



Hashcash

A Denial of Service Counter-Measure

Adam Back 1st August 2002

2022/05/12

組員：吳冠廷、李威辰、張允瀚



Outline

- Hashcash的由來
 - 比特幣
 - 驗證碼
- Hash
- Hashcash
- Hashcash code
- Cost-function
- Dynamic throttling
- 應用
- 參考文獻



比特幣

比特幣白皮書第四章，有這樣一段話：「To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash.(要想實現點對點的分散式時間戳服務，我們就需要使用一種類似於Adam Back的Hashcash的工作量證明系統)。」

比特幣借鑒了許多技術，其中POW(proof-of-word工作量證明)借鑒了hashcash演算法。




驗證碼

驗證碼是一個用來確認操作者身分的驗證方式，通常會使用在高安全性的操作上，但在許多低安全性的操作也會見到，例如查詢、發文等。

雖然這些操作不需要高安全性，但系統不希望這些操作被濫用，像是機器人、爬蟲、DDoS等。

透過驗證碼可以增加每次操作的成本，雖然對操作者來說不太方便，但需花費的成本卻微乎其微。不過對機器來說，大量操作所造成的成本疊加卻是巨大的。

但每次使用都需要驗證還是非常討厭，若是能夠讓機器自己去驗證呢？



該如何讓機器驗證自己?透過付出一定的成本來表示自己的可信度。

若是機器願意付出成本就表示他真的想進行操作，而惡意機器則會因為付出成本過大則放棄進行操作。

如果讓機器在操作前進行一個「計算複雜，驗證簡單」的算式呢?



hash(雜湊)

雜湊函式可以把明文變成一串固定格式的亂碼，相同明文會出現相同雜湊值，然而即使只變更明文的一個字，所產生的雜湊值也會相差勝遠。

對於明文來說他的雜湊值是唯一的，且不可逆。



hashcash

在網路剛興起時，電子郵件是最多人使用的，但也因此造成許多垃圾郵件的出現。

因此Adam Back提出了hashcash，hashcash使用POW的技術來防止垃圾郵件。

當發送郵件前需要進行一些計算，對少量郵件來說計算量微小，但若是大量的郵件則計算量便十分巨大。

hashcash是一種應對DoS的工具，hashcash可以在發送的電子郵件中建立hashcash標記，並驗證收到的電子郵件的hashcash標記。



hashcash利用hash的特性，讓機器不斷嘗試數字來找出題目要求的雜湊值。

以比特幣來說，題目是雜湊值前幾位數為零。

為了防止機器重複使用數字，hashcash要求雜湊數字必須包含時間戳(timestamp)。

時間戳代表答案在何時被計算的，若過時則作廢。

若是透過hashcash讓http請求本身自帶機器驗證的資訊，則可以在協議本身的設計上避免類似DDoS的攻擊。



hashcash.py

參數介紹

- b : Specify required collision bits
- c : Mint a new stamp
- m : Check a stamp for validity
- s : Time the operation performed

```
def mint(resource, bits=20, now=None, ext='', saltchars=8, stamp_seconds=False):  
  
def _salt(1): ...  
  
def _mint(challenge, bits): ...  
  
def check(stamp, resource=None, bits=None, ...
```

▲ hashcash程式架構



mint

功能:Mint a new hashcash stamp for 'resource' with 'bits' of collision.

函式定義:mint(resource, bits=20, now=None, ext="", saltchars=8, stamp_seconds=False)

saltchars=8, This provides about 17 million salts per resource, per timestamp.

mint生成challenge(ver:bits:date:res:ext:rand:)後，_mint使用challenge以及bits來進行'generalized hashcash'，回傳counter。

最後生成stamp(ver:bits:date:res:ext:rand:counter)

mint會呼叫_mint以及_salt函式



`_mint`

Answer a 'generalized hashcash' challenge'.

counter初始值為0開始遞增，不斷對challenge+counter(字串)進行雜湊，來找出前X bits為0的雜湊值。

counter就是challenge的answer。



check

檢查stamp的ver、resource、bits、date是否正確，再對stamp進行雜湊檢查是否合法。



Cost-function

1. **publicly auditable** cost-function：任何的第三方裝置都可以有效的驗證此成本函數
2. **fixed cost** cost-function：使用固定的資源來計算，在計算固定成本的token時是最快的算法
3. **probabilistic cost** cost-function：client計算token的時間是可預測的，但由於client的起始值是隨機的，因此較幸運的client可能比計算的更快速
 - **unbounded probabilistic cost** cost-function：儘管花費的時間無限長的機率趨近於0，但沒有限制上限導致在理論上仍可能花費無限長的時間計算，例：擲硬幣一直為正面的機率非常低，但在理論上仍有機會達到
 - **bouded probabilistic cost** cost-function：限制了花費時間的上限，因此必定會在有限的時間計算出結果
4. **trapdoor-free**：使challenger可以廉價的創造任意價值的token，這可能會造成伺服器可能在審計時存在利益衝突。例：廣告投放商在投放廣告時是以網頁點擊數作為依據提供利潤，若使用後門則可以膨脹點擊數，並得到不正常的利益



Cost-function

$$([e = \{1, \frac{1}{2}, 0\}], [\sigma = \{0, \frac{1}{2}, 1\}], [\{i, \bar{i}\}], [\{a, \bar{a}\}], [\{t, \bar{t}\}], [\{p, \bar{p}\}])$$

i 表示成本函數是可互動的(interactive), \bar{i} 表示 \bar{i} 成本函數不可互動的(non-interactive)

a 表示成本函數是可被第三方驗證的(publicly auditable), \bar{a} 表示成本函數是不可被第三方驗證的(not publicly auditable)。

t 表示伺服器在計算成本函數時有後門(trapdoor), \bar{t} 表示伺服器在計算成本函數時沒有後門。

p 表示成本函數是可平行化的(parallelizable), \bar{p} 表示成本函數是不可平行化的(non-parallelizable)。



Cost-function

	Trapdoor-free	Trapdoor
Interactive	Hashcash $(e=1, \sigma=1, i, a, \bar{t}, p)$	Client-puzzles $(e=1, \sigma=1/2, i, a, t, p)$ Time-lock $(e=1/2, \sigma=0, i, \bar{a}, t, \bar{p})$
non-interactive	Hashcash $(e=1, \sigma=1, \bar{t}, a, \bar{t}, p)$	Time-lock $(e=1/2, \sigma=0, \bar{t}, \bar{a}, t, \bar{p})$



應用

- Hashcash設計了一個cost-function，在郵件標題中加入一個時間戳(stamp)。如果發送的是有意義的郵件，就需要花費一定的CPU時間來生成Hashcash戳(stamp)並發送，以此來證明發送的不是垃圾郵件
- 用來計算工作量證明機制
- 也被用來計算比特幣的獎勵機制



Dynamic throttling(動態限流)

使用交互hashcash可以根據伺服器的CPU負載進行動態調整客戶端所需的工作因子。

Dynamic throttling可以與舊客戶端兼容，並抵抗DoS攻擊。

在高負載期間，非hashcash客戶端會無法連線，或是被放置在有限的連線池中，受制於較舊、較差的DoS對策，像是random connection dropping。



參考文獻

[1]橙皮书(2018)。〈hashcash算法：从你最熟悉的“验证码”来解释区块链的意义〉， <https://mp.weixin.qq.com/s/5AEXWmkpZx4hZILI6-ODJg>

[2]張詠晴(2018)編譯。〈Hashcash究竟是什麼？比特幣竟是參考了這個機制〉， <https://news.knowing.asia/news/9032c2d8-9e61-40e9-a1a1-0aa1f0265f7b>

[3]Adam Back。〈hashcash - hashcash anti-spam / denial of service counter-measure tool〉， <http://www.hashcash.org/docs/hashcash.html>

[4]ITPUB博客(2007)。〈用Python的hashcash打击垃圾邮件(转)〉， <http://blog.itpub.net/8225414/viewspace-951087/>