NutMiC'19, June, 2019

Verifiable Delay Functions: How to Slow Things Down (Verifiably)

Dan Boneh

Stanford University

What is a VDF?

(verifiable delay function)

Intuition: a function X ⟶ Y that

(1) takes time T to evaluate, even with polynomial parallelism,

(2) the output can be verified efficiently

- Setup(λ, T) ⟶ public parameters *pp*
- Eval(*pp*, $\boldsymbol{x}$) ⟶ output $\boldsymbol{y}$, proof $\boldsymbol{\pi}$ (parallel time $\boldsymbol{T}$)
- Verify(*pp*, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\pi}$) ⟶ { *yes*, *no* } (time poly(λ, **log** $\boldsymbol{T}$))

Security Properties (simplified)

[B-Bonneau-Bünz-Fisch'18]

- Setup(λ, T) ⟶ public parameters *pp*
- Eval(*pp*, ***x***) ⟶ output ***y***, proof ***π*** (parallel time T)
- Verify(*pp*, ***x***, ***y***, ***π***) ⟶ { *yes, no* } (time poly(λ, log T))

"**Uniqueness**": if Verify(*pp*, *x*, ***y*, *π***) = Verify(*pp*, *x*, ***y'*, *π'***) = yes

then ***y = y'***

"**ε-Sequentiality**": for all parallel algs. *A*, time(*A*) < (1-ε)·time(Eval),

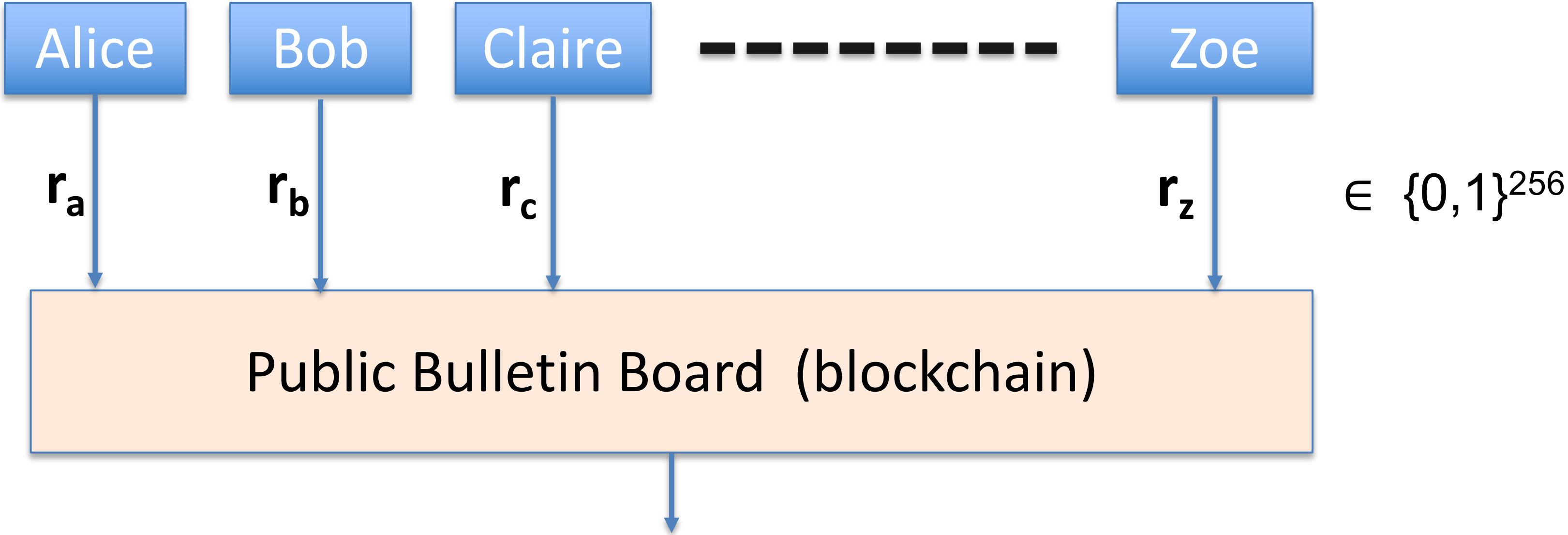
for random x∈X, *A* cannot distinguish **Eval(*pp*, *x*)** from a random y∈Y

Application: lotteries

Problem: generating verifiable randomness in the real world?

Standard solutions are unsatisfactory

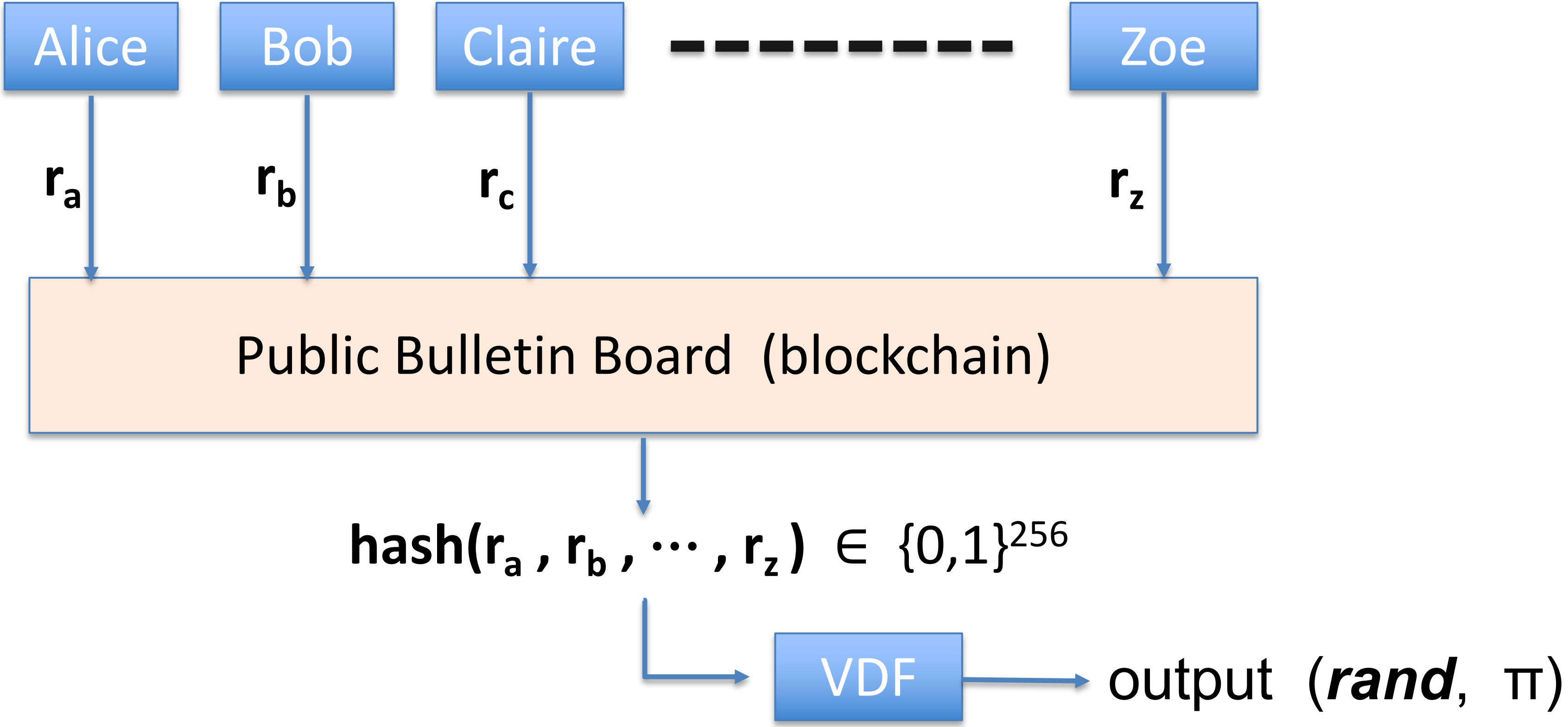
Broken method: distributed generation



output $\textbf{\textit{rand}} = \mathbf{r_a} \oplus \mathbf{r_b} \oplus \cdots \oplus \mathbf{r_z} \quad \in \{0,1\}^{256}$

Problem: Zoe controls value of ***rand*** !!

Solution: slow things down with a VDF [LW'15]



$\mathbf{hash}(\mathbf{r}_a , \mathbf{r}_b , \cdots , \mathbf{r}_z) \in \{0,1\}^{256}$

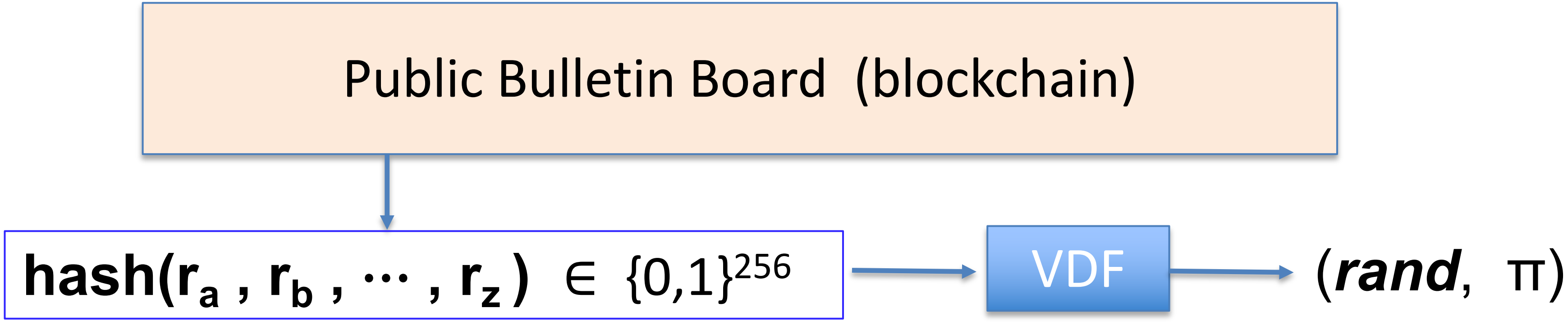
output (***rand***, π)

Solution: slow things down with a VDF

- Submissions: start at 12:00pm, end at 12:10pm
- VDF delay: about one hour (≫ 10 minutes)

Sequentiality: ensures Zoe cannot bias output

Uniqueness: ensures no ambiguity about output



Public Bulletin Board (blockchain)

hash$(\mathbf{r_a}, \mathbf{r_b}, \cdots, \mathbf{r_z}) \in \{0,1\}^{256}$

VDF

$(\textbf{\textit{rand}}, \pi)$

Being implemented and deployed …

ETHEREUM
v2.0

Construction 1: from hash functions

Hash function $H\colon \{0,1\}^{256} \longrightarrow \{0,1\}^{256}$ (e.g. SHA256)

- pp = (public parameters for a SNARK)

$$H^{(T)}(x) = \underbrace{H(H(H(H(H(\ldots (H(H(x))) \ldots)))))}_{T \text{ times}} \qquad \text{(sequential work)}$$

- Eval(pp, x): output $\boldsymbol{y = H^{(T)}(x)}$, proof **π = (SNARK)**

- Verify(pp, **x**, **y**, **π**): accept if SNARK proof is valid

Construction 1: from hash functions

Problem: computing SNARK proof $\boldsymbol{\pi}$ takes longer than computing $\mathbf{y} = H^{(T)}(\mathbf{x})$

$\Rightarrow$ adversary can compute **y** long before Eval(**pp**, **x**) finishes

Simple solution using $\log_2(T)$-way parallelism [B-Bonneau-Bünz-Fisch'18]

Construction 2: exponentiation

Why?

G: finite abelian group

- **Assumption 1**: the order of G cannot be efficiently computed

pp = (G, H: $X \longrightarrow G$)

T squarings, e.g. T = 10^9

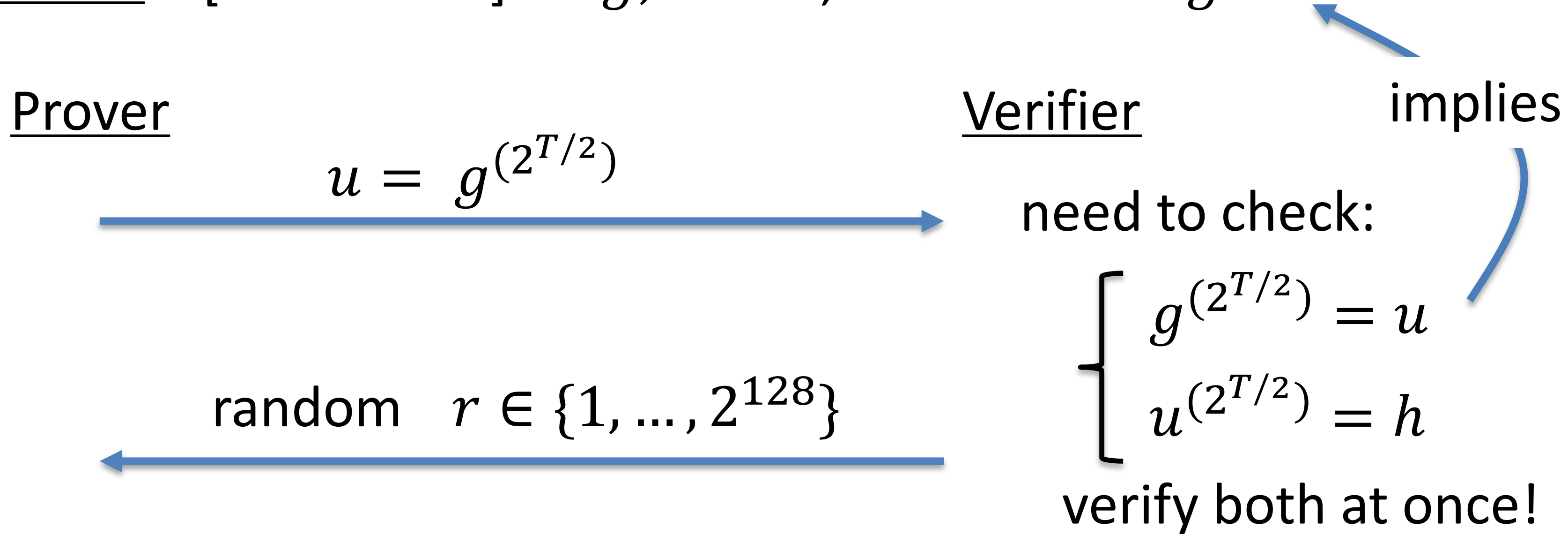
- **Eval(pp, x):** output $y = H(x)^{(2^T)} \in G$

  need proof **π = (proof of correct exponentiation)**

[Pietrzak'18, Wesolowski'18]

Proof of correct exponentiation (T=power of 2)

Method 1: [Pietrzak'18] $g, h \in G$, claim: $h = g^{(2^T)}$



Prover

Verifier

implies

$u = g^{(2^{T/2})}$

need to check:

$$\begin{cases} g^{(2^{T/2})} = u \\ u^{(2^{T/2})} = h \end{cases}$$

random $r \in \{1, \dots, 2^{128}\}$

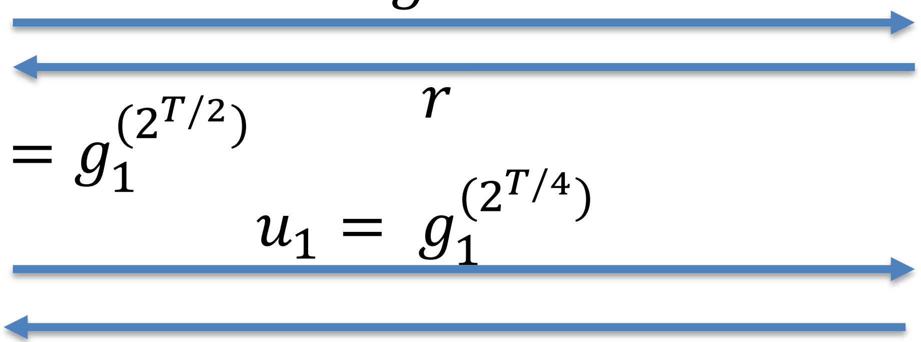
verify both at once!

Set $g_1 = g^r u$, $h_1 = u^r h.$ Recursively prove $\boxed{h_1 = g_1^{(2^{T/2})}}$

Proof of correct exponentiation [P’18]

Prover (g, h)

Verifier (g, h)



$u = g^{(2^{T/2})}$

r

$g_1 = g^r u$, $h_1 = u^r h$

claim: $h_1 = g_1^{(2^{T/2})}$

$u_1 = g_1^{(2^{T/4})}$

r_1

$g_2 = g^{r_1} u$, $h_2 = u^{r_1} h$

⋮ ($\log T$ rounds)

claim: $h_{\log T} = g_{\log T}^2$

compute: $h_{\log T}$, $g_{\log T}$

accept if $\boxed{h_{\log T} = g_{\log T}^2}$

Proof **π** = $(u, u_1, \dots, u_{\log T})$

Proof of correct exponentiation [P'18]

As a non-interactive proof:

- Proof **π** = $\left(u, u_1, \ldots, u_{\log T}\right)$ via the Fiat-Shamir heuristic

$$r_i = \text{hash}(g, h,\ u, r, \ldots, u_{i-1}, r_{i-1}, u_i), \qquad i = 1, \ldots, \log T$$

Computing the proof **π:** fast, only $O(\sqrt{T})$ steps

- By storing $\sqrt{T}$ values while computing $g^{(2^T)}$

Soundness

<u>**Theorem**</u> [BBF'18] (informal): suppose $h \neq g^{(2^T)}$,

but prover *P* convinces verifier (with non-negligible probability ϵ).

Then there is an algorithm, whose run time is twice that of *P*, that outputs (with prob. ϵ^2)

$(\boldsymbol{w}, \boldsymbol{d})$ **where** $\mathbf{1} \neq \boldsymbol{w} \in \boldsymbol{G}$ **and d < 2¹²⁸** **such that** $\boldsymbol{w}^{\boldsymbol{d}} = \mathbf{1}$

so: $\overbrace{\text{hard to find } 1 \neq w \in G \text{ of known order}}^{\text{assumption 2}} \Rightarrow$ protocol is secure

Assumption 2 is necessary for security

Suppose some (w, d) is known where $1 \neq w \in G$ and $w^d = 1$.

$\Rightarrow$ Prover can cheat with probability $1/d$

How? $\boxed{\text{set} \quad h = \boldsymbol{w} \cdot g^{(2^T)} \neq g^{(2^T)} \;, \qquad u = \boldsymbol{w} \cdot g^{(2^{T/2})}}$

Now, verifier falsely accepts whenever $\underbrace{r + 1 \equiv 2^{T/2} \quad (mod\ d)}_{\text{holds with prob. 1/d}}$

why? in this case: $\underset{u^r h}{\underset{\shortparallel}{h_1}} = \underset{(g^r h)^{(2^{T/2})}}{\underset{\shortparallel}{g_1^{(2^{T/2})}}}$

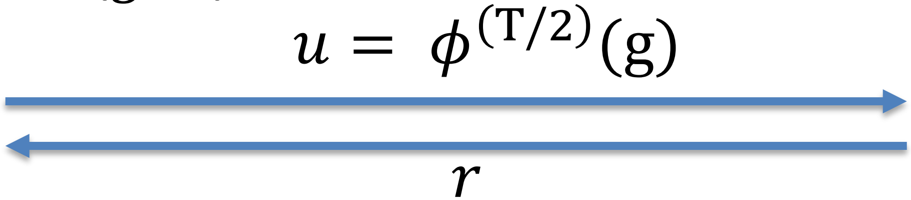
More generally … nothing special about squaring

G: finite abelian group. $\phi : G \to G$ an endomorphism

$\boldsymbol{g, h \in G}$ **,** **claim:** $\boldsymbol{h = \phi^{(\mathrm{T})}(\mathrm{g})}$

<u>Prover</u> (g, h)

<u>Verifier</u> (g, h)



$u = \phi^{(\mathrm{T}/2)}(\mathrm{g})$

r

$g_1 = g^r u$, $h_1 = u^r h$

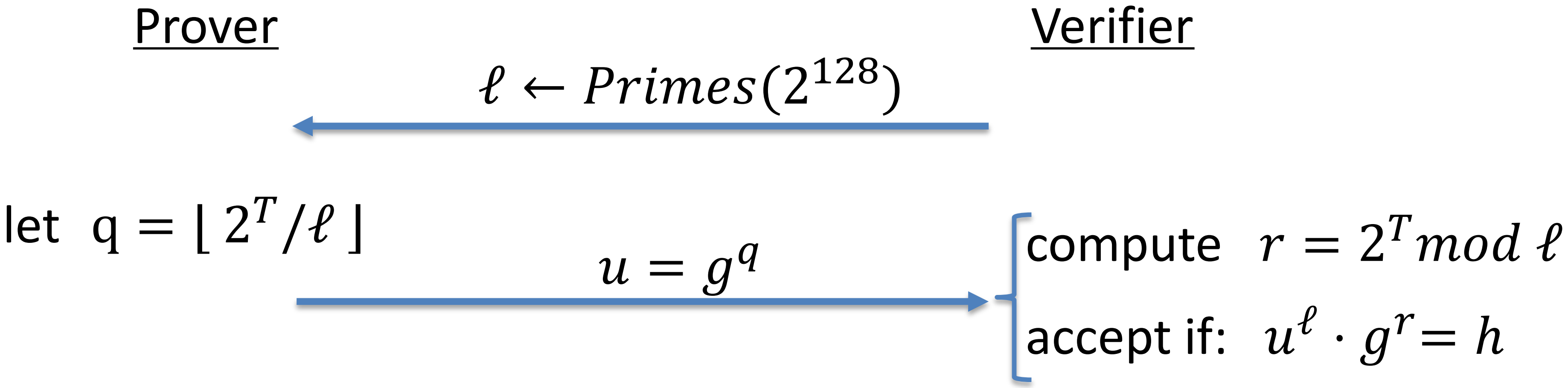
claim: $h_1 = \phi^{(T/2)}(\mathrm{g}_1)$

⋮

Proof $\boldsymbol{\pi}$ = $(u, u_1, \dots, u_{\log T})$

Proof of correct exponentiation: method 2

Method 2: [Wesolowski’18] $g, h \in G$, claim: $h = g^{(2^T)}$



Prover

Verifier

$\ell \leftarrow Primes(2^{128})$

let q $= \lfloor 2^T / \ell \rfloor$

$u = g^q$

compute $r = 2^T mod\ \ell$

accept if: $u^\ell \cdot g^r = h$

Proof $\boldsymbol{\pi} = (u)$

single element!

Soundness

Need assumption 2: hard to find $1 \neq w \in G$ of known order

… but is not sufficient

Security relies on a stronger assumption

called the *adaptive root assumption*.

Candidate abelian groups

Goal: group G with no elements ≠1 of known order

- **n** $\in \mathbb{Z}$, unknown factorization. $G_n = (\mathbb{Z}/n)^* / \{\pm 1\}$

  Con: trusted setup to generate n (or a large random n)

- $p \equiv 3 \ (mod\ 4)$ prime. G_p = class group of $\mathbb{Q}(\sqrt{-p})$.

  Con: no setup, but complex operation (slow verify)

  Pro: can switch group every few minutes ⇒ smaller params

Candidate abelian groups

Goal: group G v

Note DJB parallelism for exponentiation in G_n

- **n** $\in \mathbb{Z}$, unknown factorization. $G_n = (\mathbb{Z}/n)^*/\{\pm 1\}$

  Con: trusted setup to generate n (or a large random n)

- $p \equiv 3 \ (mod\ 4)$ prime. G_p = class group of $\mathbb{Q}(\sqrt{-p})$.

  Con: no setup, but complex operation (slow verify)

  Pro: can switch group every few minutes ⇒ smaller params

Assumption 2 in class groups?

hard to find $1 \neq w \in G_p$ of known small order

Cohen-Lenstra: frequency d divides $|G_p|$:

d=3: 44%, d = 5: 24%, d = 7: 16%

Open: When 3 divides $|G_p|$,

can we efficiently find an element of order 3 in G_p?

The Chia class group challenge

Recent class number record: 512-bit discriminant

- *Beullens, Kleinjung, Vercauteren 2019:*

The Chia challenge: computing larger class numbers

- Are there interesting discriminants to include in challenge?

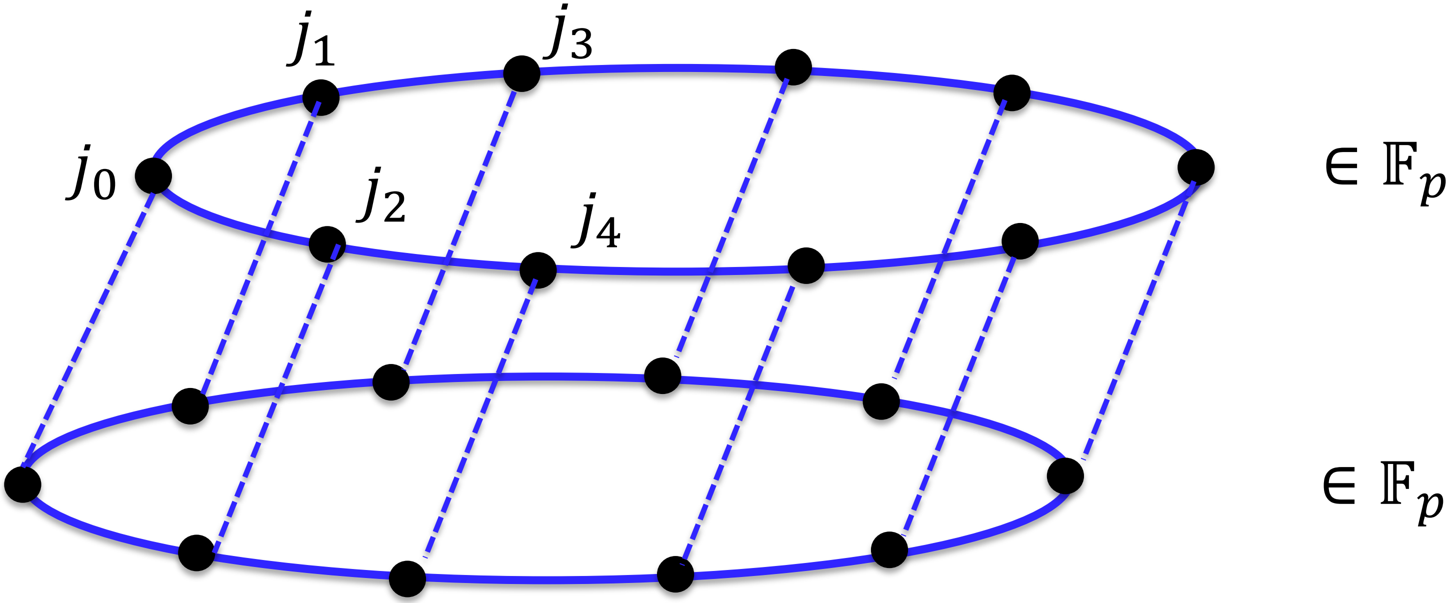
https://github.com/Chia-Network/vdf-competition

VDF construction 3: isogenies

[De Feo, Masson, Petit, Sanso’ 19]

Degree-2 supersingular isogeny classes over $\mathbb{F}_p$: $(p \equiv 7 \ mod \ 8)$

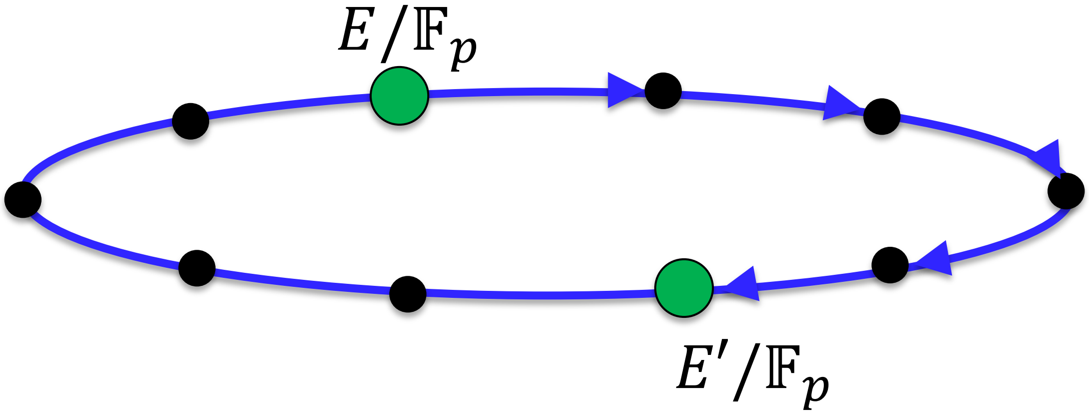
(curves and isogenies defined over $\mathbb{F}_p$)



VDF construction 3: isogenies

[De Feo, Masson, Petit, Sanso' 19]

Degree-2 supersingular isogeny classes over $\mathbb{F}_p$: $(\mathrm{p} \equiv 7 \ mod \ 8)$



T steps

$$\phi : E \to E' \ , \qquad \hat{\phi} : E' \to E \ , \qquad \deg(\phi) = 2^T$$

Tools

$$|E(\mathbb{F}_\mathrm{p})| = |E'(\mathbb{F}_\mathrm{p})| = p + 1.$$

$$\begin{tikzcd} E \arrow[r, shift left, "\phi"] & E' \arrow[l, shift left, "\hat{\phi}"] \end{tikzcd}$$

Let $\ell \mid p + 1$ be a large prime factor of $p + 1$

<u>Fact</u>: For all $P \in E[\ell] \cap E\left(\mathbb{F}_\mathrm{p}\right)$ and $P' \in E'[\ell] \cap E'\left(\mathbb{F}_\mathrm{p}\right)$

$$\underbrace{\hat{\boldsymbol{e}}_\ell(\boldsymbol{P},\ \widehat{\boldsymbol{\phi}}(\boldsymbol{P}'))}_{\text{non-degenerate pairing on } E} = \underbrace{\hat{\boldsymbol{e}}'_\ell(\boldsymbol{\phi}(\boldsymbol{P}),\ \boldsymbol{P}')}_{\text{non-degenerate pairing on } E'}$$

The VDF (over $\mathbb{F}_\mathrm{p}$)

[De Feo, Masson, Petit, Sanso' 19]

Setup: (1) choose $P \in E[\ell] \cap E(\mathbb{F}_\mathrm{p})$, compute $P' = \phi(P)$

(2) $H : X \to E'[\ell] \cap E'(\mathbb{F}_\mathrm{p})$

$$pp = (E, E', H, \phi, P, P')$$

No proof π !!

$\boxed{\textbf{Eval}(pp, x) = \hat{\phi}\big(H(x)\big)}$ (T steps)

Verify(pp, x, y): accept if $\boxed{\hat{\boldsymbol{e}}_\ell(\boldsymbol{P}, \boldsymbol{y}) = \hat{\boldsymbol{e}}'_\ell(\boldsymbol{P}', \boldsymbol{H}(\boldsymbol{x}))}$

and $y \in E[\ell] \cap E\big(\mathbb{F}_\mathrm{p}\big)$.

Does Eval take T steps?

Can an attacker find a low degree isogeny $\psi: E' \to E$??

Answer: yes, if $End_{\bar{\mathbb{F}}_p}(E)$ is known [Kohel, Lauter, Petit, Tignol, 2014]

Solution: use a trusted setup to generate a supersingular $E/\mathbb{F}_\mathrm{p}$ s.t. $End_{\bar{\mathbb{F}}_p}(E)$ is unknown

Summary and open problems

VDFs are an important new primitive

- Several elegant constructions, but looking for more.

Problem 1: is there a simple fully post-quantum VDF?

Problem 2: other groups of unknown order?

- goal: no trusted setup and fast group operation

To learn more: see survey at https://eprint.iacr.org/2018/712

THE END