國立臺灣師範大學理學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering College of Science

National Taiwan Normal University

Master's Thesis

在 TPS 有限的公有區塊鏈上實現具大量操作數目需求的 DPKI 系統

Implement a DPKI system with a large number of operations on a public blockchain with limited TPS

江宏文

Jiang, Hung-Wen

指導教授： 黃冠寰 博士

中華民國 109 年 8 月

August 2020

誌謝

碩士生涯猶如白駒過隙，在師大的日子已經到了第二年的尾聲，在這些時光中，我學習到了許多大學不曾接觸過的新事物，其中非常感謝黃冠寰教授的教導，在跟隨教授的這些日子裡，除了課業與學業方面的成長，在做事情的態度、邏輯思維的角度、解決問題的方法，都深深受到教授的啟發，讓我在人生的道路上更為順遂。

除了老師的教導外，實驗室的學長、同學們也帶給我很多的幫助，碩一時修了很多課程，有時上課內容過於難懂或是作業遇到瓶頸，學長都會將自身的理解與經驗告訴我，讓我解決難關。而撰寫論文或看論文時，也會跟同學們互相辯駁、討論，讓我的思維在這種激烈的碰撞下，產生許多原本料想不到的靈感與解答。

最後再次感謝我的指導教授黃冠寰教授，老師曾說過一句話「如果你真的懂一個道理，你一定可以用簡單的方式告訴別人」這句話讓我印象非常深刻，因為這句話，讓我常常在作事情的時候，都會思考要如何簡單的傳達給別人，並以此進行反思，也再次感謝我的同學們，因為有你們，讓我的碩士生活過得非常充實、快樂，謝謝。

江宏文　誌於

國立臺灣師範大學資訊工程所

民國　109 年 6 月

摘要

PKI 架構是目前被廣泛使用的網路身分驗證架構，使用者會向憑證認證機構註冊身分資訊以取得數位憑證，再將數位憑證展示給他人當作身分證明，他人看到憑證後，必須向 CA 取得撤銷名單才能確認憑證是否有效。雖然這個架構已經在網路中使用 10 年以上，但他安全性其實有很大的疑慮，因為 CA 可能會受到 DDoS、DNS 攻擊，導致檢驗者無法索取撤銷名單，造成身分驗證機制完全失去效用。目前以有許多可以改善 CA 單點故障的新型 PKI 架構，但是它們本身或多或少還有其他缺陷，導致到目前為止大家還是使用最原版的架構。

為此，我們提出了半去中心化的 PKI 架構，透過區塊鏈可以輕易的避開單點故障問題，並且融入的自動賠償機制，使用者透過特定協議，取得密碼學證據，釐清憑證錯誤的責任歸屬，再將證據交由智能合約進行自動判決以及賠償，可以免除現實中，使用者與 CA 不同國家時，申訴、客服可能遇到的困難。

關鍵字：公開金鑰基礎設施、PKI、DPKI、證明違約、智能合約、區塊鏈、自動賠償

目錄

附表目錄

附圖目錄

第一章 緒論

第一節 Public Key Infrastructure

公開金鑰基礎建設(Public Key Infrastructure，PKI)，是一種由密碼學原理、擁有者、管理者三方組成的基礎架構，其旨在於數位憑證(Digital Certificate)的相關應用，如建立及發放憑證、管理使用、復原撤銷等，而數位憑證可以用來驗證使用者身分，是網路上目前所使用的身分驗證方法，常見的 PKI 應用如企業(網站、IoT)、個人(報稅、網銀、e-mail)。

任何人都可向憑證認證機構(Certificate Authority，CA)註冊自己的公開金鑰與個人相關資訊，以取得一張經過憑證認證機構簽名的數位憑證，之後在網路通訊時可以藉由展示此數位憑證，檢驗者會根據憑證的有效性，來決定是否信任憑證的擁有者。

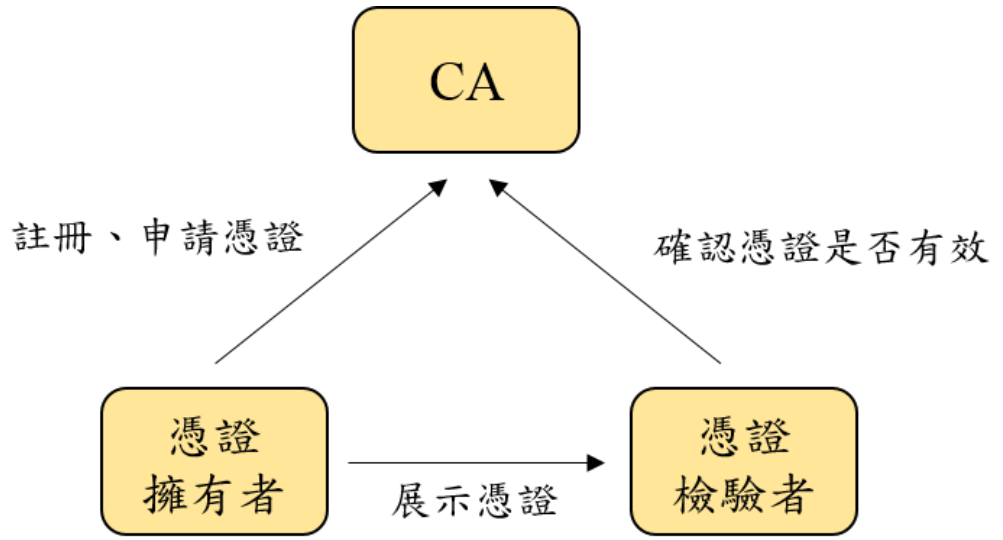

圖 1 PKI 基礎架構

憑證有效性除了透過驗證 CA 簽名外，還需要向 CA 確定該憑證是否被撤銷，憑證撤銷方式有兩種，第一種「CRL[1]」是 CA 會定期發布撤銷名單，而檢驗者需要先下載完整的撤銷名單，再逐一比對要檢驗的憑證是否再名單內。第二種「OCSP[2]」則是 CA 會建立憑證狀態資料庫，檢驗者只需要向 CA 提供憑證資訊，CA 就會根據資料庫回傳該憑證的最新狀態給檢驗者，目前憑證狀態有「good」、「revoked」、「unknown」三種。

第二節 Digital Certificate

數位憑證是用於 PKI 架構的電子資料，可以用來證明公開金鑰(Public Key)的擁有者身份，其標準格式為 X.509[3]。數位憑證會儲存擁有者身分資訊、公鑰以及 CA 對該憑證所簽屬的數位簽章(Digital Signature)，該簽章可以確保數位憑證的儲存內容正確無誤。擁有者憑著此憑證，可向其他網路上的服務商或使用者證明身份，來獲得對方的信任，以致使用某些需要身份證明的網路服務，而收到憑證的網路服務商或使用者，會向 CA 詢問該憑證是否過期、數位簽章是否有效等資訊，來核實憑證的正確性。其中最主要的功用是在檢查擁有者身分時，擁有者不需傳送個人相關敏感資訊（如電話、身分證、住址等）給驗證者。透過這種資料交換方式，擁有者既可證明身分，也不必公開私人資料，對個人隱私有極大的保障。

第三節 Public Key Infrastructure 的問題

PKI 雖然在網際網路上使用已久,但其本身有不少問題陸陸續續被人提出[4]。在架構方面，因運作全仰賴 CA 負責，故惡意攻擊者常會透過癱瘓或是駭入 CA 等方式，來攻擊憑證的使用者，例如:2011 年 Diginotar 事件[5]，攻擊者駭入 CA 後頒發一系列假憑證。

在流程方面又分為頒發跟撤銷兩部分，首先頒發憑證的部分，很多 CA 發證門檻低，只要你付錢就可以取得，這代表惡意使用者會在前期偽裝成正常網站或服務，潛伏一段時間後才轉變為釣魚、病毒網站等進行惡意攻擊[6]。接著在撤銷部分，CA 具有單點故障問題，這個「單點」並不是指某一設備，而是指 CA 本身，因為使用者欲檢查憑證有無撤銷需向 CA 進行連線確認，但使用者有可能因為各種問題(例如：CA 被 DoS 攻擊、DNS 被串改)，導致無法順利向 CA 整體取得資訊；除此之外，CA 為中心化架構，頒發及撤銷憑證通常是不透明的，雖然 CA 擁有撤銷憑證的權利，但使用者若被惡意/意外撤銷憑證，會「很難」或「需要很長的一段時間」才能進行申訴，因為使用者無法取得實質的證據證明自己的清白，再者，網路服務無遠弗屆，使用者與 CA 不一定會在相同的地域上，聯繫客服往往需要大量時間進行轉接、翻譯。

第四節 Decentralized Public Key Infrastructure

為了解決中心化帶來的種種問題，開始有人研究「去中心化公開金鑰基礎建設(Decentralized Public Key Infrastructure，DPKI)」，從發現 PKI 問題開始到現在，已有不少成功的案例或實作，目前可分成兩大類別，其一為採用信任網路(Web of trust)為基礎，如 PGP[7]，它移除 CA 的部分，憑證改由使用者自己頒發，透過取得其他使用者簽名(信任)來增加自己憑證的可信度，雖然解決了 CA 單點故障的問題，但可信度極度仰賴受他人簽名數量，導致新加入的成員難已受到大家信任，也無法快速融入網路內；另一種則是使用近幾年蓬勃發展的區塊鏈技術[8]來完成，又稱為 Blockchain-based PKI，因為區塊鏈具有資料公開及不可串改的特性，拿來存放、管理憑證擁有很高的透明性及安全性，並且其底層為分散式系統，可以避免單點故障問題，詳細區塊鏈細節會在後面章節說明。

目前 Blockchain-based PKI 有全去中心化及半去中心化兩種，前者如 Mustafa Al-Bassam 所做[9]，其結合信任網路(Web of trust)與區塊鏈技術，讓使用者將憑證本身以及它人簽名儲存到區塊鏈上，與 PGP 相似只是儲存憑證位置不同而已；後者有 Karen Lewison and Francisco Corella 所做[10]，其保留了 CA 的部分，並讓 CA 將憑證資訊、撤銷資訊均放入區塊鏈，使用者只需要到區塊鏈查看憑證資訊，不必與 CA 通訊，也不用像[9]一樣需要累積一段時間才能正常使用。

第五節 Blockchain-based PKI 的問題

儘管半去中心化架構看起來很完美，但解決舊問題的同時，新問題也隨之產生，導致 Blockchain-based PKI 始終無法取代目前的 PKI 成為主流，主要問題主要有三點，詳情如下：

問題一 區塊鏈頻寬限制

塊鏈交易速度非常慢，以區塊鏈技術中的以太坊(ethereum)[11]為例，其 TPS(每秒交易量)只有 10，一天最多 86 萬多筆交易，但光是一家免費的憑證機構 let's encrypt，一天所要頒發的憑證數量就大約 100 萬張[12]，區塊鏈完全無法負荷如此龐大的憑證發放。

問題二 缺乏監督機制

將憑證儲存在分散式系統的區塊鏈中，雖然解決檢驗憑證時的單點故障問題，但是在發放與撤銷的動作流程上，其實還是不透明的中心化情形，坦白來說 CA 只是換了一個儲存憑證資訊的位置而已，使用者還是無法取得有用的證據，來證明 CA 放入區塊鏈的憑證資訊是否有問題。

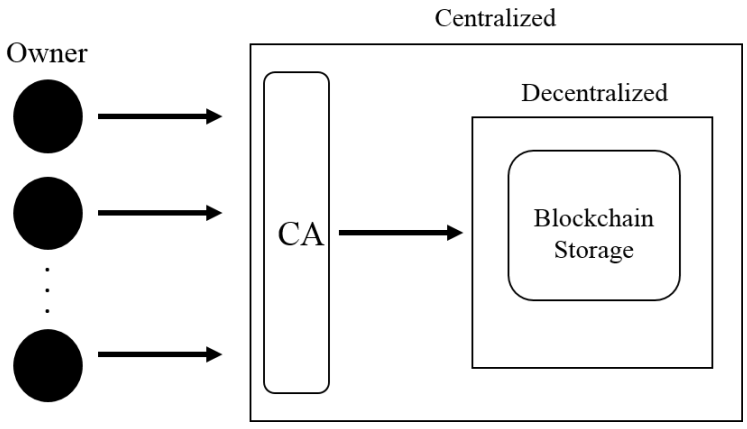

圖 2 Blockchain-based PKI 的中心化部分示意圖

問題三 牴觸 GDPR

歐盟於 2018 年實施一般資料保護規範(General Data Protection Regulation，GDPR)[13]，其目的為加強歐盟境內人民電子資料保護(personal data)的保護，法規內容不僅重新定義個人資料(personal data)的範疇，對資料的使用與儲存也有更多嚴格的限制，且不論企業或服務提供者是否在歐盟境內，只要使用資料或服務對象與歐盟人民有關，均需符合 GDPR 之規範，若違反 GDPR 將被處以高額罰鍰，然而 GDPR 內有兩項新法規，與區塊鏈技術的特性互相牴觸，導致區塊鏈必定無法符合 GDPR：

其一，個人資料儲存時必須假名化(pseudonmisation)，意即若不使用額外資訊，旁人是無法從儲存的資料逆推當事人，且額外資料必須另外存放，然而[9][10]兩篇論文，直接將憑證放入區塊鏈的做法，等於直接將憑證擁有者的個人資訊以明文方式公開到網路上。

其二，資料的當事人可以行使「被遺忘權」，資料擁有者可要求控制資料的一方，刪除所有個人資料的任何連結、副本或複製品，但很不巧的，區塊鏈具有不可刪除的特性，資料一旦放上去，除非所有儲存節點損毀(幾乎不可能)，因此憑證的擁有者永遠無法行駛被遺忘權。

第六節 目標&解決方法

PKI 與 DPKI 雖然都是致力於解決身分驗證，但或多或少都有一些缺陷，因此我們目標是融合兩方之長，建立一個更為安全、方便的身分驗證架構，由於 PKI 最大的問題在於單點故障以及流程不透明，而區塊鏈技術正好是 PKI 問題的最佳解法，因此本篇論文將著重於解決 Blockchain-based PKI 的問題。

為了解決區塊鏈本身頻寬限制以及抵觸 GDPR 部分，我們研究了新的資料結構「TP-Merkle Tree[14]」來進行憑證的儲存及驗證，透過此結構可以將一百萬張的憑證壓縮成一筆資料再儲存到區塊鏈上，完美的解決了假名化以及交易數量限制，至於運作不透明的部分，我們建立了一套包含釐清責任、快速申訴、自動賠償的完整監督機制，透過 PoV 技術取得有效證據的情況下，可以大幅度降低 CA 出錯或作詭的機率，即使發生問題，使用者只需要使用證據，即可輕易從區塊鏈上得到相應的加密貨幣[15]當做賠償，而不必考慮跨國客服時貨幣、時差及語言不同等問題。

第二章 去中心化架構

第一節 Blockchain & 加密貨幣

區塊鏈是近年興起的記帳技術，由一堆節點共同儲存、管理所有交易的歷史記錄，公開透明、安全為其特色，也可被稱為分散式帳本。

它使用了 P2P 架構讓所有人都能輕易成為帳本擁有者(節點)，接著使用許多密碼學原理如:加密雜湊、數位簽章等，讓人難以偽造、改變其它節點的帳本內容，之後再透過「共識機制」讓所有節點進行公平的獎勵競爭，最後根據競爭結果來統一所有節點的帳本內容，假如有攻擊者想要攻擊區塊鏈，那他至少要掌握 51%以上的節點才有可能改變投票結果，而這種攻擊方式又被稱為 51%攻擊，但是以目前的技術來看是幾乎不可能達成，因此區塊鏈帳本可以說是不可能被串改的，故資料被記錄到區塊鏈後，除非所有節點都不存在，否則會永遠的保存下去。

加密貨幣是運用在區塊鏈上的交易媒介，其所有的交易情形都會記錄到區塊鏈，所以它的安全性非常高，且相較於實體貨幣，在國際間使用不易受到各國法規或政策的影響，擁有者也可以隨時隨地向任何節點查看帳本，因此加密貨幣是不受時間、空間影響的交易貨幣，在跨國傳輸上極為便利，目前常用的加密貨幣有比特幣(BTC)[16]、以太幣(ETH)[17]、萊特幣(LTC)[18]。

第二節 Ethereum & Smart Contract

以太坊(Ethereum)是一個開源的公共區塊鏈平台，專用加密貨幣為以太幣，其擁有獨特的智能合約平台，該平台讓使用者可以撰寫及發佈程式到區塊鏈上儲存，需要執行程式的人只需支付「手續費」給節點就可以執行程式。而其可在區塊鏈上執行程式的特色，使之被稱為「第二世代區塊鏈」。

以太坊的交易速度，約為每秒 10 筆交易，使用工作量證明(Proof-of-Work)作為共識機制，工作量證明的雜湊值(hash)採用 sha-3 進行計算，其具有運算困難，但驗證極為快速簡單的特色，但 PoW 也因計算困難的關係使之在進行共識時會消耗非常大量的電力，且消耗這些電力就只是為了取得共識，從旁觀者的角度來看這是一種極大的資源浪費，為了改善這個問題，以太坊正在努力將共識機制從 PoW 轉為權益證明(PoS)。

一個建立在以太坊上的基本功能被稱為智能合約，其跟一般的程式語言一樣可以使用 if…else…等條件句，來完成各種不同的功能，且因區塊鏈公開透明的特性，智能合約的程式碼都是開源可讀的，目前已有許多使用智能合約來開發應用程式，小至簡單的資料統計，大至複雜的電玩遊戲，而這些程式被稱為去中心化應用程式(DApp)，本篇論文正是透過建立 DApp，處理監督機制裡使用者自動賠償的部分。

第三章 基於區塊鏈的 PKI 容錯架構

第一節 系統架構

具有容錯機制的 Blockchain-based PKI，是一種不同以往 PKI 架構，在系統流程上比傳統 PKI 多了稽核、申訴兩步驟，而憑證的撤銷、儲存、檢驗也採用不同以往的方式進行，為了方便理解，此論文使用網頁上 PKI 應用為基礎，具有容錯機制的系統架構如下圖所示。

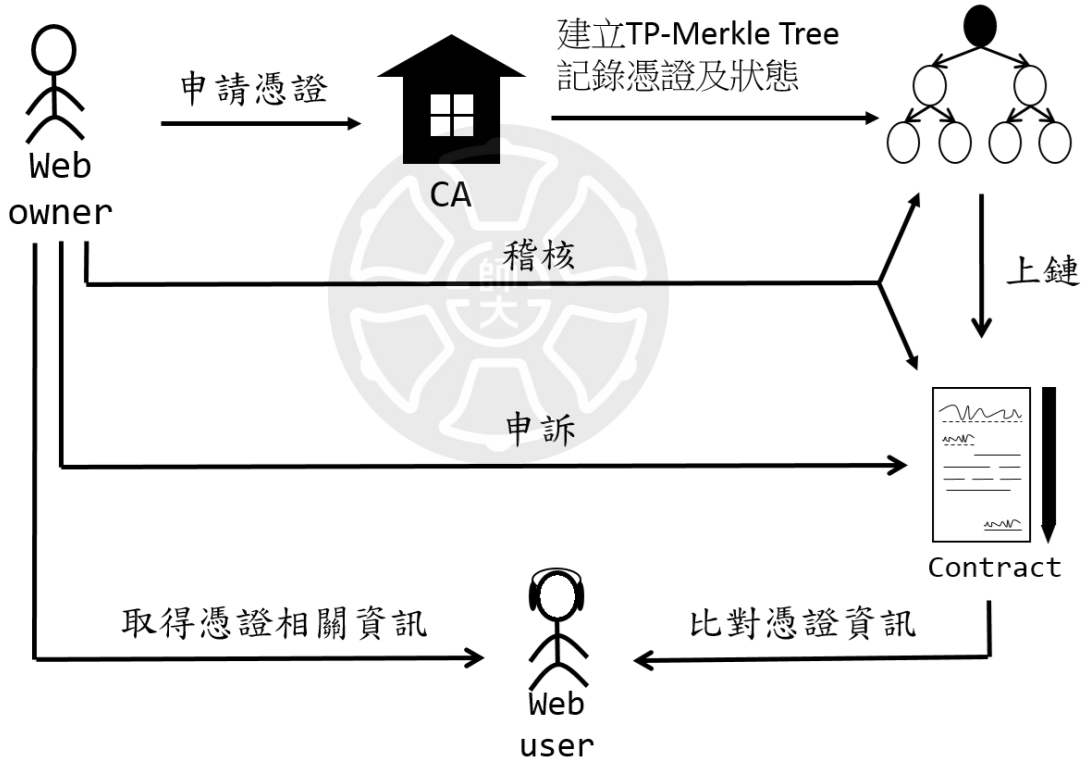

圖 3 以網頁 PKI 應用為基礎的容錯架構示意圖

從上圖可以得知角色之間的關係，至於角色定義將於下說明。

(1) Web Owner：網頁憑證的擁有者，會將憑證公告給一般 user 檢驗，同時也是架構中進行流程稽核、錯誤申訴的角色，之後以 Owner 代稱。

(2) CA：憑證機構，負責處理 Owner 對憑證相關請求，如憑證申請、撤銷憑證、更換憑證資訊等，也是負則將憑證資訊上傳至區塊鏈上，屬於架構中管理者的角色。

(3) Web User：瀏覽網頁的一般民眾，只負責檢驗憑證是否有效，在架構中為驗證者的角色，之後以 User 代稱。

(4) Contract：CA 發布智能合約，負責處理憑證儲存、自動申訴裁決和自動賠償的工具。

第二節 系統流程

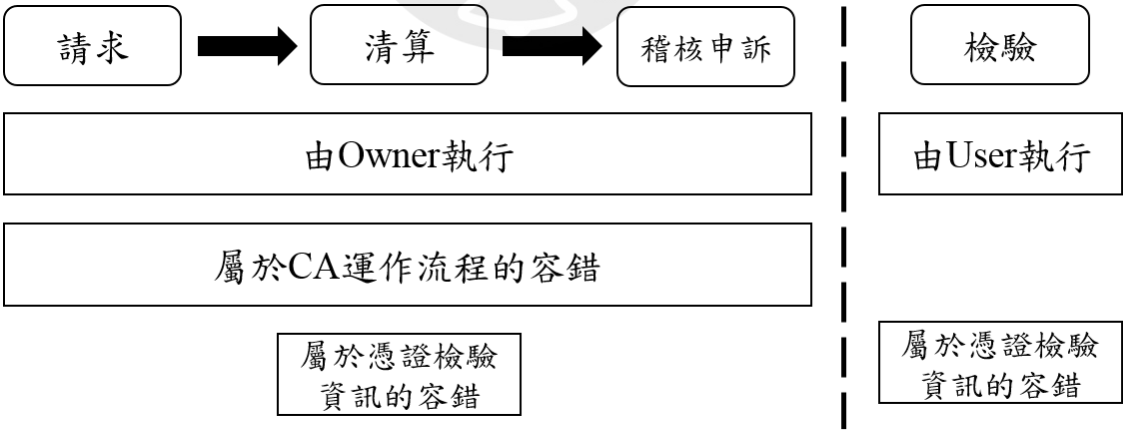

圖 4 系統流程功能說明

系統流程主要可以分成 4 個階段—「請求」、「清算」、「稽核和申訴」以及「檢驗」，其中，「請求」、「清算」與「稽核&申訴」三者合一可以達到對 CA 運作流程的檢查，我們又將這三個階段合稱為「監督機制」，後面章節會詳細描述

系統是如何達到監督的效果；至於最後的「檢驗」階段，採用較為特別的分層式的驗證方式，除了降低 CA 的負擔外，User 還可以辨別其取得的撤銷名單或其他驗證資訊是否正確，就算發現錯誤也能夠透過其他管道取得正確的訊息，可以說是提供憑證驗證資訊的容錯性。

另外，為了讓申訴更容易，我們還將憑證狀態由原本的三種改為四種，其名稱與含意表示如下：

1. Add　　：憑證有效，代表新加入的憑證
2. Renew　：憑證有效，代表已續約 或是 結束 Pause 狀態
3. Pause　　：憑證無效，代表憑證目前有問題，有機會回到有效狀態
4. Revoked　：憑證無效，代表憑證已被永久撤銷

第三節 TP-Merkle Tree

憑證的撤銷、檢驗方式，與以往 PKI 架構不同，其實這與憑證的儲存方式有關，以往的 PKI 架構，憑證驗證資訊都是獨立分開的，不論是儲存空間還是驗證資訊，都屬於一對一的概念，但為了解決區塊鏈頻寬問題，我們使用了 TP-Merkle Tree 這個資料結構來進行儲存，這個資料結構可以將數百萬張的憑證壓縮成一個 32 bytes 的資料，變成了一對多的概念，因此以往的驗證方式是絕對無法使用的，因此在介紹撤銷、驗證是如何進行之前，我們必須先了解 TP-Merkle Tree 是怎麼

對運作的。

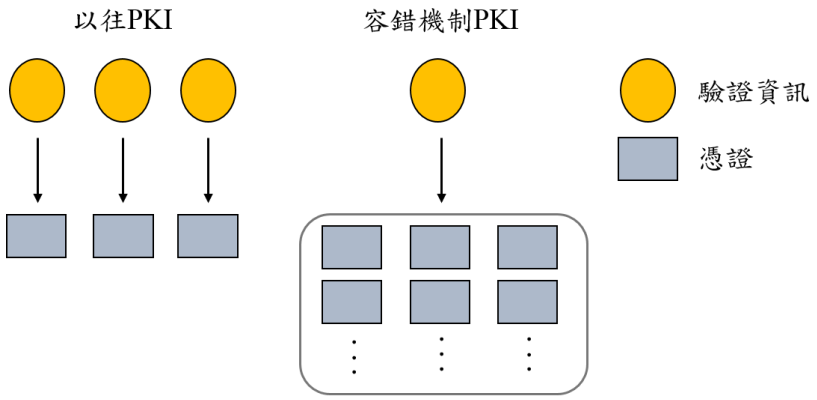

圖 5 憑證儲存的差異

第一段 Merkle Tree

Merkle Tree 又被稱為哈希樹、雜湊樹，因為它是使用雜湊函數建立而成，是一種樹狀資料結構，通常以完滿二元樹的方式進行實作，樹的有三種節點 leaf node、internal node、root node，每個節點都是一個唯一雜湊值，建立樹時會先計算每個 leaf node 的個別雜湊值，接著將 internal node 的左右節點雜湊值相加在計算一次雜湊，就這樣從下往上逐一計算所有節點，直到算出 root hash。

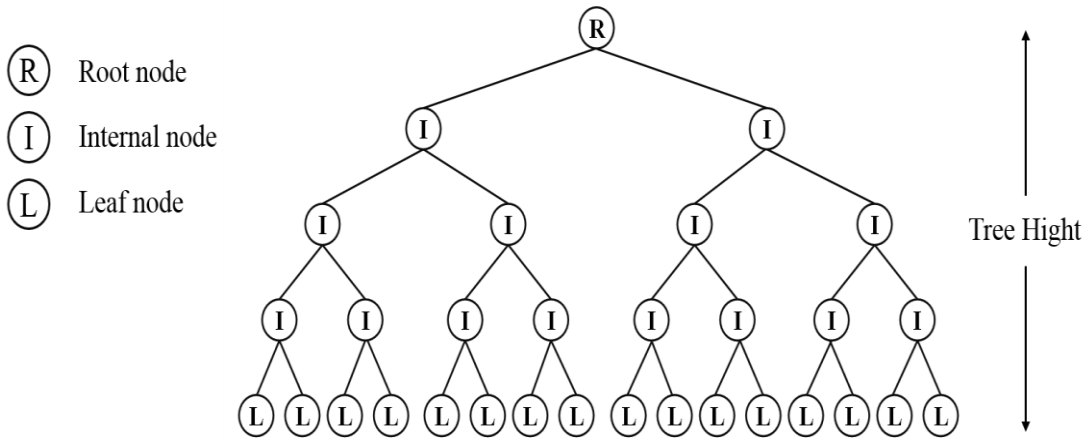

圖 6 Merkle Tree 結構

雜湊值有一個特性，就是輸入的資料只要不變，不管計算幾次都是相同的結果，所以常常被拿來當作驗證資料的完整性與正確性，這代表使用 Merkle Tree 來儲存資料，可以透過驗證 Root hash 來證明資料是否有誤。

第二段 Index Function

TP-Merkle Tree 就是以上述為基礎，加入了定位功能，並且會事先決定好樹高才開始放資料，leaf node 可以放複數資料進去，但每筆資料必須以 Key-Value pair 的方式儲存，例如：(Key1, Value1)，key 值又稱作 indexValue 是定位用的參數，而 value 則是實際要存放的 data，至於 leaf node 底下所有的 Key-Value pair 我們稱之為 list of Key-Value pairs，要定位時只需將 indexValue 放入 Index Function Γ函式進行的計算，即可得到指定的 leaf node 位置，Index Function Γ函式如下：

$$\Gamma(IndexValue) = SHA256(IndexValue)\ mod\ 2^{N-1}$$

假設樹高為 N，而2^{N-1}則是計算樹的總節點數，透過 mod 的動作就可以限制定位不會超過樹的節點總數。

透過這種方式放入資料後，任何人都可以透過 indexValue 來找尋相應的資料。

第三段 Slice

Slice 是整個 TP-Merkle Tree 驗證方法中的重要證據，每一個 leaf node 都會對應一條 Slice，其形式如下：

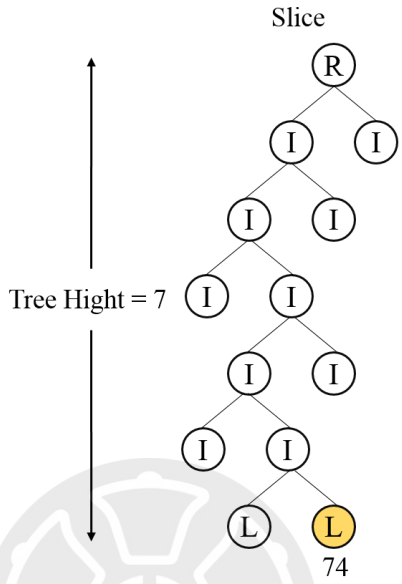

圖 7 leaf node 74 為例的 Slice 結構

Slice of leaf node 74 = Root node | Left Internal node | Right internal node |…| Left Leaf node | Right Leaf node，Slice 上包含自己，以及所有的兄弟節點、父節點的雜湊值，驗證資料時只要由下往上計算一次 Slice，在與以前記錄 Root hash 做比較就可以知道資料是否正確，對驗證者來說，不需要儲存所有的 hash 值，只要取得自己資料所在的 Slice 即可驗證。

透過上述兩個方式，雖然我們將一百萬張憑證都放到 Tree 裡面，但憑證擁有者可以透過 Index Function 來確認自己的憑證所在位置，知道位置後再取得相應的 Slice 即可進行稽核。

第四節 請求協議

傳統 PKI 的 Owner 無法監督 CA 最大的問題在於無法取得有效證據，為此，我們研究了違約證明協定(Proof of Violation, PoV)，在[19、20]兩篇論文中，PoV 原本是應用在雲端服務商與雲端服務使用者之間釐清責任歸屬的一套方法，透過訂定嚴格的通訊協議以及密碼學原理，讓服務商與使用者雙方在每次互動後都會產生一個密碼學證據(cryptographic proof)，並且這個證據會經過雙方簽章，發生問題時能以此判定責任的規屬。雲端服務商與雲端使用者之間的關系，剛好跟 CA 與 Owner 之間的關系雷同，但原本的 PoV 協議並不適合應用在 PKI 上，因此我們需要重新定義協議內容。

首先，一個完整的互動過程會產生兩個訊息：請求($\boldsymbol{M_{request}}$)、回條 ($\boldsymbol{M_{reply}}$)，其基本訊息格式定義如下：

$\boldsymbol{M_{request}}$ = (Operation | PK | IndexValue | CO) $_{\text{Sign\_Owner}}$

$\boldsymbol{M_{reply}}$ = (H($\boldsymbol{M_{request}}$) | result | H(Certificate) | Status) $_{\text{Sign\_CA}}$

欄位說明，Operation 代表要進行的服務項目；PK 為 Public Key 代表 Owner 的 PKI 公鑰；IndexValue 為憑證索引值，會放入Γ函式進行定位；CO 為 Clearence Order 的簡稱，代表 TP-Merkle Tree 的清算編號，是申訴的重要證據之一，後面章節會詳細說明；H(Certificate)為憑證的雜湊值，代表此次互動要使用的憑證；Status 代表憑證狀態。

接著根據 Owner 對憑證相關需求，將基礎訊息擴充為三種不同請求協議，協

議中包含完整格式、互動的步驟，詳情如下所示。

第一段 申請憑證(Apply Certificate)

第一步：Owner 向 CA 發送一個 Apply Certificate 請求，簡稱$ACM_{request}$，格式如下。

$ACM_{request}$ = (Apply | PK_{owner} | Web IP | IndexValue | CO) $_{Sign\_Owner}$

* Web IP 代表要申請憑證的網站位址

第二步：CA 檢查$ACM_{request}$簽名是否正確

第三步：CA 檢查 Web IP 是否已註冊，若無註冊則產生新的數位憑證。

第四步：CA 使用 IndexValue 與Γ函式進行定位，再將 IndexValue 當作 Key、憑證的雜湊值與狀態當作 Value，儲存到剛剛計算的 leaf node，格式如下。

Key-Value pair = (IndexValue, H(Certificate) | Add) $_{Sign\_CA}$

第五步：CA 依據執行結果產生相應回條ACM_{reply}，之後將回條以及憑證經過簽名後回傳給 Owner，格式如下。

1. ACM_{reply} = (H($ACM_{request}$) | H(Certificate) | Add) $_{Sign\_CA}$

2. (Certificate)$_{Sign\_CA}$

第六步：Owner 檢查ACM_{reply}簽名是否正確。

第七步：Owner 保存ACM_{reply}，以用作之後進行申訴的證據。

第二段 變更狀態(Change Status)

此協定為 Owner 進行憑證續約，或是想要暫停或撤銷憑證時使用，不過需要注意的是，若要撤銷憑證，必須要先進入暫停狀態，代表 Owner 必須執行兩次 Change Status 的請求，例如 Add → Pause → Revoked 或 Renew → Pause → Revoked。

第一步：Owner 向 CA 發送一個 Change Status 請求，簡稱$CSM_{request}$，格式如下：

$CSM_{request}$ = (Change | Status |PK_{owner}| hash(certificate) | IndexValue | CO) $_{\text{Sign\_Owner}}$

第二步：CA 檢查$CSM_{request}$簽名是否正確。

第三步：CA 使用 IndexValue 計算定位，再到樹上相應位置檢查憑證及狀態是否存在、正確。

第四步：CA 依據 Owner 填入得 Status，改變該憑證在 TP-Merkle Tree 上儲存的狀態。

第五步：CA 依據執行結果產生相應回條CSM_{reply}，之後將回條經過簽名後

回傳給 Owner，格式如下。

CSM_{reply} = (H($CSM_{request}$) | Status) $_{Sign\_CA}$

第六步：Owner 檢查$CSM_{request}$簽名是否正確

第七步：Owner 保存此回條，以用作之後進行申訴的證據。

第三段 更換金鑰(Replace Key)

若 Owner 懷疑私鑰洩漏、或其他安全疑慮，可向 CA 提出更換 Public Key 的請求。

第一步：Owner 向 CA 發送 Replace Key 請求，簡稱$RKM_{request}$，格式如下。

$RKM_{request}$ = ((Replace | PK_{owner_old} | PK_{owner_new}| hash(certificate) $_{old}$ | IndexValue | CO) $_{Sign\_Owner\_old}$)$_{Sign\_Owner\_old}$

* PK_{owner_old}代表原本的 PK，PK_{owner_new}代表要替換的 PK，$RKM_{request}$必需要使用新、舊兩把 PK 都簽名過。

第二步：CA 檢查兩個簽名是否正確。

第三步：CA 使用 IndexValue 計算定位，再到樹上相應位置檢查憑證及狀態是否存在、正確。

第四步：CA 使用新的 PK 建立新的數位憑證(certificate $_{new}$)

第五步：CA 更新原本 TP-Merkle Tree 記錄的憑證雜湊值及狀態

第六步：CA 依據執行結果產生相應回條RKM_{reply}，之後將回條及新的憑證經過簽名後回傳給 Owner，格式如下。

1. RKM_{reply} = (H($RKM_{request}$) | result | hash(certificate) $_{new}$ | Renew)$_{Sign\_CA}$
2. (Certificate $_{new}$)$_{Sign\_CA}$

第七步：Owner 檢查$RKM_{request}$簽名是否正確。

第八步：Owner 保存此回條，以用作之後進行申訴的證據。

第五節 清算

在請求階段，CA 會蒐集到許多憑證資訊，並將它們全部放到 TP-Merkle Tree 上，這時的憑證資訊都還只存在 CA 本身的 server 內，而在 CA 蒐集到一定數量的憑證，或是經過一段時間之後，CA 會計算目前 TP-Merkle Tree 的 Root hash，並將 Root hash 上傳至智能合約進行儲存，之後 Owner 就可以向 CA 索取 Slice 等驗證資訊，進行下一階段的稽核動作，最後 CA 會將整棵樹上傳到公共平台供所有人下載。

第一段 清算流程

第一步：CA 計算目前 TP-Merkle Tree 的 Root hash。

第二步：上傳 CO、Root hash 到 Contract 進行儲存。

第三步：Owner 可以使用自己的 indexValue、CO 向 CA 索取

Slice、list of key-value pairs，其格式如下。

(CO | Slice | List of key-value pairs) $_{Sign\_CA}$

第四步：CA 公告整棵 TP-Merkle Tree 到 IPFS。

第二段 清算規則

清算時間並沒有強制規定，本篇論文考量憑證檢驗具有時效性，因此實驗時以一天為單位進行清算，而清算完畢後，並「**不會清除**」原本的 TP-Merkle Tree，而是持續在這顆樹上作 leaf node 的新增或修改，這也就代表，憑證一但放上去，就會永遠存在，頂多會因為 Owner 的請求而產生儲存狀態變化而已，因此，每日清算所得的 Roothash 以及公告的 TP-Merkle Tree，可視為 TP-Merkle Tree 該日的「Snapshot」，Owner 可以透過 CO 及 Roothash 來確認不同時期下 Tree 的狀態，這也代表，Owner 可以對任何時期的 CA 提出申訴。

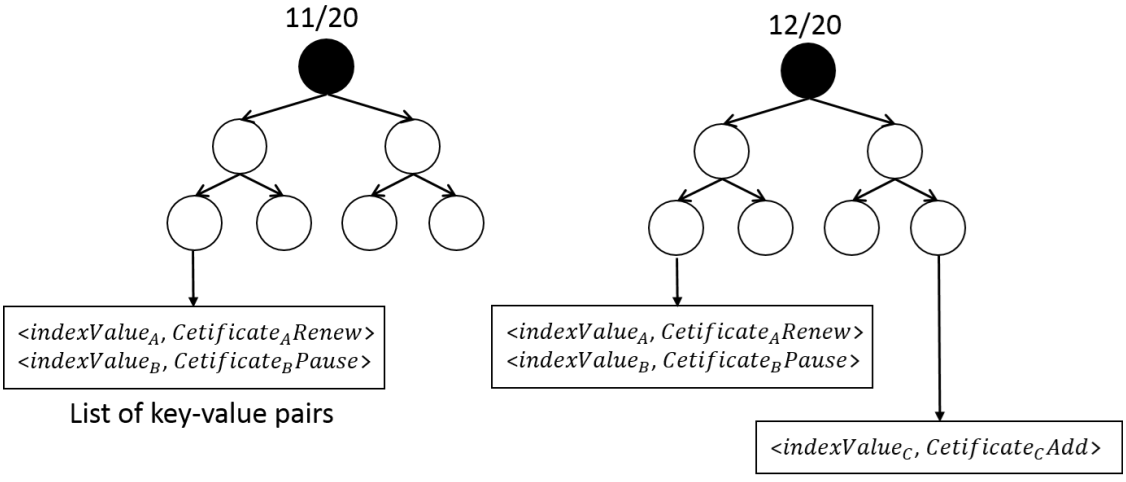

圖 8 Tree 變化示意圖

IPFS[21]是星際檔案系統的簡稱，它是一種分散式檔案儲存系統，被我們拿來當作布告欄使用，因為它是 P2P 架構，比較不用擔心資料會遺失等問題，又可以輕易讓任何人下載，可被其他 P2P 的檔案系統作替代，如：BitTorrent。

第六節 稽核

第一段 證據稽核

在請求與清算階段，Owner 取得了$M_{request}$、list of key-value pairs、Slice，這些東西我們合稱為**密碼學證據**，Owner 可以透過檢查這些證據的正確性，來辨別 CA 操作是否正確，詳細步驟如下。

第一步：Owner 比對 CA 給的 CO、Slice 的 Root hash 是否與 Contract 上一致

第二步：Owner 計算 Slice、list of key-value pairs 是否等於 Root hash

第三步：檢查 list of key-value pairs 內，自己的憑證狀態是否與M_{reply}記錄的一致。

簡單說明一下稽核原理，第一步驟檢驗 Root hash，可以確認 CA 給的 Slice 是否屬於同一天的 Tree，第二步驟則是確認 Slice 是否正確，若前兩步驟都沒問題，則表示此樹為正確的資料，Owner 可以相信 tree 所記錄的憑證狀態，第三步驟檢查憑證狀態及M_{reply}，可以讓 Owner 知道 CA 是否有按照自己的請求，對憑證作出正確改變，因為M_{reply}經過 CA 的簽名，假如 Tree 的狀態與M_{reply}不同，代表 CA 沒有實際完成 Owner 的請求。

第二段 錯誤情形

Owenr 可透過稽核發現的錯誤，大致上可以分成兩個部分，第一種是密碼學證據本身錯誤；第二種則是 CA 運作流程錯誤，而每種分類底下又有許多不同的情形，大致上如下圖。

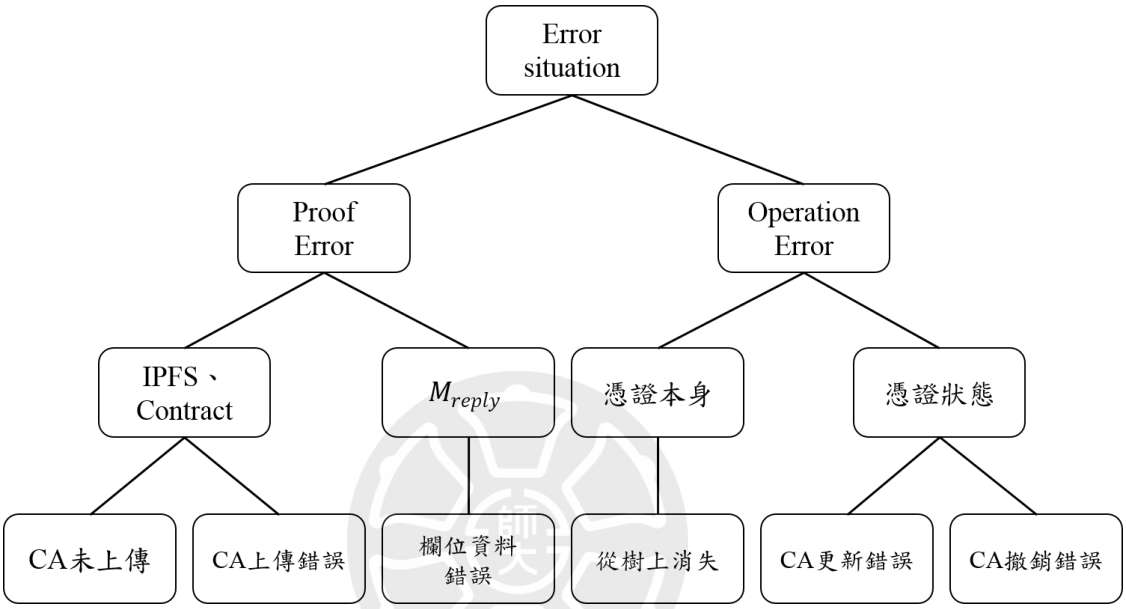

圖 9 錯誤情形分類圖

雖然大部分情形都可以進行申訴，但在 IPFS 及 Contract 上的錯誤情形，我們並沒有辦法進行申訴。以「未上傳證據」來說，CA 在上傳證據到這兩個地方時，並不會與任何CA 進行互動，因此無法透過訂定通訊協議的方式取得密碼學證據。至於「CA 上傳錯誤」，只有 IPFS 的上傳錯誤無法申訴，因為 Contract 上的 Root hash 錯誤，Owner 可以透過 Slice 來證明，但 IPFS 上傳的是整棵 TP-Merkle Tree，驗證整棵 Tree 的情況，需要計算所有的 leaf node，而 Contract 有資料上傳限制，Owner 無法將錯誤的 Tree 完整打入 Contract 內驗證。

所幸，我們可以假設上述兩種情況並不會發生，第一個理由是，雖然不能申訴，但是 Owner 還是可以知道 CA 是否有錯誤，因此 CA 不論未上傳或上傳錯誤，都會因為失去 Owner 信任，導致後續經營問題，再者，使用者也可以在現實生活中簽署相關合約，透過現實的法律來約束 CA。

第七節 申訴

前面已經分析完所有錯誤情形，在這個章節將會詳細說明任何錯誤的詳細申訴流程，包括 Owner 要上傳甚麼證據、Contract 會如何判決。

第一段 憑證從樹上消失

定義：Owner 在稽核階段的第三個步驟，檢查憑證資料時，直接找不到憑證訊息，意即 key-value pair 移失。這有兩種可能，第一種是 CA 在申請的時候就未放入 Tree 中，另一種則是 CA 某日不小心刪除 key-value pair。

情境：

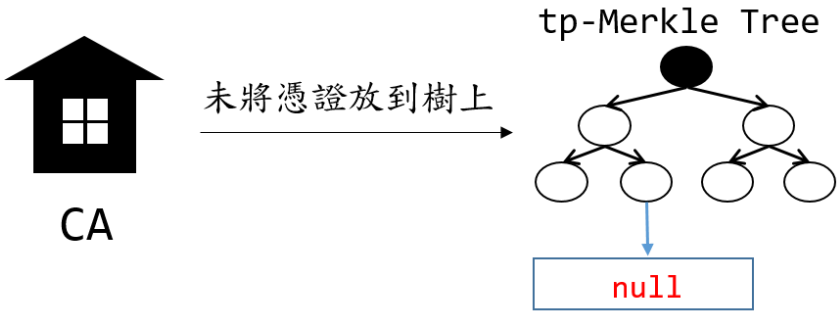

圖 10　CA 未放入之情境

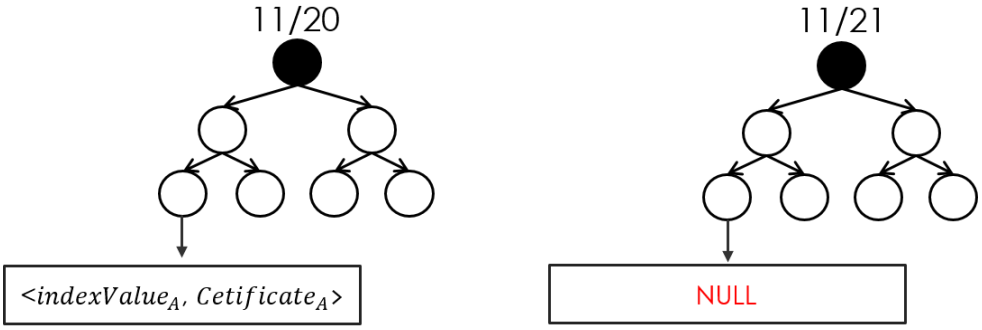

圖 11 從樹上消失之情境

申訴依據：$ACM_{request}$ 、ACM_{reply} 、List of key-value pairs。

申訴步驟：

第一步：Owner 將回條ACM_{reply}、Slice、list of key-value pairs 打入 Contract。

第二步：Contract 檢查回條上 CA 簽名是否正確。

第三步：Contract 確認 CO 與 Root hash 是否與本身記錄的一致。

第四步：Contract 計算 Slice、list of key-value pairs 是否正確。

第五步：Contract 檢查 list of key-value pairs 內是否有ACM_{reply}記錄之憑證。

第六步：若 List of key-value pairs 查無憑證，則判定 CA 弄丟憑證，Contract 將轉出一筆加密貨幣給 Owner，當作賠償金。

第二段 CA 狀態更新錯誤

定義：Owner 有向 CA 發送$CSM_{request}$，CA 同意請求也給了CSM_{reply}，但 Owner 在稽核階段的第三個步驟，檢查憑證資料時，發現 key-value pair 的憑證狀態與CSM_{reply}記錄不同。

情境：

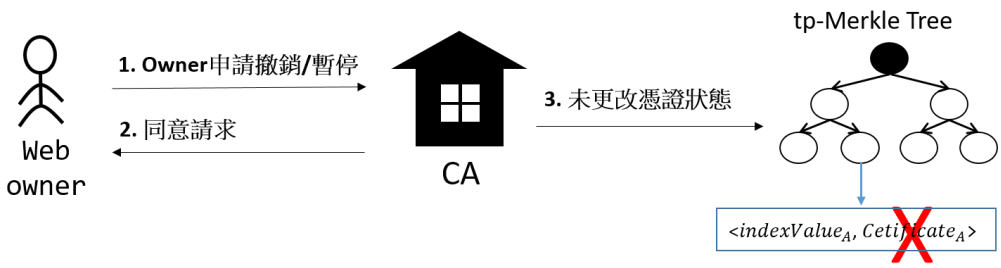

圖 12　CA 狀態更新錯誤之情境

申訴依據：CSM_{requet} 、CSM_{reply}、List of key-value pairs。

申訴步驟：

第一步：Owner 將回條CSM_{reply}、Slice、list of key-value pairs 打入 Contract。

第二步：Contract 檢查回條上 CA 簽名是否正確。

第三步：Contract 確認 CO 與 Root hash 是否與本身記錄的一致。

第四步：Contract 計算 Slice、list of key-value pairs 是否正確。

第五步：Contract 比對 List of key-value pairs 內記錄的憑證狀態是否與 CSM_{reply}記錄相同。

第六步：若比對結果不相同，判定 CA 狀態更新錯誤，Contract 將轉出一筆加密貨幣給 Owner，當作賠償金。

第三段 CA 撤銷憑證錯誤

定義：CA 撤銷憑證並未按照正常程序(先暫停，在撤銷)，導致 Owner 在稽核階段，發現憑證狀態從 Add 或 Renew，直接變為 Revoked。

情境：

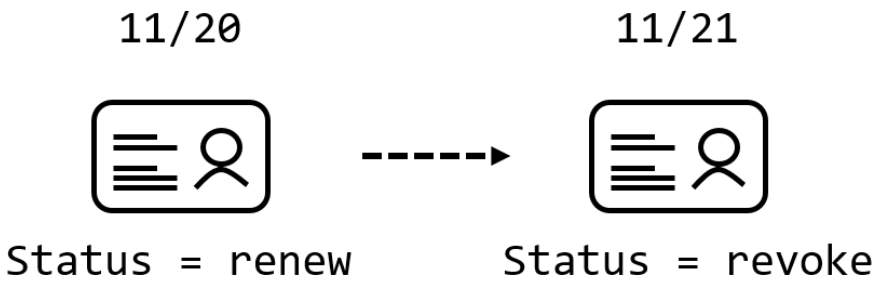

圖 13　CA 撤銷憑證錯誤之情境

申訴依據：CO 相連的兩條 Slice。

申訴步驟：

第一步：Owner 將兩條 Slice、list of key-value pairs 打入 Contract。

第二步：Contract 檢查兩個 Slice 的 CO 是否相連。

第三步：Contract 確認兩對 CO 與 Root hash 是否與本身記錄的一致。

第四步：Contract 計算兩條 Slice、list of key-value pairs 是否正確。

第五步：Contract 比對兩個 List of key-value pairs 內記錄的憑證狀態，是否分別為 Pause、Revoked。

第六步：若比對結果不符合，判定 CA 未照正常程序進行撤銷，Contract 將轉出一筆加密貨幣給 Owner，當作賠償金。

第四段 M_{reply}欄位資料錯誤

定義：Owner 收到的任何種類M_{reply}，其 result 為 Accept，但記錄的資料與$M_{request}$不同

情境：

圖 14　M_{reply}欄位資料錯誤之情境

申訴依據：$M_{request}$ 、M_{reply}

申訴步驟：

第一步：Owner 將M_{reply} 、Slice、list of key-value pairs 打入 Contract。

第二步：Contract 檢查回條上 CA 簽名是否正確。

第三步：Contract 確認M_{reply}的 result 是否為 Accept。

第四步：Contract 比對M_{reply}與$M_{request}$的 Status 是否一致。

第五步：若比對結果不符合，判定 CA 回條有誤，Contract 將轉出一筆加密貨幣給 Owner，當作賠償金。

M_{reply}欄位錯誤除了 Status 以外，還有 Hash(Certificate)也可能會發生錯誤，但 Hash(Certificate)錯誤會導致 Owner 在稽核時，查無憑證資訊，因此可接使用「憑證不在樹上」的方式申訴即可

第五段 CA 上傳錯誤證據

定義：Owner 取得的 Slice，其 Root hash 與 Contract 上記錄的不一致。這有兩種可能，第一種是 CA 上傳錯誤的 Root hash 到 Contract 上，第二種則是 CA 發

給 Owner 的 Slice 有錯誤。可以同時申訴兩種情況的理由是 Contract 只有 CA 能上傳，而 Slice 有 CA 簽名，兩者都只有 CA 可以操作，因此若資料發生不同步，我們可以合理推斷一定是 CA 自己操作失誤。

情境：

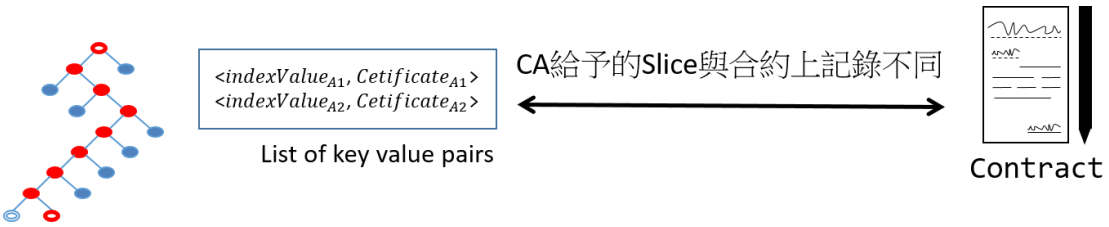

圖 15　CA 上傳錯誤證據之情境

申訴依據：Slice

申訴步驟：

第一步：Owner 將 Slice、list of key-value pairs 打入 Contract。

第二步：Contract 檢查 Slice、list of key-value pairs 簽名是否正確。

第三步：Contract 確認 CO 與 Root hash 是否與本身記錄的一致。

第四步：若比對結果不符合，判定 CA 上傳錯誤證據，Contract 將轉出一筆加密貨幣給 Owner，當作賠償金。

第八節　檢驗憑證

檢驗憑證的方式與前面的稽核流程相似，均要計算 Slice 與 list of key-value pairs，但對 User 來說，只需要確認 key-value pair 是否為 CA 所公告的，而不像 Owner 除了要計算 Slice 與 list of key-value pairs 是否正確外，還要檢查 key-value pairs 所存放的資料是否有物，因此我們以 User 的角度設計了另一種較簡單的驗證的方式，並且將驗證資訊改由 Owner 提供給 User，不需要透過 CA，這是以往的 PKI 架構所作不到的，最主要的原因是 User 無法確定 Owner 給的撤銷資訊是否正確。

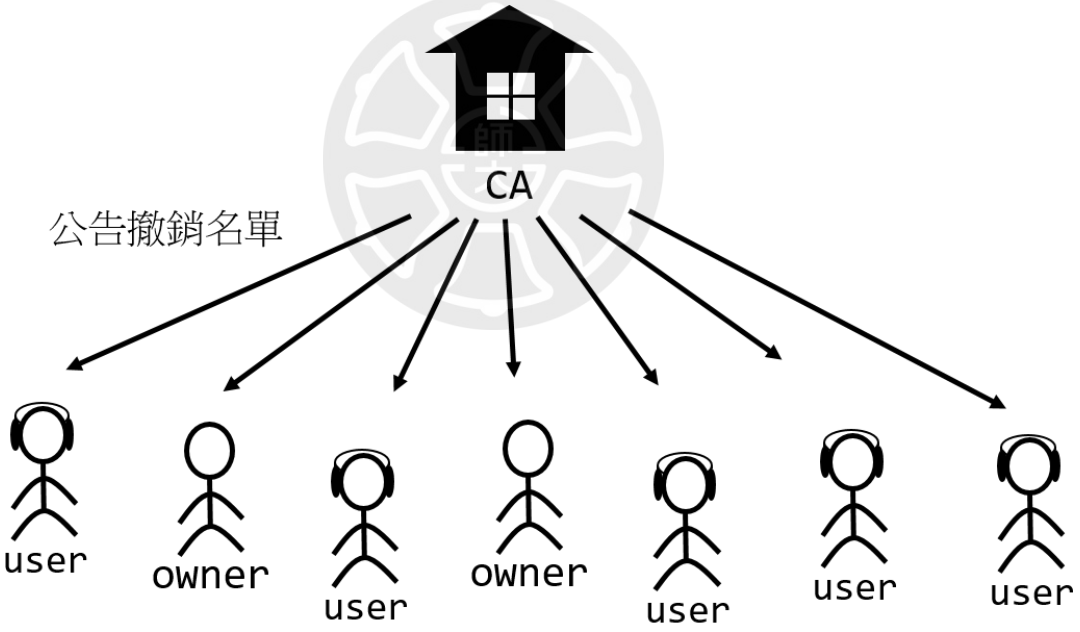

圖 16 傳統 PKI 檢驗流程

本篇論文將證據上傳到 Contract 的動作，讓 User 擁有識別 Owner 是否造假的能力，即使 Owner 提供的資料真的有誤，User 可直接當作憑證無效，或是改向

CA 索取正確的憑證資料。

詳細步驟如下：

第一步：Owner 在每日清算階段向 CA 取得 Slice、list of key-value pairs。

第二步：Owner 將驗證資訊與憑證本身一起公告在網站上。

第三步：若有 User 進入網站，它會先到 Contract 上比對 Root hash 是否與網站公告的一樣。

第四步：計算 Slice、Slice、list of key-value pairs 是否正確。

第五步：確認憑證狀態是否有效。

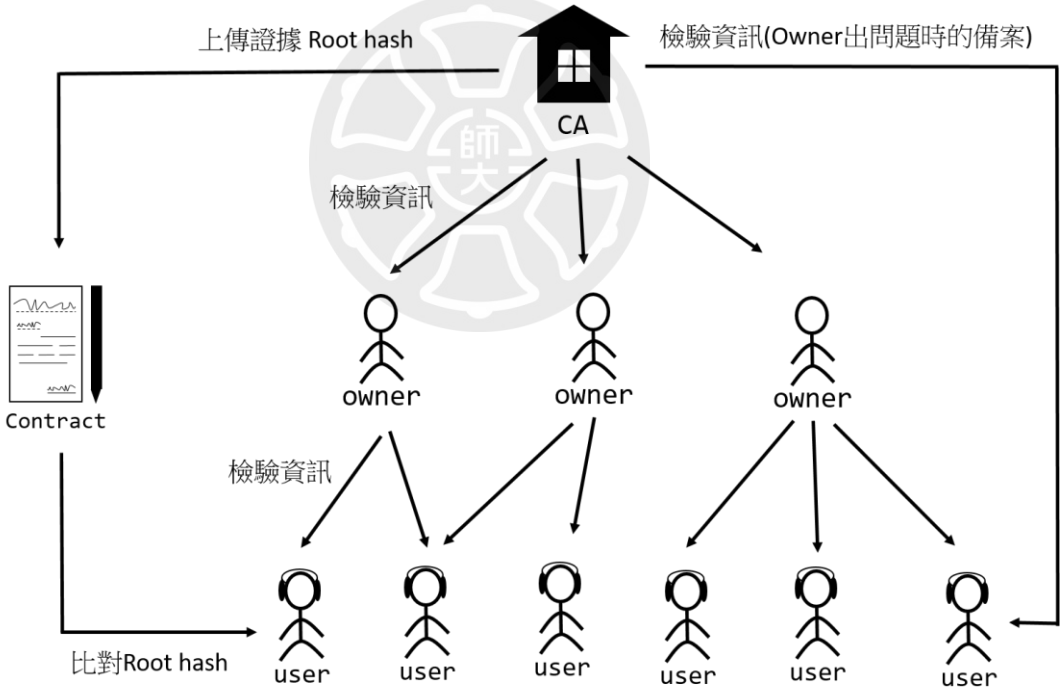

圖 17 分散式 PKI 檢驗流程

這種分層式的檢驗架構帶來許多好處，從上圖可以看到，CA 不用再處理大量 User 的憑證狀態查詢請求，只需要面對第二層的 Owner 們，能夠大幅度降低

Server 的負擔，我們可以透過公式更明確的展示其中差異。

(1) 傳統式檢驗：$burden_{CA}$ = N*S+M

其中$burden_{CA}$為 CA 的 server 負擔，N 為 User 數量，M 為 Owner 數量，S 為 User 一天瀏覽的網頁數量。

(2) 分散式檢驗：$burden_{CA}$ = M+1+K

M 為 Owner 數量，1 為向 contract 上傳證據，K 為 User 因 Owner 給予錯誤資訊時，CA 進行備案支援的次數。

除了 CA 有好處以外，對 Owner 來說，每日索取 Slice 的動作，可以及時發現 CA 是否操作錯誤；對 User 來說則是可以使用不只一種管道進行憑證檢驗。

第四章　實驗結果

第一節　環境說明

我們採用以太坊智能合約來實現申訴用合約，該合約使用 Solidity 語言進行開發，然後部署到 Ropsten Test Network 進行測試，申訴所耗費的金額使用以太幣匯率進行分析，系統架構使用 Java 11 實作，實驗用電腦作業系統為 win10，CPU 處理器為 Intel(R) core(TM) i5-4570 CPU @ 3.20GHz 3.20 GHz，記憶體為 16GB。

第二節 TP-Merkle Tree

TP-Merkle Tree 可以壓縮資料，但根據不同的樹高，每個 leaf node 所能存放的 key-value pair 數量也不同，這代表不同樹高所取出來的 Slice 大小有明顯差異，而這個差異將影響到 Owner、User 檢驗時需要索取的資料大小。由於放入資料是使用 Index Function，所以我們將根據 Index Function 在不同樹高的碰撞情形，就可以分析 key-value pair 的數量，

表 5- 1 TP-Merkle Tree 碰撞測試

樹高	葉節點數	平均碰撞次數	最小碰撞次數	最大碰撞次數
1	1	1000000	1000000	1000000
4	8	125000	124395	125439
6	32	31250	30955	31656
8	128	7813	7582	7983
10	512	1951.658	1812	2075
12	2048	487.951	403	559
14	8192	121.890	84	164
16	32768	30.405	9	53
18	131072	7.754	0	24
20	524288	2.35	0	11
21	1048576	1.462	0	8

表 5- 2 TP-Merkle Tree 樹高對於 Owner 與 CA 的儲存空間需求。

樹高	Client	Server
4	7.629 MB	136 MB
8	488.786 KB	136 MB
10	122.533 KB	136 MB
12	31.76 KB	136 MB
14	8.16 KB	137 MB
18	2.66 KB	164 MB
20	2.54 KB	251 MB
21	2.67 KB	367 MB

表 5-2 的實驗數據是透過 java.io.ObjectOutputStream 中的 writeObject 將建構好的 TP-Merkle Tree 輸出成一個物件檔儲存在電腦當中。其中，CA 儲存的是整棵 TP-Merkle Tree，而 Client 需要儲存的只有 Slice、list of key-value pairs；14 層以前 leaf node 少，但 list of key-value pairs 龐大，所以 CA 儲存少，Client 儲存多，14 層以後情況則相反，但考量到 CA 與 client 之間的儲存空間成本不同，因此 21 層樹高是最理想的狀態。

第三節 智能合約

智能合約有別於一般的程式，開發完畢要上線進行使用時，開發者不需額外架設主機來佈署程式，而是直接佈屬到區塊鏈中，當有人要使用合約中的某些功能時，需要呼叫礦工來幫忙執行，因為智能合約中可能會涉及到交易、資料儲存或讀取記錄等跟區塊鏈有關的操作。雖然礦工會幫忙做所有事情，但他們並不是無償工作，不論是佈屬合約還是執行合約功能，都需要支付礦工一筆價格不等的手續費來當作使用區塊鏈的成本，而這個成本又被稱為「Gas」，每一筆手續費會由「Gas Use」以及「Gas Price」兩者相乘而得。

Gas Use 代表這次執行智能合約所使用的 Gas 量，並且在執行合約前會先設定 Gas limit，其目地為限制此次執行所能使用的最大成本，這是為了避免有程式不停占用區塊鏈資源，也能保護使用者呼叫到有無窮迴圈的合約時，錢不會被瞬間被用光，因此只要執行時使用超過 Gas limit，礦工會直接判定交易失敗，並且照收手續費不退還。

Gas Price 代表使用者所願意對每成本付出的單價，使用以太幣不同大小的貨幣單位當作 Gas 單價名稱，較常用的單價名稱為 wei (10^{-18} Ether)、Gwei (10^{-9} Ether)，Gas Price 由使用者自行出價，但價格高低會影響到礦工處理交易優先順序，因此使用者常常需要在快速執行或節省手續費兩者中做抉擇。

第一段 合約部署

雖然智能合約在部署及呼叫功能時都須要支付成本，但我們可以將成本分開來看，因為 CA 只負責制定合約內容、部署合約，部署完後是由礦工負責執行，CA 不用再支付額外成本；而 Owner 只需要到合約執行申訴的功能即可，不用與 CA 進行其他互動，因此只有呼叫合約功能時給付給礦工的手續費。

表 5- 3　部署合約及合約基本功能之 Gas Use。

Function	Gas Use	Gas Price	ETH	USD	NTD
部屬合約	2,310,253	Fast (25 Gwei)	0.057756325	11.55	346.53
		Slow (15 Gwei)	0.034653795	6.93	207.92
上傳 RootHash	44,029	Fast (25 Gwei)	0.001100725	0.22	6.60
		Slow (15 Gwei)	0.000660435	0.13	3.96
驗證 slice	138,582	Fast (25 Gwei)	0.003464550	0.69	20.78
		Slow (15 Gwei)	0.002078730	0.41	12.47
驗證 k-v pairs	28,564	Fast (25 Gwei)	0.000714100	0.14	4.28
		Slow (15 Gwei)	0.000428460	0.08	2.57
驗證 Request	39,143	Fast (25 Gwei)	0.000978575	0.19	5.87
		Slow (15 Gwei)	0.000587145	0.11	3.52
驗證 Reply	38,173	Fast (25 Gwei)	0.003817325	0.19	5.72
		Slow (15 Gwei)	0.000572595	0.11	3.52

表 5- 4 申訴功能之 Gas Use。

Objection	Gas Use	Gas Price	ETH	USD	NTD
憑證不在樹上	251,634	Fast (25 Gwei)	0.00629085	1.25	37.74
		Slow (15 Gwei)	0.00377451	0.75	22.64
CA 狀態更新錯誤	250,552	Fast (25 Gwei)	0.00626380	1.25	37.58
		Slow (15 Gwei)	0.00375828	0.75	22.54
CA 憑證撤銷錯誤	285,922	Fast (25 Gwei)	0.00714805	1.42	42.88
		Slow (15 Gwei)	0.00428883	0.85	25.73
CA 上傳錯誤證據	147,466	Fast (25 Gwei)	0.00368665	0.73	22.11
		Slow (15 Gwei)	0.00221199	0.44	13.27
M_{reply}欄位錯誤	59,783	Fast (25 Gwei)	0.00149458	0.29	8.96
		Slow (15 Gwei)	0.00089675	0.17	5.38

做此實驗時，1 Ether 的價格大約是 200 美金左右，站在 CA 的角度來看，部署合約大約要 350 元台幣，但總共只需要部署一次，而每天上傳 Root Hash 的動作也只需要 6 元就可以達成，一年的總成本大約 2000 左右，對企業來說是非常少的。若以 Owner 的角度來看，假如憑證都沒有出問題，根本不需要進行申訴，就算真的需要申訴，最多也只會用到 40 元左右，但是申訴成功所得到的賠償絕對遠大於支付的手續費。

第五章 結論

以往區塊鏈 PKI 是不可行的，因高信任度的公有區塊鏈無法應付數量龐大的憑證發放，而使用私有鏈又無法滿足憑證安全性，但我們使用了 tp-Merkle Tree 這個資料結構，讓公有鏈 PKI 能夠真正負荷所有憑證的發放與撤銷。

再者，CA 上傳 Root Hash 到 Contract 時，憑證資訊均會經過雜湊，可完全符合 GDPR「假名化」，而透過智能合約設立 CA 押金制度，結合 tp-Merkle Tree 產生密碼學證據，建立全自動化賠償系統，除了保障憑證擁有者的權益，也能降低 CA 出錯的機率，最後，我們將智能合約變成一個真正的可信任第三方，讓網頁使用者不再只能從 CA 取得憑證撤銷資訊，CA 也能直接減少 90%以上的連線需求。

最後，從成本方面來看，若有 CA 採用本篇論文的架構，由於連線負擔下降，CA 不必再架設與之前相同的 Server 量，可以減少設備的維護與架設成本；至於有效的密碼學證據結合線上自動賠償機制，也讓 Owner 不再需要透過客服人員申訴，可以減少 CA 聘用客服人員的人事成本，因此不論是人事成本還是設備成本都能夠大幅度的縮減預算。

參考文獻

1. Cooper, David, et al. "RFC 5280: Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile." *IETF, May* (2008)
2. Santesson, Stefan, et al. "X. 509 Internet Public Key Infrastructure Online Certificate Status Protocol-OCSP." RFC 6960 (2013). p. 1-41
3. Housley, Russell, et al. Internet X. 509 public key infrastructure certificate and CRL profile. RFC 2459, *January* (1999).
4. Ellison, Carl, and Bruce Schneier. "Top 10 PKI risks." *Computer Security Journal* 16.1 (2000).
5. Comodo hacker claims credit for DigiNotar attack. January 24, 2014. Available from: https://www.pcworld.idg.com.au/article/399812/comodo_hacker_claims_credit_diginotar_attack/.
6. 58% of Phishing Websites Now Use HTTPS. Available from: https://www.thesslstore.com/blog/58-of-phishing-websites-now-use-https/.
7. Garfinkel, Simson. *PGP: pretty good privacy*. " O'Reilly Media, Inc.", 1995.
8. Swan, M., *Blockchain: Blueprint for a new economy*. 2015: " O'Reilly Media, Inc".
9. AL-BASSAM, Mustafa. SCPKI: a smart contract-based PKI and identity system. In: *Proceedings of the ACM Workshop on Blockchain,*

ryptocurrencies and Contracts. 2017. p. 35-40.

10. Lewison, Karen, and Francisco Corella. "Backing rich credentials with a blockchain PKI." *Tech. Rep.* (2016).

11. Ethereum. Available from: https://etherscan.io/.

12. Let's Encrypt 統計數據. Available from: https://letsencrypt.org/zh-tw/stats/.

13. Regulation, Protection. "Regulation (EU) 2016/679 of the European Parliament and of the Council." *REGULATION (EU)* 679 (2016): 2016.

14. TP-Merkle Tree. Available from: https://itrustmachines.com/

15. Swan, Melanie. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

16. Nakamoto, Satoshi. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

17. Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* 151.2014 (2014): 1-32.

18. Litecoin. Available from: https://chainz.cryptoid.info/ltc/.

19. HWANG, Gwan-Hwan; HUANG, Wei-Sian; PENG, Jenn-Zjone. Real-time proof of violation for cloud storage. In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, 2014. p. 394-399.

20. Hwang, G.H., et al., *Fulfilling mutual nonrepudiation for cloud storage.* 2016.

28(3): p. 583-599.

21. Benet, Juan. "Ipfs-content addressed, versioned, p2p file system." *arXiv preprint arXiv:1407.3561* (2014).