Efficient Verifiable Delay Functions

Benjamin Wesolowski – EUROCRYPT 2019

VDF

VDF

A verifiable delay function

VDF

A verifiable **delay function**

Evaluation requires at least some given time

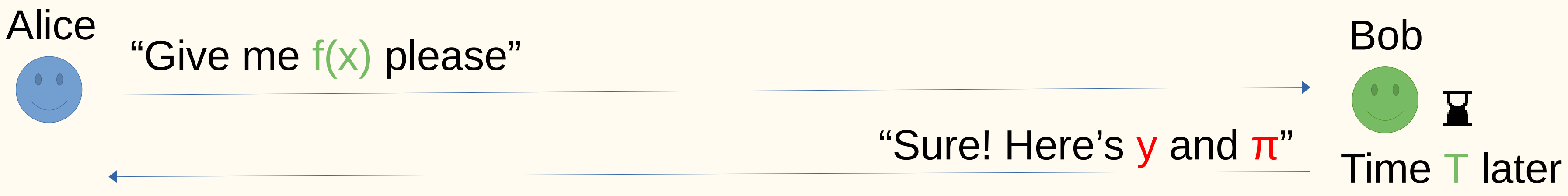
VDF

A **verifiable** **delay function**

Result easily verifiable

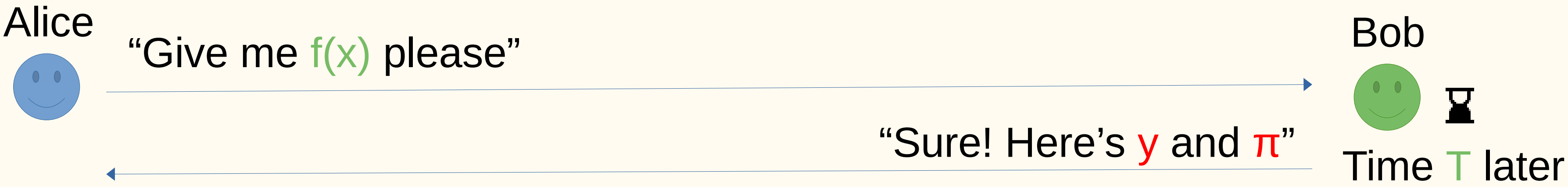
VDF

A **verifiable delay function**



VDF

A **verifiable delay function**



- *f* – function taking time *T* to compute
- *x* – The input of Alice
- *y* – The output of the function
- *π* – Proof that *y* is the correct output

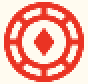
Efficient VDF

An **efficient** **verifiable** **delay function**

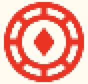
1) Size of proof π
2) Evaluation time of f
3) Verification time of result y

Why VDF?

Randomness beacons

 and  went to the casino 

To play 

But! They don’t trust 

So they propose a scheme to generate a random outcome, trusted by all parties!

commit-and-reveal

Step 1	Random r_a	Random r_b	Random r_c
Step 2	Send H(r_a)	Send H(r_b)	Send H(r_c)
Step 3	Reveal r_a	Reveal r_b	Reveal r_c
Step 4	Combine all values to get trusted random		

commit-and-reveal

			! 
Step 1	Random r_a	Random r_b	Random r_c
Step 2	Send H(r_a)	Send H(r_b)	Send H(r_c)
Step 3	Reveal r_a	Reveal r_b	Observe r_a and r_b Reveal r_c only if good outcome

VDF based

Step 1	Random r_a	Random r_b	Random r_c
Step 2	Send r_a	Send r_b	Send r_c

Time limit at most $q < T$

Step 3

Run VDF (takes time T) on r_a, r_b and r_c

↓

Trusted random value

Related Work

Background & Related work

Time locks – Rivest, Shamir, Wagner – 1996

Slow-timed hash functions – Lenstra, Wesolowski – 2016

VDFs – Boneh et al – 2018

Simple VDFs – Pietrzak – 2018

Foundations

What is time?

Amount of **sequential** work in a given model M

Attacker’s model not exactly specified

All operations an attacker could perform

Cost function *c*

Time-cost function *t*

Measuring an algorithm:

$C(A,x)$

$T(A,x)$

2 Constructions

Need: group of unknown order

RSA vs Class Group of Imaginary Quadratic Field

- Simpler to work with
- Tricky to achieve unknown order

- Tricky to work with
- Simpler to achieve unknown order

RSA Setup

Basic Idea

Given group G of **unknown** order, a timing parameter t,
for a random $x \in G$, compute $y = x^{2^t}$.

Given a random prime l, compute the proof $\pi = x^{\lfloor 2^t / l \rfloor}$.
With $r = 2^t \bmod l$, verify by checking that $y = \pi^l g^r$.

For RSA setup, $G = (\mathbf{Z}/N\mathbf{Z})^\times / \{\pm 1\}$ for an RSA modulus N,
with N of **unknown factorization**.

(δ,t)-Time-Lock Game

Let $k \in \mathbf{Z}_{>0}$ be a difficulty parameter (entropy), and A be an algorithm playing the game. With $t \in \mathbf{Z}_{>0}$, $\delta : \mathbf{Z}_{>0} \rightarrow \mathbf{R}_{>0}$ ($\delta(k)$ gives the time-cost of computing a single squaring in the group). The game goes as follows:

1. An RSA modulus N is generated randomly, for the security parameter k;
2. $A(N)$ outputs an algorithm B;
3. An element $g \in \mathbf{Z}/N\mathbf{Z}$ is uniformly randomly generated;
4. $B(g)$ outputs $h \in \mathbf{Z}/N\mathbf{Z}$.

A wins the game if $h = g^{2^t} \bmod N$ and $T(B,g) < t\,\delta(k)$, where $T(B,g)$ is the time for $B(g)$ to output h.

Time-Lock Assumption

Proposed by Rivest, Shamir and Wagner. Can be expressed as follows:

There is a $\delta : \boldsymbol{Z}_{>0} \rightarrow \boldsymbol{R}_{>0}$ such that:

1. $\exists$ an algorithm S, s.t. $\forall$ RSA modulus N generated with security parameter k, and any $g \in \mathbf{Z}/N\mathbf{Z}$, $S(N, g) = g^2 \bmod N$, and the time-cost $T(S, (N, g)) < \delta(k)$; and
2. $\forall t \in \boldsymbol{Z}_{>0}$, no algorithm A of **polynomial cost** * wins the (δ, t)-time-lock game with non-negligible probability (w.r.t k).

* One can define the cost as the **size of the circuit** used for computation.

Trapdoor VDF

$\text{keygen} \rightarrow (pk, sk)$	Generate a public-private key pair
$\text{trapdoor}_{sk}(x, \Delta) \rightarrow (y, \pi)$	Compute y using the secret key (cheating), along with a proof
$\text{eval}_{pk}(x, \Delta) \rightarrow (y, \pi)$	Compute y using the public key, along with a proof
$\text{verify}_{pk}(x, y, \pi, \Delta) \rightarrow \text{Boolean}$	Check if y is the correct output for x, possibly with the help of π

Construction: RSA Setup – Evaluation

Let $H:\{0,1\}^* \to \{0,1\}^{2k}$ denote a secure cryptographic hash function;
$N = pq$, where p, q be 2 primes generated by an RSA keygen routine;
and $g = H_N(x) = \text{int}(H(x)) \bmod N$.

$\text{keygen} \to (pk, sk)$	$pk = (N, H_N), sk = (p-1)(q-1)$
$\text{trapdoor}_{sk}(x, t) \to (y, \pi)$	$y = g^{2^t \bmod sk} \bmod N$
$\text{eval}_{pk}(x, \Delta) \to (y, \pi)$	$y = g^{2^t} \bmod N$

Construction: RSA Setup – Verification

Let $\text{Primes}(2k)$ denote the set containing the first 2^{2k} prime numbers, l be a prime number sampled uniformly at random from $\text{Primes}(2k)$, and $r = 2^t \bmod l$

$$\text{trapdoor}_{sk}(x, t) \rightarrow (y, \pi) \qquad \pi = g^{\lfloor 2^t / l \rfloor \bmod sk} \bmod N$$

$$\text{eval}_{pk}(x, \Delta) \rightarrow (y, \pi) \qquad \pi = g^{\lfloor 2^t / l \rfloor} \bmod N$$

$$\text{verify}_{pk}(x, y, \pi, \Delta) \rightarrow \text{Boolean} \qquad y == \pi^l g^r \bmod N$$

Fiat–Shamir Transformation and Optimization

The protocol can be made **non-interactive** by letting $l = H_{\text{prime}}(g, y)$, where H_{prime} is a hash function which maps the input into an element of $\text{Primes}(2k)$.

Typically, an evaluator would compute the output y and the proof π, and send the pair (y, π) to the verifiers. And the verifiers would compute l from g and y.
Instead, it's also possible to transmit (l, π) and compute y from g, π and l, and verify that $l == H_{\text{prime}}(g, y)$. This reduces the bandwidth and storage footprint almost by 2 ($\text{sizeof}(l) \sim 10^2$, $\text{sizeof}(y) \sim 10^3$).

Computing the Proof π (~ O(t))

Data: an element g in a group G (with identity 1_G), a prime number ℓ and a positive integer t.

Result: $g^{\lfloor 2^t/\ell \rfloor}$.

$x \leftarrow 1_G \in G$;

$r \leftarrow 1 \in \mathbf{Z}$;

for $i \leftarrow 0$ **to** $T-1$ **do**

  $b \leftarrow \lfloor 2r/\ell \rfloor \in \{0,1\} \in \mathbf{Z}$;

  $r \leftarrow$ least residue of $2r$ modulo ℓ;

  $x \leftarrow x^2 g^b$;

end

return x;

Algorithm 4: Simple algorithm to compute $g^{\lfloor 2^t/\ell \rfloor}$, with an on-the-fly long division [5].

Computing the Proof π (~ O(t / log(t)))

Let $p = \lfloor 2^t / l \rfloor,\ B = 2^\kappa$ (where $\kappa \in [1, t]$).

$$\Rightarrow p = \sum_{i=0}^{..} b_i B^i \quad = \sum_{b=0}^{B-1} b \left(\sum_{i | b_i = b} B^i \right) \quad \text{(where } b_i \in [0, B-1] \text{)}$$

$$\mathrm{T} \sim t/\kappa + \kappa 2^\kappa \overset{\kappa = \log(t)/2}{\sim} O(t/\log(t)),\ \mathrm{C} \sim t/\kappa \sim O(t/\log(t))$$

$$= \sum_{i=0}^{..} \sum_{j=0}^{\gamma-1} b_{i,j} B^{j+i\gamma} \quad \text{(where } b_{i,j} \in [0, B-1], \gamma \in [1, \tfrac{t}{\kappa}] \text{)}$$

$$= \sum_{j=0}^{\gamma-1} B^j \sum_{i=0}^{..} b_{i,j} B^{i\gamma} \quad = \sum_{j=0}^{\gamma-1} B^j \left(\sum_{b=0}^{B-1} b \left(\sum_{i | b_{i,j} = b} B^{i\gamma} \right) \right)$$

$$\mathrm{T} \sim t/\kappa + \gamma \kappa 2^\kappa \overset{\kappa = \log(t)/3,\ \gamma = \sqrt{t}}{\sim} O(t/\log(t)),\ \mathrm{C} \sim \sqrt{t}$$

Proof Shortness vs Prover Efficiency

The computation of π can only start after the evaluation of the VDF output g^{2^t} is completed, which results in an inevitable overhead $\frac{T}{\omega}$.

Such an overhead can be **reduced** by computing a proof π_1 for the intermidiate result $g_1 = g^{2^{t_1}}$, where $t_1 = \frac{t\omega}{\omega+1}$, which can start earlier in parallel. In the end, we just need to complement the proof with another proof π_2 for $y = g_1^{2^{t/(\omega+1)}}$, resulting in an overhead $\sim \frac{T}{\omega^2}$.

This trick can be applied recursively.

Δ-Evaluation Race Game

Let A be a party playing the game. With $\Delta : \mathbb{Z}_{>0} \rightarrow \mathbb{R}_{>0}$ a function of the security parameter k*, the game goes as follows:

1. keygen outputs pk;
2. $A(pk)$ outputs an algorithm B;
3. An x is generated randomly according to some random distribution of min-entropy at least k;
4. $B^O(x)$ outputs y, where O is an oracle that outputs $\text{trapdoor}_{sk}(x', \Delta)$ on any input $x' \neq x$.

A wins the game if $\text{eval}_{pk}(x, \Delta)$ outputs y and $T(B, X) < \Delta$.

*Think of $\Delta(k) \sim t\,\delta(k)$ for some specific t and δ.

Δ-Sequentiality

A trapdoor VDF is Δ-sequential if any polynomially bounded player (with respect to the implicit security parameter) wins the Δ-evaluation race game with negligible probability.

Remark: Suppose the input x is hashed as $H(x)$ (by a secure cryptographic function) before being evaluated, i.e.

$$\text{trapdoor}_{sk}(x,\Delta)=t_{sk}(H(x),\Delta)$$

for some procedure t, and similarly for $eval$ and $verify$. Then, it is unnecessary to give to B access to the oracle O, i.e. the application of H offsets any potential advantage gained from getting access to the oracle O.

Proof of (tδ)-Sequentiality

Let A be a player winning the $(t\delta)$-evaluation race game with probability p_{win} under the random oracle model, who is limited to q oracle queries.

1. show that there is a player C for the (δ, t)-time-lock game with a winning probability of at least $\left(1-q/2^k\right)p_{\text{win}}$. (i.e. if A can win with non-negligible probability, so can C)
2. By the time-lock assumption, C cannot win with non-negligible probability, and therefore neither can A do so.
3. Hence, p_{win} must be negligible $\Rightarrow (t\delta)$-sequentiality holds.

Soundness

A trapdoor VDF is sound if any polynomially bounded algorithm solves the following **soundness-breaking game** with negligible probability:

given as input the public key pk, output a message x, a value y' and a proof π' such that $y' \neq \text{eval}_{pk}(x, \Delta)$, and $\text{verify}_{pk}(x, y', \pi', \Delta) = \text{true}$.

Root Finding Game

Let A be a party playing the game. The game goes as follows:

1. keygen outputs an RSA modulus N, which is given to A;
2. A outputs a $u \in \mathbf{Z}/N\mathbf{Z}$;
3. An integer l is sampled uniformly from $\text{Primes}(2k)$ and given to A;
4. A outputs a $v \in \mathbf{Z}/N\mathbf{Z}$.

A wins the game if $v^l = u \neq \pm 1 \bmod N$.

*No known reduction of this problem to standard assumptions such as factoring N or the RSA problem.

Proof of Soundness

Let A be a player winning the soundness-breaking game with probability p_{win} under the random oracle model, who is limited to q oracle queries.

1. show that there is a player D for the root finding game with a winning probability of at least $p_{\text{win}}/(q+1)$. (i.e. if A can win with non-negligible probability, so can D)
2. Since the root finding problem is (believed to be) hard, C cannot win with non-negligible probability, and therefore neither can A do so.
3. Hence, p_{win} must be negligible $\Rightarrow$ soundness holds

Generalizations and Extensions

- Construction can be generalized to be based on other sets of finite groups, as long as the (generalized) assumptions also hold
- In particular, the paper recommends the class group of an imaginary quadratic field of discriminant d. Advantages include simpler group generation process than RSA setup.
- The proofs can be aggregated: producing a single short proof that simultaneously proves the correctness of several VDF evaluations.
- The proofs can be watermarked: tying a proof to the evaluator’s identity.

Conclusion

Summary

- VDFs and Efficiency
- Applications of VDFs
- Time-lock & groups of unknown order
- Setup & Proof complexity
- Sequentiality

Thank you for your attention!