Chia and VDF (verifiable delay function)

Introduction

VDF 是一個在區塊鏈的世界中加入延遲的重要工具，或是說一段真實世界時間經過的證明。VDF 的特色是就算使用平行運算加速，VDF 依然能夠保證一定時間的經過。VDF 的 output 需要經過一連串的迭代計算完成，無數次的迭代使得電腦需要耗費一定時間計算，並且每次迭代的數值是由上一次迭代的輸出決定，因此很難被平行運算加速。

在 Dan Boneh 提出了 VDF 概念以及形式化之後，Wesolowski (2018)和 Pietrzak(2018)分別提出的兩種構造 VDF 的方法，兩個方法都是對一個未知階的群上做連續平方運算。Wesolowski 的證明比較短、驗證更快，但是 Pietrzak 的構造中，生成證明的速度比較快。產生群的方式一般是使用 RSA 群，即生成兩個非常大且不相近的質數 p, q，使得 pq = N。在無法對 N 做因式分解的情況下，prover 只能乖乖地對 VDF 進行連續平方的運算。使用 RSA 群的問題是需要一個 trusted setup 生成兩個 p, q，而且不能被洩漏出去，否則安全性將大打折扣。在 Dan Boneh 對兩篇論文的調查中提到，使用一個虛二次數域的類群會是一個更好的方式。

Chia Network 在 Proof of Space 的基礎上加入 VDF，延遲農夫出快的時間，產生新的 Proof of Sapce and Time。Chia Network 最後選擇使用 Pietrzak 構造的 VDF。

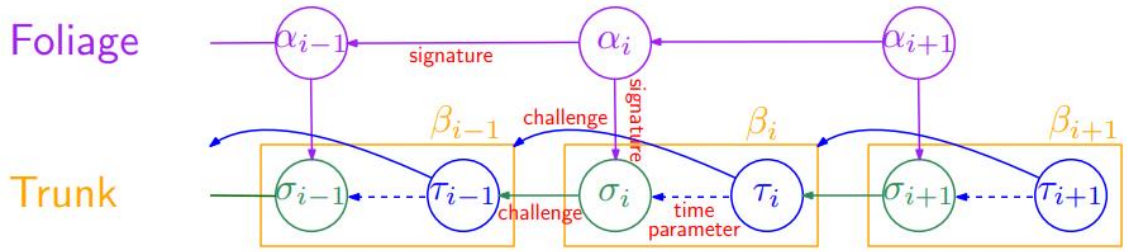

Figure 1: Illustration of the Chia blockchain that will be explained in §3. A block $\beta = (\sigma, \tau)$ in the (ungrindable) trunk chain contains a proof of space σ followed by a VDF output τ. A block α in the foliage contains the payload (transactions, timestamp), a signature of the previous block in foliage (to chain these blocks), and a signature of the proof of space (to bind the foliage to the trunk). This figure just illustrates the three deepest blocks of a chain, Figure 5 on page 20 illustrates a "richer" view that contains forks and non-finalized blocks.

Chia Network 共有兩條鏈，一條是 Trunk (樹幹)，一條是 Foilage (樹葉)。Trunk 中包含有 proof of space 以及 VDF 的輸出，Foilage 中則有交易資訊。所有 proof of space 和 VDF 的挑戰都來自於前一個 Trunk 之中。

Group and Order

Group(群)是一個由集合以及二元運算組成的代數結構，舉例來說 (N, +)是最常見的群，叫做整數加法群。一個群必須滿足群公理 : 封閉性、結合律、單位元素、反元素。

封閉性:兩個群內的元素計算結果也會在群之中。

結合律: 對群內的元素 a, b, c, (a*b)*c = a*(b*c)

單位元素:存在 e 使得 a*e = e*a = a

反元素: 對每個 a 皆存在一個 b 使得 b*a = a*b = e

Order(階): 一個群的元素個數。

Wesolowski 和 Pietrzak 使用的都是整數模 n 乘法群，常被記做 $(\mathbb{Z}/n\mathbb{Z})*$ 或是Z_n^*。這個群內的元素就是所有小於等於 n 並且與 n 互質的整數。群的階可以用歐拉函數$\varphi(n)$表示，例如$\varphi(8) = 4$，因為 1, 3, 5, 7 與 8 互質。

對於任一個質數 p 來說，$\varphi(p) = p-1$。

若$n = p^k$，$\varphi(n) = \varphi(p^k) = p^k - p^{k-1}$， 除了 p 的倍數以外，其餘都跟 n 互質。

若 m, n 互質，$\varphi(mn) = \varphi(m)\varphi(n)$。

從以上兩個條件，我們可以對 N 做因式分解並計算他的 order

$$N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$$

$$\varphi(N) = \prod_{i=1}^{r} p_i^{k_i-1}(p_i-1)$$

RSW Time Lock Puzzle

VDF 的設計都是基於 Rivest, Shamir and Wagner(1996)的 time-lock puzzle 問題構造而成。Time-Lock Puzzle 的目的是加密一段文字使得這段文字在一定時間內都無法被解密完成。

Time Lock Puzzle 由 (N, x, T)構成，其中 N = p*q，T 是時間參數，且$x \in Z_n^*$。問題是去計算$y = x^{2^T} \bmod N$。 如果知道 N 的因式分解，亦即知道 p, q，就可以把上述問題分解成兩個小步驟。

1. $\varphi(N) = (p-1)(q-1)$
2. $e := 2^T \bmod \varphi(N)$
3. $y := x^e \bmod N$

雖然步驟比較多，但是在 T 非常大的時候($T \approx 2^{30}$)，可以節省很多時間，因為 e 會被限縮在 0 到 N 的範圍之間。如果不知道 N 的因式分解，那麼就只能乖乖的進行連續平方的運算。

以下舉個例子:

假設 x = 11, T = 8, N = 161=7*23:

依照下面表格的操作知道最後答案等於 95。

T	1	2	3	4	5	6	7	8

y	121	151	100	18	2	4	16	95

然而利用上面提到的兩次指數運算:

$$\varphi(161) = (7\text{-}1)(23\text{-}1) = 132$$

$$2^8 = 124\ mod\ 132$$

$$11^{124} = 95 \bmod 161$$

What is Verifiable Delay Function?

VDF 是一個需要消耗一定時間運算的函數，但是一旦被計算完成之後，利用其 output 以及附帶的證明可以很快的進行驗算。VDF 含有三個步驟的演算法:

$$\text{setup}(\lambda, \text{T}) \rightarrow pp$$

$$\text{Eval}(\text{pp}, \text{x}) \rightarrow (y, \pi)$$

$$\text{Verrify}(\text{pp}, \text{x}, \text{y}, \pi) \rightarrow \{accept, reject\}$$

其中

$\text{pp} := (\text{G}, \text{H}, \text{T})$ ，G 是一個未知階的有限群，H 是一個雜湊函數。

$\text{Eval}(\text{pp}, \text{x})$的過程如下:

計算 $\text{y} = \text{H}(\text{x})^{2^{\text{T}}} \in G$

計算證明π

輸出 (y, π)

計算證明也需要耗費時間導致實際計算時間增加到$(1 + \epsilon)\text{T}$，實作上$\text{T} = 2^{30}, \epsilon = 0.01$。

注意到這邊 Verifier 是不知道 G 的生成方式，因此也無法利用因式分解進行速算法，只能透過證明去驗證 Y 是否正確。

輸出 y 的正確性

為了讓這個問題更抽象化，先定義一些符號:

$$g := H(x) \in G$$

$$h := y = g^{2^T} \in G$$

$$T > 0$$

$$\iota_{exp} = \{(G, g, h, T) : h = g^{2^T} \in G\}$$

Wesolowski 論證

$$\text{Primes}(\lambda) = \textit{前} 2^{\lambda} \textit{個質數}$$

0. The verifier checks that $g, h \in \mathbb{G}$ and outputs *reject* if not,
1. The verifier sends to the prover a random prime ℓ sampled uniformly from $Primes(\lambda)$,
2. The prover computes $q, r \in \mathbb{Z}$ such that $2^T = q\ell + r$ with $0 \leq r < \ell$, and sends $\pi \leftarrow g^q$ to the verifier.
3. The verifier computes $r \leftarrow 2^T \bmod \ell$ and outputs *accept* if $\pi \in \mathbb{G}$ and $h = \pi^{\ell} g^r$ in $\mathbb{G}$

在證明過程中，prover 需要計算的部分是

$$q = \left[\frac{2^T}{l}\right]$$

$$r = 2^T - ql$$

$$\pi = g^q$$

Verifier 需要計算的部分則是

$$r \leftarrow 2^T mod\ l$$

$$h = \pi^l g^r \in G$$

注意到 prover 需要計算 q, r 比起 verifier 只需要考慮同餘項需要更多運算，加上 q 涉及到 2 的 T 次方，是一個非常大的數字。

Pietrzak 的論證

Pietrzak 提出的構造方式相對複雜很多，是一個遞迴的證明過程。

0. The verifier checks that $g, h \in \mathbb{G}$ and outputs *reject* if not,
1. If $T = 1$ the verifier checks that $h = g^2$ in $\mathbb{G}$, outputs *accept* or *reject*, and stops.
2. If $T > 1$ the prover and verifier do:
   (a) The prover computes $v \leftarrow g^{(2^{T/2})} \in \mathbb{G}$ and sends v to the verifier. The verifier checks that $v \in \mathbb{G}$ and outputs *reject* and stops, if not.
   Next, the prover needs to convince the verifier that $h = v^{(2^{T/2})}$ and $v = g^{(2^{T/2})}$, which proves that $h = g^{(2^T)}$. Because the same exponent is used in both equalities, they can be verified *simultaneously* by checking a random linear combination, namely that

$$v^r h = (g^r v)^{(2^{T/2})} \quad \text{for a random } r \text{ in } \{1, \ldots, 2^\lambda\}.$$

   The verifier and prover do so as follows.
   (b) The verifier sends to the prover a random r in $\{1, \ldots, 2^\lambda\}$.
   (c) Both the prover and verifier compute $g_1 \leftarrow g^r v$ and $h_1 \leftarrow v^r h$ in $\mathbb{G}$.
   (d) The prover and verifier recursively engage in an interactive proof that $(\mathbb{G}, g_1, h_1, T/2) \in \mathcal{L}_{\text{EXP}}$, namely that $h_1 = g_1^{(2^{T/2})}$ in $\mathbb{G}$.

Figure 1: Pietrzak's succinct argument for $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{EXP}}$

這裡第一個重點是 prover 必須告訴 verifier:

$$\mathrm{h} = \mathrm{v}^{2^{\mathrm{T}/2}} \; and \; v = g^{2^{T/2}} \rightarrow h = g^{2^T}$$

假設 T = 4 :

$$\mathrm{h} = \mathrm{g}^{2^4} = g^{16}$$

$$\mathrm{v}^{2^{\mathrm{T}/2}} = v^4 = (g^4)^4 = \mathrm{g}^{16}$$

Verifier 需要計算:

$$\mathrm{g}_1 \leftarrow g^r v \; and \; h_1 \leftarrow v^r h \in G$$

Prover 需要計算:

$$v \leftarrow g^{2^{\{\frac{T}{2}\}}} \in G$$

完成一次證明之後會產生新的一組 tuple

$$\left(G, g_1, h_1, \frac{T}{2}\right) \in \iota_{EXP}$$

這組 tuple 會在一次通過證明產生$\left(G, g_2, h_2, \frac{T}{4}\right)$，一直持續到 T = 1 為止。

我們也可以發現在每一次遞迴 prover 都需要計算 v 值:

$$v_1 = g^{2^{\frac{T}{2}}}$$

$$v2 = g_1^{2^{\frac{T}{4}}} = (g^{r_1} v_1)^{2^{\frac{T}{4}}} = \left(g^{2^{\frac{T}{4}}}\right)^{r_1} g^{2^{\frac{3T}{4}}}$$

$$v3 = g_2^{2^{\frac{T}{8}}} = \left(g_1^{r_2} v_2\right)^{2^{\frac{T}{8}}} = \left(g^{r_1 r_2} v_1^{r_2} v_2\right)^{2^{T/8}} = \left(g^{2^{\frac{T}{8}}}\right)^{r_1 r_2} \left(g^{2^{\frac{3T}{8}}}\right)^{r_1} \left(g^{2^{\frac{5T}{8}}}\right)^{r_2} \left(g^{2^{\frac{7T}{8}}}\right)$$

在計算 VDF 得到$h = g^{2^T}$的時候，VDF 會儲存2^d個元素 : $g^{2^{i*\frac{T}{2^d}}} \forall i = 0,1,2,\ldots 2^d - 1$。

兩個 VDF 構造的差異

Wesolowski 提供的 proof 更短，並且驗證這個 proof 也更快速。

Pietrzak 的構造有兩個好處:

Pietrzak 的優勢是生成 proof 的速度更快。

Pietrzak 的 VDF 使用 adaptive root assumption 來證明健全性，Wesolowski 使用 low order assumption。adaptive root assumption 比 low order assumption 更強，也就是說如果 adaptive root assumption 成立那 low order assumption 必定成立。

Reference

https://blog.priewienv.me/post/verifiable-delay-function-1/

https://www.chia.net/assets/ChiaGreenPaper.pdf

https://eprint.iacr.org/2018/712.pdf

https://eprint.iacr.org/2018/627.pdf