{V}$ outputs accept if $y = x^{2^T} = x^4 \pmod N$ and reject otherwise. If $T>2$ go to the next step.
2. The prover $\mathcal{P}$ sends $\mu' = x^{2^{T/2-1}} \pmod N$ to $\mathcal{V}$.
3. If $\mu' \notin \mathbb{Z}_N^*$ $\mathcal{V}$ outputs reject and stops, otherwise $\mathcal{V}$ computes $\mu := \mu'^2 \bmod N$ (note that $\mu \in QR_N$).
4. $\mathcal{V}$ samples a random $r \xleftarrow{\$} \mathbb{Z}_{2^\lambda}$ and sends it to $\mathcal{P}$ (in the non-interactive version of the protocol r is the hash of the prover's message μ').
5. If $T/2$ is even, $\mathcal{P}$ and $\mathcal{V}$ output

   $$(N,x',T/2,y')$$

   where

   $$\begin{aligned} x' &:= x^r \cdot \mu \mod N & \left(= x^{r+2^{T/2}}\right) \\ y' &:= \mu^r \cdot y \mod N & \left(= x^{r\cdot 2^{T/2}+2^T}\right) \end{aligned}$$

   (note that if $y = x^{2^T}$ then $y' = x'^{2^{T/2}} = \pmod N$)). If $T/2$ is odd, output

   $$(N,x',T/2+1,y'^2) .$$

2.2 security statement

Theorem 1. *If the input (N,x,T) to the protocol satisfies*

1. *$N = pq$ is the product of safe primes, i.e., $p = 2p'+1, q = 2q'+1$ for primes p',q'.*
2. *$\langle x\rangle = QR_N$.*[1]

Then for any malicious prover $\widetilde{\mathcal{P}}$ who sends as first message y anything else than the solution to the RSW time-lock puzzle, i.e.,

$$y \neq x^{2^T} \bmod N$$

$\mathcal{V}$ will finally output accept *with probability at most*

$$\frac{3\log(T)}{2^\lambda} .$$

3 security proof

It will be convenient to define the language

$$\mathcal{L} = \{(N,x,T,y) \quad : \quad y \neq x^{2^T} \bmod N \text{ and } \langle x\rangle = QR_N\}$$

We'll establish the following lemma.

Lemma 1. *For N as in Thm. 1, and any malicious prover $\widetilde{\mathcal{P}}$ the following holds. If the input to the halving protocol in §2.1 satisfies*

$$(N,T,x,y) \in \mathcal{L}$$

then with probability $\geq 1 - 3/2^\lambda$ $\mathcal{V}$'s outputs is either reject *or satisfies*

$$(N,\lceil T/2\rceil,x',y') \in \mathcal{L}$$

Before we prove the lemma, let's see how it implies Theorem 1.

Proof of Theorem 1. In every iteration of the halving protocol the time parameter decreases from T to $\lceil T/2\rceil$ and it stops once $T=2$, this means we iterate for at most $\lceil\log(T)\rceil - 2$ rounds. By assumption, the input (N,x,T,y) to the first iteration is in $\mathcal{L}$, and by construction, the only case where $\mathcal{V}$ outputs accept is on an input $(N,x,2,y)$ where $y = x^{2^T} = x^4 \bmod N$, in particular, this input is *not* in $\mathcal{L}$.

So, if $\mathcal{V}$ outputs accept, there must be one iteration of the halving protocol where the input is in $\mathcal{L}$ but the output is not. By Lemma 1, for any particular iteration this happens with probability $\leq 3/2^\lambda$. By the union bound, the probability of this happening in any of the $\lceil\log(T)\rceil - 2$ rounds can be upper bounded by $3\log(T)/2^\lambda$ as claimed. □

---

[1] That is, x generates QR_N, the quadratic residues modulo N. For our choice of N we have $|QR_N| = p'q'$, so

$$\langle x\rangle \overset{def}{=} \{x,x^2,\ldots,x^{p'q'} \bmod N\} = QR_N \overset{def}{=} \{z^2 \bmod N \ : \ z \in \mathbb{Z}_N^*\} .$$

.

Proof of Lemma 1. We just consider the case where T is even, the T odd case is almost identical.

Assuming the input to the halving protocol satisfies $(N,x,T,y) \in \mathcal{L}$, we must bound the probability that $\mathcal{V}$ outputs reject or the output $(N,x',T/2,y') \notin \mathcal{L}$.

If $T = 2$ then $\mathcal{V}$ outputs reject and we're done. Otherwise, if $\widetilde{\mathcal{P}}$ sends a $\mu' \notin \mathbb{Z}_N^*$ in step 3. then $\mathcal{V}$ outputs reject and we're done. So from now we assume $\mu' \in \mathbb{Z}_N^*$, and thus $\mu = \mu'^2 \in QR_N$. We must bound

$$\Pr_r[(y' = x'^{2^{T/2}} \bmod N) \vee (\langle x' \rangle \neq QR_N)] \leq 3/2^\lambda$$

by the union bound we can upper bound the two events separately, i.e.,

$$\Pr_r[y' = x'^{2^{T/2}} \bmod N] + \Pr_r[\langle x' \rangle \neq QR_N] \leq 3/2^\lambda \quad (3)$$

Eq.(3) follows by the two claims below.

Claim 1. $\Pr_r[\langle x' \rangle \neq QR_N] \leq 2/2^\lambda$.

Proof of Claim. We'll denote with e_μ the unique values in $\mathbb{Z}_{p'q'}$ satisfying $x^{e_\mu} = \mu$ (it's unique as $\mu \in \langle x \rangle = QR_N$ and $|QR_N| = p'q'$). As $x,\mu \in QR_N$, also $x' = x^r \cdot \mu = x^{r+e_\mu}$ is in QR_N, and $\langle x' \rangle = QR_N$ holds if $ord(x') = p'q'$, which is the case except if

$$(r + e_\mu) = 0 \bmod p' \quad \text{or} \quad (r + e_\mu) = 0 \bmod q'$$

either which happens for at most one choice of $r \in \mathbb{Z}_{2^\lambda}$ (as $2^\lambda < \min(p',q')$). The claim follows. □

Claim 2. $\Pr_r[y' = x'^{2^{T/2}} \bmod N] \leq 1/2^\lambda$.

Proof of Claim. If $y \notin QR_N$, then also $y' = \mu^r \cdot y \notin QR_N = \langle x' \rangle$ (as $a \in QR_N, b \notin QR_N$ implies $a \cdot b \notin QR_N$). In this case $y' \notin \langle x' \rangle$ so the probability in the claim is 0.

From now on we assume $y \in QR_N = \langle x \rangle$ and let $e_y \in \mathbb{Z}_{p'q'}$ be the unique value such that $x^{e_y} = y \bmod N$. Using $\langle x \rangle = QR_N$ in the last step below we can rewirte

$$\begin{aligned} y' = x'^{2^{T/2}} \bmod N &\iff \\ \mu^r y = (x^r \mu)^{2^{T/2}} \bmod N &\iff \\ x^{r \cdot e_\mu + e_y} = x^{(r+e_\mu) \cdot 2^{T/2}} \bmod N &\iff \\ r \cdot e_\mu + e_y = (r + e_\mu) \cdot 2^{T/2} \bmod p'q' & \end{aligned}$$

rearranging terms

$$r(e_\mu - 2^{T/2}) + e_y - e_\mu 2^{T/2} = 0 \bmod p'q' \ . \quad (4)$$

If $e_\mu = 2^{T/2}$ this becomes

$$e_y - 2^T = 0 \bmod p'q'$$

which does not hold as by assumption we have $y \neq x^{2^T} \bmod N$. So from now on we assume $e_\mu \neq 2^{T/2} \bmod p'q'$. Then for $a = e_\mu - 2^{T/2} \neq 0 \bmod p'q'$ (and $b = e_y - e_\mu 2^{T/2}$) eq.(4) becomes

$$r \cdot a = b \bmod p'q'$$

which holds for at most one choice of r and the claim follows. □

□

4 verifiable delay functions

In this section we define verifiable delay functions (VDF), we mostly follow the definition from [BBBF18]. A VDF is defined by a four-tuple of algorithms:

PoSW.Setup$(1^\lambda) \to$ **pp** on input a statistical security parameter 1^λ outputs public parameters **pp**.

PoSW.Gen$(\mathbf{pp}, T) \to (x,T)$ on input a time parameter $T \in \mathbb{N}$, samples an input x.

PoSW.Sol$(\mathbf{pp},(x,T)) \to (y,\pi)$ on input (x,T) outputs (y,π), where π is a proof that the output y has been correctly computed.

PoSW.Ver$(\mathbf{pp},(x,T),(y,\pi)) \to$ {accept/reject} given an input/output tuple $(x,T),(y,\pi)$ outputs either accept or reject.

The statistical security parameter λ measures the bit-security we expect from our protocol, i.e., an adversary running in total time τ should have advantage no more than $\tau/2^\lambda$ in breaking the scheme. It only makes sense to consider time parameters T that are much smaller than 2^λ (say we require $T \leq 2^{\lambda/2}$) so the sequential running time of the honest prover is much smaller than what is required to break the underlying hardness assumptions.

Efficiency of setup, sampling and verification. The PoSW.Setup and PoSW.Gen algorithms are probabilistic, PoSW.Ver is deterministic. They all run in time $\mathsf{poly}(\log(T),\lambda)$.

Efficiency of solving. The PoSW.Sol algorithm can compute the output y in T sequential steps (in this work a "sequential step" is a squaring or multiplication in $\mathbb{Z}_N^*$). In [BBBF18] it is required that the proof π is computable with bounded $\mathsf{poly}(\log(T),\lambda)$ parallelism so it's available shortly after y is computed. We will achieve a stronger guarantee, where π can be computed in $o(T)$ sequential steps (but if parallelism is available, the time required can be further reduced from $O(\sqrt{T})$ to $O(\log(T))$).

Completeness. The completeness property simply requires that correctly generated proofs will always accept, that is, for any λ, T

$$\Pr\left[\begin{array}{l}\mathsf{PoSW.Ver}(\mathbf{pp},(x,T),(y,\pi)) = \mathsf{accept}\\ \text{where}\\ \mathbf{pp} \xleftarrow{\$} \mathsf{PoSW.Setup}(1^\lambda)\\ (x,T) \xleftarrow{\$} \mathsf{PoSW.Gen}(\mathbf{pp},T)\\ (y,\pi) \leftarrow \mathsf{PoSW.Sol}(\mathbf{pp},(x,T))\end{array}\right] = 1$$

Security (sequentiality). The first security property is sequentiality. For this we consider a two part adversary $\mathscr{A} = (\mathscr{A}_1,\mathscr{A}_2)$, where $\mathscr{A}_1$ can run a pre-computation and choose T. Then $\mathscr{A}_2$ gets a random challenge for time T together with the output state of the precomputation, we require that whenever

$$\Pr\left[\begin{array}{l}\mathsf{PoSW.Ver}(\mathbf{pp},(x,T),(\tilde{y},\widetilde{\Pi})) = \mathsf{accept}\\ \text{where}\\ \mathbf{pp} \xleftarrow{\$} \mathsf{PoSW.Setup}(1^\lambda)\\ (T,\mathsf{state}) \xleftarrow{\$} \mathscr{A}_1(\mathbf{pp},(x,T))\\ (x,T) \xleftarrow{\$} \mathsf{PoSW.Gen}(\mathbf{pp},T)\\ (\tilde{y},\widetilde{\Pi}) \xleftarrow{\$} \mathscr{A}_2(\mathbf{pp},(x,T),\mathsf{state})\end{array}\right] \neq \mathsf{negl}(\lambda)$$

the $\mathscr{A}_2$ adversary must use almost the same *sequential* time T as required by an honest execution of $\mathsf{PoSW.Sol}(\mathbf{pp},(\pi,T))$, and this even holds if $\mathscr{A}$ is allowed massive parallel computation (say we just bound the total computation to $2^{\lambda/2}$). This means there's no possible speedup to compute the VDF output by using parallelism. Let us stress that by this we mean any parallelism that goes beyond what can be used to speed up a single sequential step, which here is a multiplication in $\mathbb{Z}_N^*$, and we assume the honest prover can use such bounded parallelism.

Security (uniqueness). The second security property is uniqueness, which means that one cannot come up with an accepting proof where the y part was wrongly computed. Formally, for an adversary $\mathscr{A} = (\mathscr{A}_1,\mathscr{A}_2)$ we have (unlike in the previous definition, here we don't make any assumption about $\mathscr{A}_2$'s sequential running time, just the total running time of $\mathscr{A}$ must be bounded to, say $2^{\lambda/2}$)

$$\Pr\left[\begin{array}{l}\mathsf{PoSW.Ver}(\mathbf{pp},(x,T),(\tilde{y},\widetilde{\Pi})) = \mathsf{accept}\\ \text{and } \tilde{y} \neq y\\ \text{where}\\ \mathbf{pp} \xleftarrow{\$} \mathsf{PoSW.Setup}(1^\lambda)\\ (T,\mathsf{state}) \xleftarrow{\$} \mathscr{A}_1(\mathbf{pp},(x,T))\\ (x,T) \xleftarrow{\$} \mathsf{PoSW.Gen}(\mathbf{pp},T)\\ (y,\pi) \leftarrow \mathsf{PoSW.Sol}(\mathbf{pp},(x,T))\\ (\tilde{y},\widetilde{\Pi}) \xleftarrow{\$} \mathscr{A}_2(\mathbf{pp},(x,T),\mathsf{state})\end{array}\right] = \mathsf{negl}(\lambda)$$

5 a VDF from RSW

In this section we explain the simple transformation of the protocol from §2 into a VDF and then discuss the efficiency, security and some other issues of this construction.

To keep things simple we'll assume that the time parameter $T = 2^t$ is a power of two. The four algorithms from §4 are instantiated as

$\mathsf{PoSW.Setup}(1^\lambda)$ The statistical security parameter λ defines another security parameter λ_{RSA} specifying the bitlength of an RSA modulus, where λ_{RSA} should be at least as large so that an RSA modulus λ_{RSA} offers λ bits of security (e.g. $\lambda = 100$ and $\lambda_{\mathsf{RSA}} = 2048$). In the security proof we'll use that $\lambda \geq \lambda_{\mathsf{RSA}}/2$, which holds for any reasonable choice of λ_{RSA}.

The setup algorithm samples two random $\lambda_{\mathsf{RSA}}/2$ bit safe primes p, q and output as public parameters the single λ_{RSA} bit RSA modulus $N := p \cdot q$.

$\mathsf{PoSW.Gen}(N,T)$ samples a random $x \in QR_N$ and outputs (x,T).

PoSW.Sol$(N,(x,T))$ outputs (y,π) where $y = x^{2^T} \bmod N$ is the solution of the RSW time-lock puzzle and $\pi = \{\mu'_i\}_{i\in[t-2]}$ is a proof that y has been correctly computed. It is derived by applying the Fiat-Shamir heuristic to the protocol in §2. Recall that in this heuristic the public-coin challenges $r_i \in \mathbb{Z}_{2^\lambda}$ of the verifier are replaced with a hash of the last prover message. Concretely, we fix a hash function $hash : \mathbb{Z}_N^2 \to \mathbb{Z}_{2^\lambda}$,[2] let $(x_1,T_1,y_1) = (x,T,y)$ and for $i = 1\ldots t-3$ let

$$\begin{aligned}
T_{i+1} &:= T_i/2 \quad (= T/2^i)\\
\mu'_i &:= x_i^{T_i/2-1} \bmod N\\
\mu_i &:= {\mu'_i}^2 = x_i^{T_i/2} \bmod N\\
r_i &:= hash(x,\mu'_i)\\
x_{i+1} &:= x_i^{r_i}\cdot\mu_i \bmod N\\
y_{i+1} &:= {\mu'_i}^{r_i}\cdot y_i \bmod N
\end{aligned}$$

PoSW.Ver$(N,(x,T),(y,\pi))$ parse $\pi = \{\mu'_i\}_{i\in[t-2]}$ and check if any $\mu'_i \not\in \mathbb{Z}_N^*$, if this is the case output reject. Otherwise set $(x_1,T_1,y_1) = (x,T,y')$ and then for $i = 1\ldots t-3$ compute

$$\begin{aligned}
T_{i+1} &:= T_i/2 \quad (= T/2^i)\\
\mu_i &:= {\mu'_i}^2 \bmod N\\
r_i &:= hash(x,\mu'_i)\\
x_{i+1} &:= x_i^{r_i}\cdot\mu_i \bmod N \qquad (5)\\
y_{i+1} &:= {\mu'_i}^{r_i}\cdot y_i \bmod N \qquad (6)
\end{aligned}$$

Finally check whether

$$y_{t-2} \overset{?}{=} x_{t-2}^4 \bmod N \tag{7}$$

and output accept if this holds, otherwise output reject.

5.1 public parameters for the VDF

For the security of the VDF it's crucial that a prover does not know the factorization of the public parameter N, as otherwise he could compute $x^{2^T} \pmod N$ in just two exponentiations as in eq.(1). Thus one either has to rely on a trusted party, or use multiparty-computation to sample N. In particular, it's possible to sample N securely as long as not all the participants in the multiparty computation are malicious. Such an "MPC ceremony" has been done before, e.g. to set up the common random string for Zcash.[3] [BBBF18] suggest two ways to avoid a common setup for the VDF presented here, the first is choosing a very large random N using public coins, so that N has two large prime factors with high probability. The second suggestion is to use the class group of an imaginary quadratic order [BW88].

Let us mention that the random-oracle based PoSW [MMV13, CP18] don't require a setup procedure at all.

5.2 efficiency of the VDF

Cost of verification. The cost of running the verification PoSW.Ver$(N,(x,T=2^t),(y,\pi))$ is dominated by the $2(t-3)$ exponentiations (with λ bit long exponents) in eq.(5,6). As exponentiation with a random λ bit exponent cost about 1.5λ multiplications,[4] the cost of verification is around $3\cdot\lambda\cdot(t-3)$ multiplications in $\mathbb{Z}_N^*$. For concreteness, consider an implementation where $\lambda = 100, \lambda_{\mathsf{RSA}} = 2048$ and assume $t = 40$, this gives a cost of about $3\cdot\lambda\cdot(t-3) = 11100$ multiplications, which corresponds to $11100/(2048\cdot 1.5) \approx 3.6$ full exponentiations in $\mathbb{Z}_N^*$.

A minor improvement. There's a simple way to save on verification time and proof size. Currently, for $\Delta = 2$ we run the halving protocol for $t-\Delta-1 = t-3$ rounds and then in eq.(7) check if $y_{t-\Delta} = x_{t-\Delta}^{2^\Delta}$. We can run the protocol for fewer rounds, say set $\Delta = 8$, which saves 6 rounds and thus reduces proof size from 37 to 31 elements (or about 16%). We also save $2\cdot 6$ exponentiations as in eq.(5,6), but as the final check eq.(7) for general Δ becomes $y_{t-\Delta} \overset{?}{=} x_{t-2}^{2^\Delta} \bmod N$ this gets more expensive as Δ grows. With $\Delta = 8$ the overall computation still drops by about 14% for a total cost of slightly above 3 full exponentiations.

---

[2] Instead of just hashing the previous prover message μ'_i, we'll also hash the instance x. This is not important for the security proof (showing that the Fiat-Shamir heuristic is sound in the random oracle model), but it's a cheap way of deterring precomputation attacks, which in practice means we can use a smaller λ and thus get better efficiency.

[3] `https://z.cash/technology/paramgen.html`

[4] Exponentiation is typically done via "square and multiply", which for a z bit exponent with hamming weight $h(z)$ requires $z + h(z)$ multiplications, or about $1.5\cdot z$ multiplication for a random exponent (where $h(z) \approx z/2$).

Cost of computing the proof. Computing the proof $(y,\pi) \leftarrow \mathsf{PoSW.Sol}(N,(x,T))$ requires one to solve the underlying RSW puzzle $y = x^{2^T} \pmod N$, which is done by squaring x sequentially T times (the security of the RSW puzzle and thus also our VDF relies on the assumption that there's no shortcut to this computation).

On top of that, for the VDF we also must compute the proof $\pi = \{\mu'_i\}_{i\in[t-2]}$. Recall that $\mu'_i = x_i^{T_i/2-1}$ and $\mu_i = {\mu'_i}^2 = x_i^{T_i/2}$. Below we discuss how to compute the μ_i instead of the μ'_i, this doesn't really affect the argument but the "-1" in the exponent of the μ'_i's makes the statements a bit more messy, we also assume $T = 2^t$ is a power of 2.

If naïvely implemented, computing the μ_i will require $T/2$ squarings for μ_1, $T/4$ for μ_2 etc., for a total of $T \approx T/2 + T/4 + T/8 \ldots + 8$. Fortunately we don't have to compute $\mu_1 = x^{2^{T/2}}$ as we already did so while computing $y = x^{2^T}$ by repeated squaring (cf. eq.2). This observation already saves us half the overhead. We can also compute the remaining $\mu_2, \mu_3, \ldots$ using stored values, but it becomes increasingly costly. As we discuss below, in practice one could compute, say $\mu_1, \ldots, \mu_{10}$ using stored values, and then fully recompute the remaining $\mu_{11} = x_{11}^{T/2^{11}}, \mu_{12}, \ldots$ which will only require $T/2^{10}$ squarings.

To see how the μ_i's can be efficiently computed for small i, let $z \in QR_N$ let $\overline{z}$ denote z's log to basis x, i.e., $x^{\overline{z}} = z \bmod N$. We have $\overline{x}_1 = 1, \overline{y}_1 = 2^T$ and

$$\begin{aligned} \overline{\mu}_i &:= \overline{x}_i \cdot 2^{T/2^i} \\ \overline{x}_{i+1} &:= r_i \cdot \overline{x}_i + \overline{\mu}_i \\ \overline{y}_{i+1} &:= r_i \cdot \overline{\mu}_i + \overline{y}_i \end{aligned}$$

How those exponents concretely develop for $i = 1$ to 3 is illustrated in Figure 1. For example, we can compute μ_3 assuming we stored the $x^{T/8}, x^{T3/8}, x^{T5/8}, x^{T7/8}$ values as

$$\mu_3 = (x^{2^{T/8}})^{r_1 \cdot r_2} + (x^{2^{T5/8}})^{r_2} + (x^{2^{T3/8}})^{r_1} + x^{2^{T7/8}}$$

In general, computing μ_i will require to store 2^{i-1} values, and then compute 2^{i-1} exponentiations with exponents of bitlength $\lambda \cdot (i-1)$ (let us mention that we can't speed this up by first taking the exponents module the group order $p'q'$ as it's not know).

Asymptotically, computing the first half $\mu_1, \ldots, \mu_{t/2}$ using stored values, and recompute $\mu_{t/2+1}, \ldots, \mu_{t-2}$ from scratch, will requrie $2^{t/2} \cdot \lambda \cdot t/2 + 2^{t/2} \approx \sqrt{T} \cdot \log(T) \cdot \lambda$ multiplications and we need to store $\sqrt{T}$ values in $\mathbb{Z}_N^*$ during the computation of y. In general, the optimal strategy will depend on the parallelism and storage available to the prover.

5.3 security of the VDF

Uniqueness. If we model *hash* as a random oracle, then by Theorem 1 and the Fiat-Shamir heuristic we are guaranteed that a prover will not find an accepting proof $(\tilde{y}, \tilde{\pi})$ where $\tilde{y} \neq x^{2^T}$.

Concretely, the probability that a prover that makes up to q queries to *hash* finds such an accepting proof where $y \neq x^{2^T}$ is at most $3 \cdot q/2^\lambda$.

As outlined in Footnote 2, by using the challenge instance x as an additional input to the random oracle, we get soundness $3 \cdot q/2^\lambda$ even against an adversary who makes q queries *after* receiving the challenge x, but can make a huge number of oracle queries before that.

Sequentiality. To break sequentiality means computing y faster than in T sequential computations. We rely on the same assumption as [RSW96], which simply states that such a shortcut does not exist (the only difference to [RSW96] is that our N is the product of safe primes, not a general RSA modulus).

References

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO 2018*, 2018.

[BW88] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, 1988.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of*

i	$\overline{x}'_i$	$\overline{\mu}_i$	$\overline{y}_i$
1	1	$2^{T/2}$	2^{T}
2	$r_1 + 2^{T/2}$	$r_1 \cdot 2^{T/4} + 2^{3T/4}$	$r_1 \cdot 2^{T/2} + 2^{T}$
3	$r_1 \cdot r_2 + r_2 \cdot 2^{T/2} + 2^{T/4} \cdot r_1 + 2^{3T/4}$	$r_1 \cdot r_2 \cdot 2^{T/8} + r_2 \cdot 2^{T5/8} + r_1 \cdot 2^{T3/8} + 2^{T7/8}$	$r_1 \cdot r_2 \cdot 2^{T/4} + r_2 \cdot 2^{3T/4} + r_1 \cdot 2^{T/2} + 2^{T}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 1: Exponents of the the values in the protocol, here $z = x^{\overline{z}}$.

Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II, pages 451–467, 2018.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990.

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

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

[Sha90] Adi Shamir. IP=PSPACE. In *31st FOCS*, pages 11–15. IEEE Computer Society Press, October 1990.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.