On Verifiable Delay Functions from Time-Lock Puzzles

Hamza Abusalah Karen Azari Dario Fiore
Chethan Kamath Erkan Tairi

PKC 2026

(Trapdoor) Verifiable Delay Functions [BBBF18]

A function $F : \mathcal{X} \to \mathcal{Y}$ that is slow to compute (without trapdoor) but fast to publicly verify.

- $\mathsf{Setup}(\lambda, t) \to (\mathsf{pp}, \mathsf{td})$ — public params + trapdoor
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y := F(x), \pi)$ — sequential time t
- $\mathsf{TEval}(\mathsf{td}, x) \to (y, \pi)$ — trapdoor eval, time $\mathrm{polylog}(t)$
- $\mathsf{Ver}(\mathsf{pp}, x, y, \pi) \to \{0, 1\}$ — time $\mathrm{polylog}(t)$

(Trapdoor) Verifiable Delay Functions [BBBF18]

A function $F : \mathcal{X} \to \mathcal{Y}$ that is slow to compute (without trapdoor) but fast to publicly verify.

- $\mathsf{Setup}(\lambda, t) \to (\mathsf{pp}, \mathsf{td})$ — public params + trapdoor
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y := F(x), \pi)$ — sequential time t
- $\mathsf{TEval}(\mathsf{td}, x) \to (y, \pi)$ — trapdoor eval, time $\mathrm{polylog}(t)$
- $\mathsf{Ver}(\mathsf{pp}, x, y, \pi) \to \{0, 1\}$ — time $\mathrm{polylog}(t)$

Sequentiality

for random $x \leftarrow \mathcal{X}$, no adversary of depth $\ll t$ can compute $F(x)$.

(Trapdoor) Verifiable Delay Functions [BBBF18]

A function $F : \mathcal{X} \to \mathcal{Y}$ that is slow to compute (without trapdoor) but fast to publicly verify.

- $\mathsf{Setup}(\lambda, t) \to (\mathsf{pp}, \mathsf{td})$ — public params + trapdoor
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y := F(x), \pi)$ — sequential time t
- $\mathsf{TEval}(\mathsf{td}, x) \to (y, \pi)$ — trapdoor eval, time $\mathrm{polylog}(t)$
- $\mathsf{Ver}(\mathsf{pp}, x, y, \pi) \to \{0, 1\}$ — time $\mathrm{polylog}(t)$

Sequentiality

for random $x \leftarrow \mathcal{X}$, no adversary of depth $\ll t$ can compute $F(x)$.

Soundness

given (pp, x), no efficient adversary finds (y, π) with $y \neq F(x)$ and Ver accepts.

(Trapdoor) Verifiable Delay Functions [BBBF18]

A function $F : \mathcal{X} \to \mathcal{Y}$ that is slow to compute (without trapdoor) but fast to publicly verify.

- $\mathsf{Setup}(\lambda, t) \to (\mathsf{pp}, \mathsf{td})$ — public params + trapdoor
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y := F(x), \pi)$ — sequential time t
- $\mathsf{TEval}(\mathsf{td}, x) \to (y, \pi)$ — trapdoor eval, time $\mathrm{polylog}(t)$
- $\mathsf{Ver}(\mathsf{pp}, x, y, \pi) \to \{0, 1\}$ — time $\mathrm{polylog}(t)$

Sequentiality

for random $x \leftarrow \mathcal{X}$, no adversary of depth $\ll t$ can compute $F(x)$.

Soundness

given (pp, x), no efficient adversary finds (y, π) with $y \neq F(x)$ and Ver accepts.

Constructions: (iterated) sequential function + succinct proof system [Pie19, Wes19, BBBF18, FPS22]

Time-Lock Puzzles [RSW96]

A puzzle that is fast to generate but slow to solve. quick (randomized) generation of preimages.

- $\mathsf{PGen}(\lambda, t, s) \to z$ — randomized, time $\mathrm{polylog}(t)$
- $\mathsf{Sol}(z) \to s$ — sequential time t

Time-Lock Puzzles [RSW96]

A puzzle that is fast to generate but slow to solve. quick (randomized) generation of preimages.

- $\mathsf{PGen}(\lambda, t, s) \to z$ — randomized, time $\mathrm{polylog}(t)$
- $\mathsf{Sol}(z) \to s$ — sequential time t

Indistinguishability of puzzles (sequentiality):

for any s_0, s_1, $\quad \mathsf{PGen}(t, s_0) \approx_{\mathsf{depth} \ll t} \mathsf{PGen}(t, s_1)$.

Time-Lock Puzzles [RSW96]

A puzzle that is fast to generate but slow to solve. quick (randomized) generation of preimages.

- $\mathsf{PGen}(\lambda, t, s) \to z$ — randomized, time $\mathrm{polylog}(t)$
- $\mathsf{Sol}(z) \to s$ — sequential time t

Indistinguishability of puzzles (sequentiality):

for any s_0, s_1, $\quad \mathsf{PGen}(t, s_0) \approx_{\mathsf{depth} \ll t} \mathsf{PGen}(t, s_1)$.

Constructions:

- [RSW96]: repeated squaring in hidden-order groups
- [BGJ+16]: worst-case non-parallelizing languages (wcNPL) + randomized encodings

VDFs vs. TLPs: two ends of the same axis

	VDF	TLP
generate / setup	$\mathrm{polylog}(t)$	$\mathrm{polylog}(t)$
"hard" direction	evaluate in t	solve in t
verify	$\mathrm{polylog}(t)$	— (verifier has s)
reusable?	yes	no (one-time)
preprocessing?	yes	no

VDFs vs. TLPs: two ends of the same axis

	VDF	TLP
generate / setup	$\mathrm{polylog}(t)$	$\mathrm{polylog}(t)$
"hard" direction	evaluate in t	solve in t
verify	$\mathrm{polylog}(t)$	— (verifier has s)
reusable?	yes	no (one-time)
preprocessing?	yes	no

- A TLP is intuitively a *designated-verifier* VDF (verifier knows s).

VDFs vs. TLPs: two ends of the same axis

	VDF	TLP
generate / setup	$\mathrm{polylog}(t)$	$\mathrm{polylog}(t)$
"hard" direction	evaluate in t	solve in t
verify	$\mathrm{polylog}(t)$	— (verifier has s)
reusable?	yes	no (one-time)
preprocessing?	yes	no

- A TLP is intuitively a *designated-verifier* VDF (verifier knows s).
- VDF + (extractable) witness encryption $\Longrightarrow$ TLP [Ren25]

VDFs vs. TLPs: two ends of the same axis

	VDF	TLP
generate / setup	$\mathrm{polylog}(t)$	$\mathrm{polylog}(t)$
"hard" direction	evaluate in t	solve in t
verify	$\mathrm{polylog}(t)$	— (verifier has s)
reusable?	yes	no (one-time)
preprocessing?	yes	no

- A TLP is intuitively a *designated-verifier* VDF (verifier knows s).
- VDF + (extractable) witness encryption $\implies$ TLP [Ren25]

Can we build a (publicly verifiable) VDF *from* a TLP?

Our result

Main theorem

(one-time) VDF from TLP + iO + (injective) OWF.

Our result

Main theorem

(one-time) VDF from TLP + iO + (injective) OWF.

Corollary (with [BGJ+16, BPW16])

(one-time) VDF from wcNPL + (polynomially-secure) iO + OWF.

Our result

Main theorem

(one-time) VDF from TLP + iO + (injective) OWF.

Corollary (with [BGJ+16, BPW16])

(one-time) VDF from wcNPL + (polynomially-secure) iO + OWF.

Our VDF is the first to:

- achieve perfect soundness
- have pseudorandom outputs for free
- be a *generic* trapdoor VDF, incl. homomorphic & constrained trapdoors

Our result

Main theorem

(one-time) VDF from TLP + iO + (injective) OWF.

Corollary (with [BGJ+16, BPW16])

(one-time) VDF from wcNPL + (polynomially-secure) iO + OWF.

Our VDF is the first to:

- achieve perfect soundness
- have pseudorandom outputs for free
- be a *generic* trapdoor VDF, incl. homomorphic & constrained trapdoors

Caveat: sequentiality holds only *without* preprocessing (i.e., one-time) $\implies$ fresh pp per evaluation.

Step 1: a delay function from TLP + iO

Problem: a TLP samples a puzzle only *by first sampling its solution* — a VDF must publicly sample *just a puzzle*.

Step 1: a delay function from TLP + iO

Problem: a TLP samples a puzzle only *by first sampling its solution* — a VDF must publicly sample *just a puzzle*.

Idea: publish an obfuscation [GGH+13, SW14] of the puzzle sampler, derandomized by a puncturable PRF.

Step 1: a delay function from TLP + iO

Problem: a TLP samples a puzzle only *by first sampling its solution* — a VDF must publicly sample *just a puzzle*.

Idea: publish an obfuscation [GGH+13, SW14] of the puzzle sampler, derandomized by a puncturable PRF.

$\mathsf{Sample}_{[t,k_1,k_2]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$ pseudorandom solution

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$ derandomized puzzle

return z

$\mathsf{pp} := \mathsf{iO}(\mathsf{Sample})$, with t, k_1, k_2 hard-wired.

To evaluate: run Sample to get z, then $s := \mathsf{TLP.Sol}(z)$; output s.

Step 1: a delay function from TLP + iO

Problem: a TLP samples a puzzle only *by first sampling its solution* — a VDF must publicly sample *just a puzzle*.

Idea: publish an obfuscation [GGH+13, SW14] of the puzzle sampler, derandomized by a puncturable PRF.

$\mathsf{Sample}_{[t,k_1,k_2]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$ pseudorandom solution

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$ derandomized puzzle

return z

$\mathsf{pp} := \mathsf{iO}(\mathsf{Sample})$, with t, k_1, k_2 hard-wired.

To evaluate: run Sample to get z, then $s := \mathsf{TLP.Sol}(z)$; output s.

Perfect TLP correctness $\implies$ unique solution s.

Step 2: verifiability with *perfect* soundness

Problem: A SNARG would give only selective / computational soundness (or needs subexponentially-secure iO [WW24, WZ24]).

Step 2: verifiability with *perfect* soundness

Problem: A SNARG would give only selective / computational soundness (or needs subexponentially-secure iO [WW24, WZ24]).

Idea: delegate the proof to the obfuscated circuit. Let f be an injective one-way function and output $\pi := f(s)$.

Step 2: verifiability with *perfect* soundness

Problem: A SNARG would give only selective / computational soundness (or needs subexponentially-secure iO [WW24, WZ24]).

Idea: delegate the proof to the obfuscated circuit. Let f be an injective one-way function and output $\pi := f(s)$.

$\mathsf{Sample}_{[t,k_1,k_2]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$

$\pi := f(s)$ the "proof"

return (z, π)

- Eval outputs an empty proof; Ver re-runs Sample to get π and checks $f(s) = \pi$.

Step 2: verifiability with *perfect* soundness

Problem: A SNARG would give only selective / computational soundness (or needs subexponentially-secure iO [WW24, WZ24]).

Idea: delegate the proof to the obfuscated circuit. Let f be an injective one-way function and output $\pi := f(s)$.

$\mathsf{Sample}_{[t,k_1,k_2]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$

$\pi := f(s)$ the "proof"

return (z, π)

- Eval outputs an empty proof; Ver re-runs Sample to get π and checks $f(s) = \pi$.
- Wrong $s^* \neq s$ accepting $\implies f(s^*) = f(s)$, contradicting injectivity. **Perfect soundness.**

Step 2: verifiability with *perfect* soundness

Problem: A SNARG would give only selective / computational soundness (or needs subexponentially-secure iO [WW24, WZ24]).

Idea: delegate the proof to the obfuscated circuit. Let f be an injective one-way function and output $\pi := f(s)$.

$\mathsf{Sample}_{[t,k_1,k_2]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$

$\pi := f(s)$ the "proof"

return (z, π)

- Eval outputs an empty proof; Ver re-runs Sample to get π and checks $f(s) = \pi$.
- Wrong $s^* \neq s$ accepting $\implies f(s^*) = f(s)$, contradicting injectivity. **Perfect soundness.**
- **Trapdoor** $\mathsf{td} := k_1$: $\mathsf{TEval}(\mathsf{td}, x) = \mathsf{PPRF.FEval}(k_1, t \,\|\, x)$ (no solving needed).

Sequentiality: proof via punctured programming [SW14]

Goal: reduce sequentiality of the VDF to sequentiality of the TLP.

- Sequentiality game: a depth-bounded adversary gets pp and a random challenge x^*, and needs to output y^* such that $y^* := \mathsf{Eval}(\mathsf{pp}, x^*)$.

Sequentiality: proof via punctured programming [SW14]

Goal: reduce sequentiality of the VDF to sequentiality of the TLP.

- Sequentiality game: a depth-bounded adversary gets pp and a random challenge x^*, and needs to output y^* such that $y^* := \mathsf{Eval}(\mathsf{pp}, x^*)$.
- Key observation: the reduction may *pre-sample* x^* together with pp.
- Hardwire the challenge puzzle z^* and proof π^* into the obfuscated circuit.

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s^* := \mathsf{PPRF.FEval}(k_1, t \,\|\, x^*)$

$z^* := \mathsf{TLP.PGen}(s^*, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x^*))$

$\pi^* := f(s^*)$

Sample(x):

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s^*

Hybrids:

- H_0: real experiment

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s^* := \mathsf{PPRF.FEval}(k_1, t \,\|\, x^*)$

$z^* := \mathsf{TLP.PGen}(s^*, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x^*))$

$\pi^* := f(s^*)$

Sample(x):

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s^* := \$$

$z^* := \mathsf{TLP.PGen}(s^*, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x^*))$

$\pi^* := f(s^*)$

$\mathsf{Sample}(x)$:

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_2: switch $s^* \to$ random
  PPRF pseudorandomness at $t \,\|\, x^*$

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s^* := \$$

$z^* := \mathsf{TLP.PGen}(s^*, t; \mathsf{PPRF.FEval}(k_2, t \,\|\, x^*))$

$\pi^* := f(s^*)$

$\mathsf{Sample}(x)$:

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2^*, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_2: switch $s^* \to$ random
  PPRF pseudorandomness at $t \,\|\, x^*$
- H_3: puncture k_2 at $t \,\|\, x^*$
  iO + PPRF puncturing

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s^* := \$$

$z^* := \mathsf{TLP.PGen}(s^*, t; r^*)$

$\pi^* := f(s^*)$

$\mathsf{Sample}(x)$:

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2^*, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_2: switch $s^* \to$ random
  PPRF pseudorandomness at $t \,\|\, x^*$
- H_3: puncture k_2 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_4: coins $\to$ random
  now $z^* \equiv$ honest TLP puzzle

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s_1^*, s_2^* \leftarrow \$$

$z^* := \mathsf{TLP.PGen}(s_2^*, t; r^*)$

$\pi^* := f(s_1^*)$

$\mathsf{Sample}(x)$:

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2^*, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s_1^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_2: switch $s^* \to$ random
  PPRF pseudorandomness at $t \,\|\, x^*$
- H_3: puncture k_2 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_4: coins $\to$ random
  now $z^* \equiv$ honest TLP puzzle
- H_5: puzzle from s_2^*, win on s_1^*
  invoke **TLP sequentiality**

Sequentiality: the hybrids on the x^* branch

pp := iO(Sample), challenge x^*

---

precompute using original keys k_1, k_2:

$s_1^*, s_2^* \leftarrow \$$

$z^* := \mathsf{TLP.PGen}(s_2^*, t; r^*)$

$\pi^* := f(s_1^*)$

$\mathsf{Sample}(x)$:

if $x = x^*$ then return (z^*, π^*)

else $s := \mathsf{PPRF.FEval}(k_1^*, t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s, t; \mathsf{PPRF.FEval}(k_2^*, t \,\|\, x))$

return $(z, f(s))$

adversary wins iff it outputs s_1^*

Hybrids:

- H_0: real experiment
- H_1: puncture k_1 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_2: switch $s^* \to$ random
  PPRF pseudorandomness at $t \,\|\, x^*$
- H_3: puncture k_2 at $t \,\|\, x^*$
  iO + PPRF puncturing
- H_4: coins $\to$ random
  now $z^* \equiv$ honest TLP puzzle
- H_5: puzzle from s_2^*, win on s_1^*
  invoke **TLP sequentiality**
- proof leaks only $f(s_1^*) \implies$
  one-wayness of f ✓

Why only “one-time”?

- The reduction hardwires the challenge puzzle z^* into pp.
- A *preprocessing* adversary receives pp (the obfuscated circuit containing z^*) and may run unbounded sequential computation on it *before* the clock starts.
- But TLP sequentiality only holds against depth-bounded adversaries. Reduction breaks.

Extensions

Efficient-verifier VDF: the circuit also outputs a signature σ

$\mathsf{Sample}_{[t,k_1,k_2,\mathsf{sk}]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$

$\pi := f(s); \sigma := \mathsf{Sig.Sign}(\mathsf{sk},\ x \,\|\, \pi \,\|\, z)$

return (z, π, σ)

Ver: accept iff $f(s) = \pi$ and $\mathsf{Sig.Ver}(\mathsf{pk}, \sigma, x \,\|\, \pi \,\|\, z) = 1$ (no re-running obfuscated circuit!).

Extensions

Efficient-verifier VDF: the circuit also outputs a signature σ

$\mathsf{Sample}_{[t,k_1,k_2,\mathsf{sk}]}(x)$

$s := \mathsf{PPRF.FEval}(k_1,\ t \,\|\, x)$

$z := \mathsf{TLP.PGen}(s,\ t;\ \mathsf{PPRF.FEval}(k_2,\ t \,\|\, x))$

$\pi := f(s); \sigma := \mathsf{Sig.Sign}(\mathsf{sk},\ x \,\|\, \pi \,\|\, z)$

return (z, π, σ)

Ver: accept iff $f(s) = \pi$ and $\mathsf{Sig.Ver}(\mathsf{pk}, \sigma, x \,\|\, \pi \,\|\, z) = 1$ (no re-running obfuscated circuit!).

Richer trapdoors: trapdoor evaluation *is* a PRF evaluation $\implies$ richer PRFs give richer VDFs:

- Trapdoor-homomorphic VDFs: key-homomorphic (constrained) PRF [BFP+15, BV15]
- Trapdoor-constrained VDFs: constrained PRF [BW13]

The bigger picture: relating VDFs and TLPs

Known *generic* relations between the two timed primitives:

$$\text{TLP} \;+\; \text{iO} \;+\; \text{(injective) OWF} \;\Longrightarrow\; \textit{one-time}\ \text{VDF} \qquad \textbf{this work}$$

$$\text{trapdoor VDF} \;\Longrightarrow\; \text{TLP: } z \coloneqq (\mathsf{pp}, x, s \oplus y) \qquad \text{folklore}$$

$$\text{where } (\mathsf{pp}, \mathsf{td}) \leftarrow \text{VDF.Setup}, \;\; (y, \pi) \leftarrow \text{VDF.TEval}(\mathsf{td}, x)$$

$$\text{VDF} \;+\; \text{(extractable) witness encryption} \;\Longrightarrow\; \text{TLP} \qquad [\text{Ren25}]$$

The bigger picture: relating VDFs and TLPs

Known *generic* relations between the two timed primitives:

TLP + iO + (injective) OWF $\implies$ *one-time* VDF **this work**

trapdoor VDF $\implies$ TLP: $z := (\mathsf{pp}, x, s \oplus y)$ folklore

where $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{VDF.Setup}$, $(y, \pi) \leftarrow \mathsf{VDF.TEval}(\mathsf{td}, x)$

VDF + (extractable) witness encryption $\implies$ TLP [Ren25]

. . . and these detours are *inherent* [AAA+26]:

No *fully black-box* construction of VDF from TLP.

Conclusion

A new design paradigm: TLP + iO + OWF $\Longrightarrow$ (one-time) VDF.

- Perfect soundness, pseudorandom outputs, generic trapdoor VDF.
- Homomorphic/constrained trapdoors.

Conclusion

A new design paradigm: TLP + iO + OWF $\implies$ (one-time) VDF.

- Perfect soundness, pseudorandom outputs, generic trapdoor VDF.
- Homomorphic/constrained trapdoors.

Open questions:

- VDFs from TLPs *with* preprocessing? (must be non-black-box [AAA+26])
- VDFs from TLPs without iO (even one-time secure)?

Conclusion

A new design paradigm: TLP + iO + OWF $\implies$ (one-time) VDF.

- Perfect soundness, pseudorandom outputs, generic trapdoor VDF.
- Homomorphic/constrained trapdoors.

Open questions:

- VDFs from TLPs *with* preprocessing? (must be non-black-box [AAA+26])
- VDFs from TLPs without iO (even one-time secure)?

Thank you!

ia.cr/2025/1782

References I

[AAA+26] Hamza Abusalah, Nivesh Aggarwal, Karen Azari, Chethan Kamath, and Maximilian von Consbruch.
Separating verifiable delay functions and time-lock puzzles.
In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 33–63, Cham, 2026. Springer Nature Switzerland.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch.
Verifiable delay functions.
In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Cham, August 2018.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens.
Key-homomorphic constrained pseudorandom functions.
In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, March 2015.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters.
Time-lock puzzles from randomized encodings.
In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.

References II

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs.
Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation.
In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 474–502. Springer, Berlin, Heidelberg, January 2016.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan.
Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF.
In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Berlin, Heidelberg, March 2015.

[BW13] Dan Boneh and Brent Waters.
Constrained pseudorandom functions and their applications.
In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Berlin, Heidelberg, December 2013.

[FPS22] Cody Freitag, Rafael Pass, and Naomi Sirkin.
Parallelizable delegation from LWE.
In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 623–652. Springer, Cham, November 2022.

References III

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters.
Candidate indistinguishability obfuscation and functional encryption for all circuits.
In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[Pie19] Krzysztof Pietrzak.
Simple verifiable delay functions.
In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

[Ren25] Omar Renawi.
Time-Lock Puzzles from Verifiable Delay Functions and Witness Encryption.
12 2025.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner.
Time-lock puzzles and timed-release crypto.
Technical Report MIT/LCS/TR-684, Massachusetts Institute of Technology, 1996.

[SW14] Amit Sahai and Brent Waters.
How to use indistinguishability obfuscation: deniable encryption, and more.
In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

References IV

[Wes19] Benjamin Wesolowski.
Efficient verifiable delay functions.
In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Cham, May 2019.

[WW24] Brent Waters and David J. Wu.
Adaptively-sound succinct arguments for NP from indistinguishability obfuscation.
In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024.

[WZ24] Brent Waters and Mark Zhandry.
Adaptive security in SNARGs via iO and lossy functions.
In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 72–104. Springer, Cham, August 2024.