Verifiable Delay Functions

An introduction

Mitul Patel

Instructor: Dr Somitra Sanadhya

Indian Institute of Technology Jodhpur

May 19, 2021

Summary

The Idea behind Pietrzak VDF Scheme

Setup(λ, T)

- Let's take
  1. A Finite abelian multiplicative group of unknown order: G, with mod n.
  2. The delay parameter some T
  3. The info about input domain $\mathcal{X}$ and the output domain $\mathcal{Y}$.
  4. Some security parameter λ.
- Now the public parameter will be

$$pp \leftarrow (n, T, \mathcal{X}, \mathcal{Y}, \lambda)$$

$\text{Eval}(pp, x)$

- $\text{Eval}(pp, x) \rightarrow x^{2^T} \mod n$
- The representation of $2^T \xrightarrow{binary} 100000...00$(1 followed by T zeroes)
- Using the Square and Multiply algorithm, we can compute the $x^{2^T} \mod n$ in T sequential steps.
- The actual delay will be approximiately some multiple of T.

Verify(pp, x, y, π)

Protocol 1: Verify(n, x, T, y)

if $T == 1$ **then**
    if $y == x^2$ **then** Verifier **Accepts** ;
    else Verifier **Rejects** ;
else
    Prover Sends $\mu \leftarrow x^{2^{T/2}}$ to the Verifier.
    if $\mu \notin G$ **then**
        Verifier **Rejects** ;
    else
        Verifier samples $r \xleftarrow{\$} \mathbb{Z}_{2^\lambda}$ and sends it to the Prover
        The Prover & Verifier computes: $x' \leftarrow x^r\mu$
        The Prover & Verifier computes: $y' \leftarrow \mu^r y$
        if $T/2$ *is even* **then**
            The Prover & Verifier engages in: Verify$(n, x', T/2, y')$
        else
            The Prover & Verifier engages in: Verify$(n, x', \lceil T/2 \rceil, y'^2)$
        end
    end
end

Analysis

Analysis

- In each step the verifier basically asks for the proof of $T/2$ amount of work.
- $T/2 \rightarrow T/4 \rightarrow T/8 \rightarrow T/16 \rightarrow T/32 \rightarrow \cdots$

Analysis

- In each step the verifier basically asks for the proof of $T/2$ amount of work.
- $T/2 \to T/4 \to T/8 \to T/16 \to T/32 \to \cdots$
- Which sums up as:

$$
\begin{gathered}
T/2 + T/4 = 3T/4 \\
3T/4 + T/8 = 7T/8 \\
7T/8 + T/16 = 15T/16 \\
15T/16 + T/32 = 31T/33 \\
\vdots
\end{gathered}
$$

Analysis

- At the end when recursively $T \xrightarrow{reaches} 1$
- The verifier simply checks if, $y == x^{2^T} (\equiv y == x^2)$

Analysis

- At the end when recursively $T \xrightarrow{reaches} 1$
- The verifier simply checks if, $y == x^{2^T} (\equiv y == x^2)$
- The terms μ, x', y' grow as shown below: for i^{th} recursive-iteration,

$$\mu_i \leftarrow x^{(\prod_{k=0}^{i-1} f(k)) \cdot 2^{T_i/2}}$$

$$x'_i \leftarrow x^{(\prod_{k=0}^{i} f(k))}$$

$$y'_i \leftarrow x^{(\prod_{k=0}^{i} f(k)) \cdot 2^{T_i/2}}$$

Where: $T_i = T/2^{i-1} \mid f(k) = r_k + 2^{T_{k+1}} \;,\; f(0) = 1$

Analysis

- At the end when recursively $T \xrightarrow{reaches} 1$
- The verifier simply checks if, $y == x^{2^T} (\equiv y == x^2)$
- The terms μ, x', y' grow as shown below: for i^{th} recursive-iteration,

$$\mu_i \leftarrow x^{(\prod_{k=0}^{i-1} f(k)) \cdot 2^{T_i/2}}$$

$$x'_i \leftarrow x^{(\prod_{k=0}^{i} f(k))}$$

$$y'_i \leftarrow x^{(\prod_{k=0}^{i} f(k)) \cdot 2^{T_i/2}}$$

Where: $T_i = T/2^{i-1} \mid f(k) = r_k + 2^{T_{k+1}} \;,\; f(0) = 1$

- At each level the relation $y' = x'^{2^{T_i/2}}$ is maintained & T_i is halved.
- If we consider $r = 0$ at each level, $x' \xrightarrow{sums-to} x^{2^T}$

Analysis - Proof Construction

- Non-interactive version using Fiat-Shamir heuristic

$$\pi \leftarrow \{\mu_1, \mu_2, \mu_3, \cdots, \mu_d\} \;\middle|\; d = \log_2(T)$$

Analysis - Proof Construction

- Non-interactive version using Fiat-Shamir heuristic

$$\pi \leftarrow \{\mu_1, \mu_2, \mu_3, \cdots, \mu_d\} \mid d = \log_2(T)$$

- If we observe,

$$\mu_1 \leftarrow x^{2^{T/2}}$$

$$\mu_2 \leftarrow x^{r_1 2^{T/4} + 2^{3T/4}}$$

$$\mu_3 \leftarrow x^{r_1 \cdot r_2 \cdot 2^{T/8} + r_1 \cdot 2^{3T/8} + r_2 \cdot 2^{5T/8} + 2^{7T/8}}$$

$$\vdots$$

Analysis - Proof Construction

- So we can compute the μ_i values from already computed values from x^{2^T}.

Analysis - Proof Construction

- So we can compute the μ_i values from already computed values from x^{2^T}.
- But it would require us to store 2^d values, where $d = \log_2(T)$

Analysis - Proof Construction

- So we can compute the μ_i values from already computed values from x^{2^T}.
- But it would require us to store 2^d values, where $d = \log_2(T)$
- So, we make a tradeoff between storage & compute by having a $s \in [1 \ldots d]$, such that we only use storage upto s^{th} recursive level & recompute the rest values, using

$$\frac{T}{2^{s+1}} + \frac{T}{2^{s+2}} + \cdots + \frac{T}{2^d} < \frac{T}{2^s}$$

- with 2^s values stored & rest to be computed with $\frac{T}{2^s}$ multiplications, we need a total of $2^s + \frac{T}{2^s}$ operations, now to minimise that we need $s = \log_2(\sqrt{T})$. So the proof generation time is of the order:

$$O(\sqrt{T})$$

Analysis - Verification

- Verification basically requires the computation of:

$$x' \leftarrow x^r \mu$$

$$y' \leftarrow \mu^r y$$

  in $d = \log_2(T)$ levels.
- So the total time is dominated by the 2 exponentiations with r, in $\log_2(T)$ levels $\Rightarrow 2 \cdot \log_2(T)$ exponentiations
- So the verification time is: $O(\log_2(T)) \Rightarrow \text{polylog}(T)$.

Comparison with Wesolowski VDF Scheme

Comparison - Proof Size

- The Pietrzak scheme has proof size of $\log_2(T)$ elements:

$$\pi \leftarrow \{\mu_1, \mu_2, \mu_3, \cdots, \mu_d\} \mid d = \log_2(T)$$

- While the Wesolowski scheme has proof size of 1 element:

$$\pi \leftarrow x^b$$

Comparison - Verification Speed

- We saw that The proof generation for Piertzak scheme takes : $2\log_2(T)$ exponentiations
- While the wesolowski takes only 2 exponentiations:

$$y == \pi^L x^r$$

Comparison - Proof Generation Speed

- We saw that proof generation in Pietrzak scheme takes $O(\sqrt{T})$ time.
- While in Wesolowski it takes $O(T)$ time.
- Both the approaches are parallelizable.

References

A VDF Explainer. https://reading.supply/@whyrusleeping/a-vdf-explainer-5S6Ect.

Boneh, D., Bonneau, J., Bünz, B. & Fisch, B. Verifiable Delay Functions. *IACR Cryptol. ePrint Arch.* **2018,** 601. https://eprint.iacr.org/2018/601 (2018).

Boneh, D., Bünz, B. & Fisch, B. A Survey of Two Verifiable Delay Functions. *IACR Cryptol. ePrint Arch.* **2018,** 712. https://eprint.iacr.org/2018/712 (2018).

Introduction to Verifiable Delay Functions (VDFs). https://blog.trailofbits.com/2018/10/12/introduction-to-verifiable-delay-functions-vdfs/. Oct. 2018.

Netti, J. *Pietrzak Verifiable Delay Functions.* https://medium.com/@joenetti/pietrzak-verifiable-delay-functions-f5683131882b. May 2020.

Rocha, A. d. l. *@adlrocha - A gentle introduction to VDFs.* https://adlrocha.substack.com/p/adlrocha-a-gentle-introduction-to. June 2020.

Wesolowski, B. Efficient Verifiable Delay Functions. *J. Cryptol.* **33,** 2113–2147. https://doi.org/10.1007/s00145-020-09364-x (2020).

Thank You!