國立臺灣師範大學理學院資訊工程學系

博士論文

Department of Computer Science and Information Engineering
College of Science
National Taiwan Normal University

Doctoral Dissertation

使用定位摩克樹作資料存證的應用研究

Applications of Transaction Positioned Merkle Tree for Data Attestation

黃鯤義

HUANG, KUN-YIH

指導教授：黃冠寰 博士

Advisor：HWANG, GWAN-HWAN, Ph.D.

中華民國　一一三　年　一　月

January 2024

謝誌

在我的這一生中，感到最幸運的事之一，就是遇到了　黃冠寰教授，並擔任我的指導教授。黃教授對於包括我在內的每一位學生，指導的用心與鉅細靡遺，數十年如一日，令我相當的敬佩！

我在 2015 年，首次遠赴北歐的芬蘭，參加 IEEE TrustCom 國際學術會議口頭發表論文，2022 年於 IEEE ECICE 國際學術會議獲得最佳論文獎的肯定，這些難得的經歷，也都要感謝我的老師　黃冠寰教授的指導。

感謝這次學位口試的委員：資策會資安所總監許建榮博士，啟碁科技公司資深經理毛敬豪博士，國立東華大學資工系張道顧副教授，威聯通科技公司專案副理林哲生博士，國立臺灣師範大學資工系賀耀華副教授；每一位老師，對於論文的每一個部分與細節，都相當用心而仔細地給予我建議，讓我相當的感動，也讓論文可以更臻完善，非常謝謝所有的口試委員！

最後，感謝在我的這一生中，所有對我默默幫助過的每個人，沒有你們，也沒有現在的我，也希望我能將我的一生，奉獻給需要我的人。

黃鯤義　謹誌於

國立臺灣師範大學資訊工程研究所

中華民國一一三年一月

摘要

在大數據（big data）的網路時代，由於各種原因，無論是人為造成的或意外發生的情況，都可能導致有價值的資訊遭受損壞、竄改或竊取等危害。因此，確認各種活動或資訊交易的身份正確性，以及保障其內容、結果的安全性、以及日後追查稽核或即時稽核與驗證的相互不可否認性與可歸責性，成為大數據網路時代資訊安全的核心工作。公有區塊鏈（public blockchains）憑藉其去中心化的分散式架構，具有不可竄改性與透明性，透過共識協定使得網路節點能夠相互監督，進而達到資料的可信任性。

然而，受限於區塊鏈高額礦工費與每秒交易筆數（TPS）的低限制，大量的資訊難以儲存於區塊鏈中。因此，本論文採用了Hwang等人提出的定位摩克樹（transaction positioned Merkle tree）[83, 97, 98, 100]作為存證的基礎技術。在對定位摩克樹的效能進行一般性測試之後，筆者選擇了兩個代表性的情境進行深入研究。

第一項研究提出了雲端服務執行環境完整性即時稽核的架構，這不僅可以避免執行環境因遭攻擊、竄改或損壞所造成的意外，同時也能夠在系統運作時即時發現是否有遭受攻擊、竄改、遺失檔案或惡意軟體的植入，例如電腦病毒或木馬程式。

第二項研究模擬了如何在真實人類情境中，利用定位摩克樹與公有區塊鏈，實現基於公有區塊鏈的自動給付與申訴賠償機制。結果證明了利用定位摩克樹的

證據存證技術可以完全解決情境中的信任問題，且不受限於公有區塊鏈效能瓶頸。

總結而言，本研究提供了一個具體而有效的方法，結合定位摩克樹與公有區塊鏈，以應對大數據網路時代資訊安全的挑戰。這些方法不僅具有實用性，同時突破了公有區塊鏈效能的桎梏。

關鍵字：public blockchain, smart contract, decentralized data attestation, tp-Merkle tree, cloud computing, cloud auditing, decentralized auditing, blockchain based automatic reward

ABSTRACT

In the era of big data in the Internet, various factors, whether intentional or accidental, have led to valuable information being damaged, altered, or stolen. Therefore, ensuring the correctness of identities in various activities or information transactions, the security of their content and results, as well as the mutual non-repudiation and accountability of tracing or real-time auditing in the future, are the primary tasks of information security in the big data network era. Due to the decentralized and distributed architecture of public blockchains, which possess immutability and transparency, the network nodes can supervise each other through consensus protocols, thereby achieving data trustworthiness.

However, due to the high transaction fees (miner fees) and the low transaction per second (TPS) of blockchains, a large amount of information cannot be attested on the blockchain. Therefore, this dissertation adopts the transaction positioned Merkle tree (tp-Merkle tree) [83, 97, 98, 100] proposed by Hwang et al. as the foundational technology for evidence preservation. After conducting general performance tests on the tp-Merkle Tree, the author chose two representative scenarios for in-depth research.

The first study proposes an architecture for real-time auditing of the integrity of cloud service runtime environments. This can not only prevent accidents caused by attacks, tampering, or damage to the execution environment but also detect in real-time whether the system is under attack, being tampered with, has lost files, or

has been implanted with malicious software, such as computer viruses or Trojan horses.

The second study simulates how to implement an automatic payment and complaint compensation mechanism based on public blockchains and tp-Merkle trees in real human scenarios. The results demonstrate that the evidence preservation technology using tp-Merkle tree can completely solve trust issues in the scenario and is not limited by the performance bottleneck of public blockchains.

In conclusion, these researches provide a concrete and effective method that combines tp-Merkle trees with public blockchains to manage and deal with the challenges of information security in the big data network era. These methods are not only practical but they also overcome the limitations of public blockchain.

Keywords: public blockchain, smart contract, decentralized data attestation, tp-Merkle tree, cloud computing, cloud auditing, decentralized auditing, blockchain based automatic reward

目錄

附表目錄

附圖目錄

第一章、簡介

在日常生活中存在著許多的交易（deals or transactions）與作業，或稱為活動（operations），自從 1980 年代出現了網際網路（Internet）[1]，漸漸地，許多的活動與交易開始在網路上出現，許多商家也在網際網路上提供交易服務，包括一些加值性服務，如：雲端儲存空間的租用（cloud storage）[2] 即為一例。

為了讓企業經營更有效率而達到最大的獲利，於是「企業資源規劃」（enterprise resource planning，簡稱 ERP）的概念被提了出來 [3, 4]，其中包含有供應鏈管理（supply chain management，簡稱 SCM）以及客戶關係管理（customer relationship management，簡稱 CRM），而這其中需要大量運用資訊科技與網際網路來處理，於是 Intranet（可翻譯為：企業內網際網路）、Extranet（可翻譯為：企業間網際網路）[5] 與虛擬私人網路（virtual private network，簡稱 VPN）[6] 等的基於網際網路的基礎建設的網路技術於是出現；而另外為了處理「商業模式」（business model）的應用層面，即所謂商業模式的構成 [7]－Business Model Canvas（簡稱 BMC），「Infrastructure management」：key activities、key resources、partner network，「Product」：value propositions，「Customer interface」：customer segments、channels、customer relationships，「Financial aspects」：cost structure、revenue streams（以上整理自 Osterwalder 2004 博士論文 [7] ），所需要的資訊科技技術，以及運用在電子商務（e-Commerce）[8]、公司對公司（business to business，簡稱 B2B）、公司對客戶（business to customer，簡稱 B2C）等的資訊

網路的應用技術不斷的出現，而這些都會透過網際網路去建立商流（business flow）、物流（logistic flow）、金流（money flow）以及資訊流（information flow）等所謂「四流」的管理等 [8]。所以，資訊與網路間的安全問題，從此而益形重要而為大家所重視。

於是有些是傳統商家會利用網路提供金流與付費，有些是所有交易活動都在網際網路上，即我們所謂的「電子商務」，有些則是利用網際網路提供所謂的加值性服務（即付費服務）。經過了資訊技術的不斷進步，這服務甚至擴展至所謂的雲端服務（cloud Services）：即所謂的「基礎即服務」（Infrastructure as a Service，簡稱 IaaS）、「平台即服務」（Platform as a Service，簡稱 PaaS）、「軟體即服務」（Software as a Service，簡稱 SaaS）等的雲端加值性服務 [9]。例如 2006 年 Amazon 推出的 EC2 服務 [10]；甚至是現在的「元宇宙」（Metaverse），以及「擴增實境」（Augmented Reality，簡稱 AR）、「虛擬實境」（Virtual Reality，簡稱 VR）、「混合實境」（Mixed Reality，簡稱 MR）等的虛擬式服務 [11]。而以上這些服務的共同存在的問題是：如何確認在發生爭議時，所有作業過程是可歸責的（accountability）而進行追究？

如果這些服務的運作過程與經歷的關鍵資訊能夠存證（data attestation）[12] [13]，在爭議的追查上，或者證據的鑑識（forensics）[14] 上，才能夠進行。如果是在網路上的交易，牽涉到費用的支付，雙方才有可信賴交易證據確認費用是否正確；而違約的損害賠償，才有充分的證據得以究責。

第一節 由 Web 2.0 到 Web 3.0

所謂 Web 1.0 它的由來，其實是由於一個時代的進步，為標記前一世代，現在所後創的名詞標記，英文稱之為「retronym」。當網路社群蓬勃發展，大數據（big data）的議題來臨，以及與網際網路服務多樣化的時代，被稱為 Web 2.0 時代時，人們便開始詢問：Web 3.0 是甚麼？

由於網際網路發展之初，是由軍事用途，擴展到學術用途，一直到開放全球大眾得以使用後，也鑑於當時圖形化操作介面的興起，如 X-windows、Macintosh 等 [15]，1990 年初第一款網頁瀏覽器問世，遂開啟了網際網路普及化的開端，「全球資訊網」（World Wide Web）[16] 技術開始出現，讓一般人更容易使用網際網路；而 1993 年第一個普及化的網際網路瀏覽器（Internet blowser）問世，即「Mosaic」，成就了網際網路 Web 1.0 的時代。許多的機構紛紛開始設立網站伺服器，也就是 Web server，而為了更方便一般人搜尋網路資訊，有了所謂「入口網站」（Web portal） [17] 的出現，並藉此服務營利，如「蕃薯藤」、「奇摩」、「Excite」以及「Yahoo」等，不過這個時期的網際網路，大多以提供資訊服務為主。

1996 年 ICQ 即時通訊軟體 [18]的推出以及 1997 年社交網站 SixDegrees.com 的出現 [19]，開啟了網際網路的社群互動服務，此後網路社群便逐年蓬勃，如網際網路電子布告欄系統（BBS）、「網誌」或稱「部落格」（Weblog，簡稱 Blog），造就了我們所謂互動式的網際網路 Web 2.0 的時代，網路的兩大巨頭：Google、

Facebook 便在此時出現。2007 年 Apple 公司的 iPhone 智慧型手機問世，開啟了現代智慧型行動通訊的文化，更加速了 Web 2.0 時代的發展，2013 年第一個擴增實境的 Android 手機遊戲「Ingress」誕生，而另一個擴增實境的手機遊戲：寶可夢（Pokemon GO），更造成了全球的瘋迷。

然而此時，雲端服務的多樣化與便利性，因為智慧型手機的普及，使得人類的日常生活與所有活動，似乎很多的部分已融入到網際網路之中，這使得人類日常生活的許多資訊，無形中已悄悄集中在雲端服務提供商的伺服器中了，這明顯造成了個人隱私的極大危機。例如妳的 Android 手機的 GPS 一直開啟著，Google 也就是 Alphabet 公司，透過 Google Map，知道妳去過哪些地方？做過哪些事情？以及上網的偏好是什麼？進而可以分析出妳的個人特質！又例如 2014 年據稱是 Apple 公司的 iCloud 遭到駭客（hacker）入侵，導致個人資訊外洩，但最後卻無法證明問題與責任在誰那裡？誰該負責？對於長期使用雲端儲存空間 Dropbox 的使用者，無法證明自己的檔案是否安全的存放在雲端？也就是說，雖然雲端運算（cloud computing）[9] 讓人類的生活進化並更便利，但是他仍然是一種「中心化」（centralization）的架構—用戶必須「信任」供應商！

2009 年初，網際網路出現了第一個區塊鏈（blockchain）：比特幣（Bitcoin）[20]，「去中心化」（decentralization）的概念於是出現，「去中心化」是目前 Web 3.0 的核心概念，區塊鏈技術，是運用點對點網路（peer-to-peer，簡稱 P2P）[21] 與數位簽章（digital signature）[22] 等的技術所構成；然而現代智慧型手機的發

展，同樣加速與催生了網際網路宇宙的發展，將真實與虛擬世界整合成為所謂的「Cyber space」[23]，致使智慧服務提供商，同樣可以加速個人資料的蒐集。同樣的，若個人資料集中在服務提供商，這種服務普及到全球村（Global Village）[24] 人們的生活之後，那麼「隱私權」與「安全性」的議題將再度成為「國家安全」與「全人類安全」的議題！因此 Web 3.0 的網路環境，便是希望在去中心化的的安全架構下，讓「虛擬金融」：Defi（Decentralized Finance）[25]、「虛擬資產」：NFT（Non-Fungible Token）[26]、「虛擬服務」：VR、AR、MR 等得以實現，且目前還在發展當中，形成所謂「元宇宙」的網際空間（Cyber space）。

第二節 可信任的第三方的問題

雲端儲存空間（cloud storage）是目前最為普遍的雲端服務之一；例如：Amazon S3(Simple Storage Service）[27]、Microsoft Asure [28]、Apple iCloud [29]、Dropbox [30] 與 Google Drive [31] 等，雖然各自提供不同的應用功能或加密授權的功能，但是在它們的服務條款（service-level agreement，簡稱 SLA）上 [27]，從來不保證資料是否會因遭洩漏（leaked）、竄改（tampered）或破壞（damaged），需要賠償其所衍生的損害，而最多只有保證高度的可用度（availability）在 99.9% 以上而已。顯然如果需要保證資料是否遭洩漏、竄改或破壞所衍生的損害賠償，則必須有可以明確「歸責」的機制來證明，也就是在所有的作業過程中必須能被稽核，而這些存放稽核的證據，必須有「不可否認性」與「可歸責性」的功用，且要有可信任的公正第三方用來存放證據，這樣才得以公平仲裁。

在過去有許多的研究中，無論採用哪些加密演算法與加密協定，都無法避免需要「可信任的第三方」（trusted third party，一般簡稱 TTP）[32]。Popa et al.（2011） [33] 提出所謂「CloudProof」的技術，利用「chain hashing」做到 peer-to-peer 的互相不可否認性，但是當同一個儲存空間存在多個設備共用時，就必需有公正的第三方持有所有的 chain hashing 證據，才得以正確稽核與仲裁。Hasan & Salah（2018）[34, 35] 運用乙太坊區塊鏈，提出一個去中心化的「送交證明」（Proof of Delivery）架構，讓買方在購買實體或數位產時，確認送達並自動付費；但是當爭議（dispute）發生時，仍需要一個公正的第三方擔任離鏈的仲裁者。而目前全球使用的公鑰系統（public key infrastructure，簡稱 PKI）[36]，如：台灣的自然人憑證 [37]，也是需經由公正的第三方機構 certificate authority（簡稱 CA）發行與管理憑證（certificate）[38]。

然而當加密、授權、認證、稽核等作業需要公正的第三方機構執行時，無形中安全性的資訊或個人隱私，便集中在所謂「公正的第三方」機構裡，這種新科技造就的資訊的壟斷，在網際空間裡，顯然隱含著資訊擁有者有權無責、無「法」可管的情形，若遭到濫用將形成全球安全危機！

我們知道，過去因為自由經濟與資本主義，造就了現代化的社會進步，但是若發生經濟壟斷 [39]，則可能引發支配人類生活的問題，甚至會破壞現有文明的生活、致使文明倒退，或造成經濟獨裁等問題。因此世界各國皆有公平交易的管理部門與相關法令加以管理，例如：美國的司法部有反壟斷署（Antitrust

Division, U.S Department of Justice）。

而這種我們稱之為「新科技資本主義」所導致的「資訊壟斷」與「科技服務壟斷」的現象，如第一節所述，將有可能如同「隱形帝國」[40] 與「隱形牢籠」[41] 等書中所描述的現實狀況：人類的生活與個人、機構，乃至國家的機密資訊，會完全被資訊服務提供商或持有者所統治！這是相當可怕的。

而就人類人格分析理論 [42] 的角度，無論是從「人本論」（humanistics）、「特質論」（trait theory）、「互動論」（interactionism）、「認知-行為理論」（reciprocal determinism），或者是「心理動力理論」（psychodynamics）的觀點分析，人格會受到人類生存本能和慾望的需求，與環境、外在，乃至社會等因素間的互相影響而產生變化。因此人類為了解決個人慾望與外在環境的衝突，所採取的可能策略，就可能導致「人為的造假與破壞」。所以在人性本善與人性本惡的衝突之間，我們往往無法信任看不見的第三方，

因此，所謂的「可信任的第三方」，反而有可能造成資訊安全的危機，所以，我們需要一個去中心化（decentralized）的安全平台。

第三節 去中心化技術：區塊鏈

區塊鏈的概念，最早出現在 Satoshi Nakamoto（2008）於 2008 年 10 月 31 日發表在 www.metzdowd.com/pipermail/cryptography/2008-October/014810.html 的一篇文章，名為 *Bitcoin：A Peer to Peer Electronic Cash System* [43]；這篇文章提出了一種去中心化的公有網路加密貨幣系統。Nakamoto 當初提出這個概念與

技術，或許只是基於實驗的性質，沒想到卻在日後造成了一股強大的網路與科技風潮，並形成了 Web 3.0 的核心：具有去中心化的可信任平台。

區塊鏈 [44]，提供了一個可信任的去中心化的平台，解決了長久以來，我們對於任何需要可信任第三方才能運作的系統架構的質疑。

簡單的說，公有區塊鏈的區塊，是存在於網際網路上一種存放交易資訊的資料結構。每一個資料區塊，包含有兩個部分：如圖 1。

一、形成區塊與區塊鏈的系統資訊。

二、交易資訊。

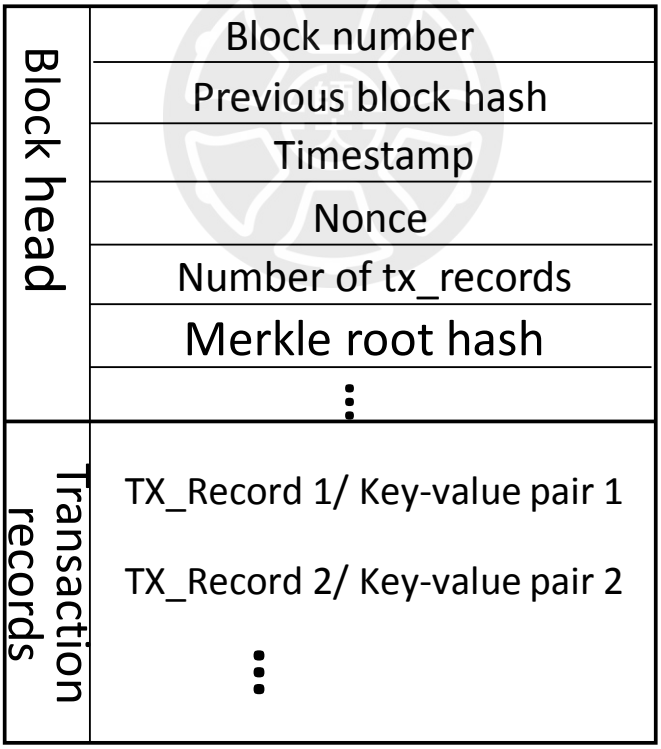


圖 1：區塊結構示意圖

公有區塊鏈系統之所以不需要一個可信任的中心系統，是因為在網際網路上的任何一部電腦，都可以不被限制地參與區塊鏈的維運，並且利用區塊鏈進行交易，因此她是一個去中心化的系統；而之所以鏈上的所有交易記錄可被信任，是他有預先制定好的一組規則，也就是我們通稱的協議（protocol），讓參與共同維運者遵循，使得該系統的所有交易記錄，無法被竄改並且可追溯。這些規則包括有：

一、區塊間依序連結的關連與區塊內交易資訊的加密與存證方式。例如：雜湊演算法（hash）、摩克樹（Merkle tree）[45]。

二、交易記錄的數位簽章方法。例如：RSA（Ron Rivest, Adi Shamir, Leonard Adleman）演算法、Diffie Hellman 演算法、橢圓曲線加密演算法（ellipse curve cryptography, ECC）等。[46]

三、共同維運區塊鏈的電腦間，產生區塊的共識（consensus）方法。例如：Proof-of-Work、Proof-of-Stake、practical Byzantine fault tolerance 等。[47] [48]

區塊鏈所構成的技術，在資訊科技領域裡，都是既有的成熟的技術，做了創新的應用組合，形成了一個去中心化可信任的系統。以下針對目前最大的兩個公有區塊鏈：比特幣區塊鏈（Bitcoin）與以太坊區塊鏈（Ethereum），作一個概覽性的介紹。

1.3.1 Bitcoin

比特幣（Bitcoin）[20] 於 2009 年 1 月 3 日正式上網，是一種數位加密貨幣（cryptocurrency），也是第一個去中心化的公有區塊鏈系統。因為他將所有的交易都記錄在區塊中，所有的區塊所形成相互關聯的區塊鏈，而任何一部在網際網路上的電腦都可以加入共同維運，並且必需儲存全部區塊鏈的資料，因此又稱為「分散式帳本」（distributed ledger）[49]。而加入維運的電腦稱為「礦工」（miner），礦工之間以「P2P」（peer to peer）[21] 的方式進行通訊；產生區塊的過程稱為「挖礦」（mining）。每一筆「交易」都必須經過數位簽章驗證，驗證通過才能記錄到區塊中。礦工依照「共識機制」所產生出來的每一個區塊，都會依序記錄前一個區塊的雜湊值，因此每一個產出的區塊都可以溯源，每一筆交易也都經過數位簽章的加密與驗證，而且所有礦工都會存放整個區塊鏈資料，因此區塊鏈上的資料是難以被竄改的。依照比特幣的共識演算法，只有在擁有維運比特幣區塊鏈總體的運算能力到達 51%以上，才能突破比特幣的安全機制。此外，比特幣區塊鏈只能進行數位加密貨幣的交易，且並不是圖靈完備（Turing-completeness）的計算系統。

比特幣所使用的技術包括有：

一、區塊與區塊間依序記錄前一區塊的雜湊值，以及每一筆交易記錄的雜湊值，其所使用的雜湊函數：SHA256。[46]

二、交易資訊的數位簽章：橢圓區線加密演算法 ECDSA（elliptic curve digital signature algorithm）[50]。

三、存證與驗證所有交易資訊所採用的方法：摩克樹 [45]。

四、區塊產生過程的共識演算法機制：Proof-of-Work [47]。

五、礦工間的通訊方式：P2P 網路 [21]。

1.3.2 Ethereum

原任比特幣雜誌（BITCOIN MAGZINE）[51] 首席編輯的俄裔加拿大籍的電腦天才 Vitalik Buterin，於 2013 年 11 月 27 日首度發布以太坊（Ethereum）白皮書：「Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.」 [52] ，同時於 2014 年 1 月 23 日在比特幣雜誌發表一篇同名的文章 [53]，並且在 2014 年 1 月 25、26 日在邁阿密所舉行的北美比特幣會議上，演講時宣布以太坊計劃，並於 2015 年初正式上線 [54]。以太坊的出現，進化了區塊鏈的發展，使區塊鏈的應用不僅僅侷限在加密貨幣上。

以太坊除了進化了比特幣的區塊鏈技術外，同時提供了開發環境，讓以太坊區塊鏈得以有除了加密貨幣以外的應用，最主要的就是以太坊智能合約（smart contract）[52, 55]。

以太坊智能合約提供了 Solidity [56]、Vyper 等圖靈完備（Turing-completeness）的程式設計語言 [57]，可以依照任何需要，事先撰寫好佈署在以太坊區塊鏈，當某條件成立時，則會觸發該佈署在以太坊區塊鏈上的智能合約執行，執行的交易內容便會自動的寫回、記錄在區塊鏈上，因此以太坊可以支援所謂的去中心化的應用開發（decentralized application，簡稱 Dapp）。

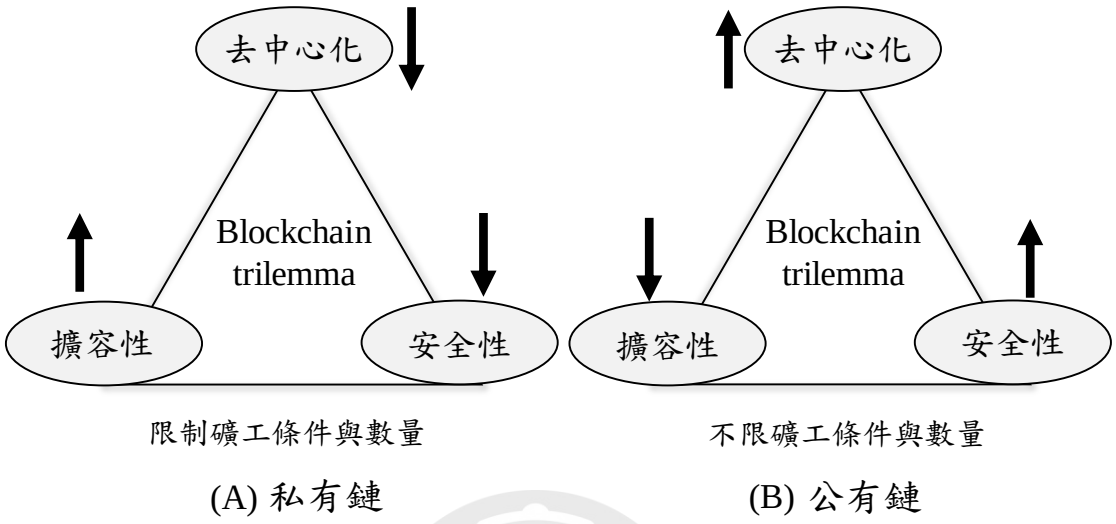
因此，以太坊區塊鏈可以說是圖靈完備的系統。

1.3.3 公有區塊鏈的問題

目前公有區塊鏈，如：比特幣、乙太幣、萊特幣 [58] 等，其參與維運的「節點」（也就是所謂「礦工」）數眾多，其所採用的共識機制如：Proof-of-Work 或 Proof-of-Stake，要發動區塊鏈 51%攻擊去竄改帳本，相當的困難，然而也因為如此，使得區塊鏈的擴容能力受到極大的限制！舉例而言，VISA 信用卡，目前每天平均全球每秒鐘交易量有數千筆，而 VisaNet 的允許容量可達每秒鐘 65,000 筆交易 [59]；如果我們以平均每秒 3 千個交易計算，若將交易資訊寫入比特幣區塊鏈，一年大約需要 48 tera-byte 的儲存空間！如果以如此大的成長量計算，礦工間的通訊也成為一大問題。每秒交易量，在區塊鏈上，我們稱之為「TPS」（transactions per second）。而事實上，目前比特幣每秒交易量不超過 7 TPS，乙太幣每秒交易量不超過 15 TPS [60]。如過要將區塊鏈去中心化的優勢，應用在各行各業，甚至是微支付系統，其擴容能力的限制，會是最大的問題。

因此以太坊區塊鏈的創始人 Vitalik Buterin 提出了公有區塊鏈的「三難」（blockchain trilemma）[61] 的問題，也就是「去中心化」（decentralization）、「安全性」（security）、「擴容能力」（scalability）三者無法同時達成，我們通常稱之為「不可能的三角」。如圖 2，私有區塊鏈（private blockchain）因為礦工數目少且受少數人控制，其每秒可處理交易數目雖可達到數千以上，但是「去中心」、「安全性」兩項無法達成。而公有區塊鏈（public blockchain），如以太坊，其

礦工數目眾多，因此「去中心」、「安全性」兩項可達成，卻因共識機制造成的延遲，與同步及複製交易的成本，造成「擴容能力」不佳，僅有 15TPS 以下。



限制礦工條件與數量　　不限礦工條件與數量

(A) 私有鏈　　(B) 公有鏈

圖 2：區塊鏈三難（blockchain trilemma）示意圖

因此，公有區塊鏈雖然提供了去中心化的信任機制，但是他的擴容問題大大的限制了他的應用與發展，歸納有以下四點問題：

一、第一個問題是交易頻寬（transaction bandwidth）不足。

如前所述，目前比特幣每秒交易量不超過 7TPS，乙太幣每秒交易量不超過 15TPS。公有區塊鏈的去中心化運作，依賴於全球網際網路使用者的維運和使用。因此，任何用戶都可以使用區塊交易來交換加密貨幣、編寫智能合約或記錄資訊。參與共識機制的公共區塊鏈中的所有節點，必須不斷更新區塊鏈中資料的任何變化，包括獲取一般用戶想要在區塊鏈上放置的交易資訊。因此，大量資訊必須透過 P2P 網路進行交換，

交易頻寬難以增加。

有人提出「私有鏈」（private blockchain）或「聯盟鏈」（consortium blockchain）試圖解決交易頻寬不足問題的方法，讓參與區塊鏈節點的數量受到限制，以加速資訊的傳播；或者改良共識機制，例如：Po*、PBFT、非同步圖形共識 [62, 63]、隨機共識 [64] 等的使用，以加快區塊的產生。但是去中心化系統的核心概念是降低進入門檻，取消參與節點的限制，不會形成對信任機器的壟斷。在私有鏈中，節點數量較少，容易增加遭受所謂 51% 攻擊的風險或資訊的壟斷，而一些公有鏈開發者打算透過採用較少節點的架構來提高速度，這反而會使得區塊鏈容易受到 51%攻擊或分散式阻斷服務攻擊（DDoS）攻擊 [65, 66]，導致整個區塊鏈系統停止運作。

二、第二個問題是有效負載空間不足。

隨著大量的交易記錄被推送到公有區塊鏈上，區塊鏈上的數據量將在短時間內迅速增加。依據共識機制，區塊鏈的礦工必須儲存區塊鏈上的每個區塊及其包含的所有交易記錄。例如，比特幣的共識機制將區塊鏈容量的成長限制在每年 70 GB 左右。在沒有這些限制的情況下，區塊的傳播和儲存會是一個主要問題。這種情況也被稱為「區塊鏈膨脹」（blockchain bloat）[67]。

三、第三個問題是缺乏隱私保護。

儲存在公共區塊鏈中的所有交易記錄,都儲存在來自世界各地的礦工，所有的記錄，都是公開透明的，只有隱藏交易者的資訊，例如比特幣的 UTXO 架構。目前，公有加密貨幣區塊鏈中的隱私保護，主要有兩種方法：（1）加密累加器 [68]，有 Zerocoin 使用；（2）CoinJoin [69]，有 SharedCoins、Dark Wallet、CoinShuffle、Dash 的 PrivateSend 功能和 JoinMarket 使用。但是以上兩種方法，只能用於加密貨幣交易。而以太坊智能合約，是可以開發 Dapp 的，他能處理一般交易或作業資訊，而不僅僅是加密貨幣交易，其中包括資產交易、專利許可，以及簽訂合約書、文件與資訊記錄等等。如果資訊記錄沒有隱私保護，便大大限制了系統的應用範圍。

四、第四個問題是公有區塊鏈目前的可應用範圍很有限。

去中心化的概念在一些應用領域上已經被接受了：例如加密貨幣的發行與交易，現在可以完全使用去中心化的模型來實現。然而，人類的經濟活動，受到法律、習慣、原有制度、人際關係的影響，往往無法免除集中式的操作。例如：物流、金融系統、醫療記錄，以及物聯網（Internet of Thing，簡稱 IoT） [70] 中資料的收集和驗證、供應鏈管理、股權交易、社群網路軟體、電子病歷、小額支付或行動支付系統、資產交易和數位產品銷售...等等，幾乎所有這些應用程式，仍然需要集中式的代理商，或是中介者進行作業。如果公有鏈無法與具有類似中心化運作的產

業融合，那麼區塊鏈作為信任機器的使用，將受到極大限制。

例如：電子書、音樂、電影租賃或電子門票等數位產品來說，普遍是以線上平台來銷售。為了拓展銷售管道，擁有者通常會委託代理商，透過代理商的網路平台進行銷售;代理商向使用者收取費用並維護用戶的帳戶記錄。這些帳戶會定期提供給擁有者必要的資料，其中包含下載和相對的版稅等的詳細資訊。然而，由於帳戶是由代理人記錄和維護，如果版權擁有者無法驗證其真實性，例如，代理人的記錄可能包含由於系統錯誤而導致的意外遺漏，或者遭入侵者竄改，又或者代理人可能故意偽造或修改記錄，以減少應支付給版權擁有者的版權費，擁有者便無從追查是否有遺漏給付費用。

基於以上的說明，簡單摘要目前公有區塊鏈仍待解決的問題有以下四點：

一、如何增加區塊鏈的交易頻寬，及交易速率？

二、如何讓大量的交易資訊得以儲存在區塊鏈上，避免『區塊鏈膨脹』？

三、除了加密貨幣以外的其他交易資訊，儲存在區塊鏈上能否有隱私保護？

四、區塊鏈如何能有效應用在真實世界的各種作業或交易場景？

如果解決了上述的第一點，是不是會導致交易量大而使『區塊鏈膨脹』？又如果解決了第三點的問題，是否資產擁有者便無法追查費用是否遭到短報短支？這是否反而形成了「不可能的四角」（quadri-lemma）的問題了呢？

1.3.4 目前已有的解決方案與其問題

所以解決公有區塊鏈頻寬不足就是區塊鏈技術發展的重要議題，解決的方案主要有兩個大方向：一是直接嘗試改善主鏈運作，這要建立一個全新的區塊鏈，一般稱為 layer one scaling solution。另一個方法是鏈下來擴容，針對現存的主鏈，使用鏈下運作來加大區塊鏈處理交易的速度，稱為 layer two scaling solution [61]。以下分為兩大段來討論之。

一、Layer one scaling solution

1、改良共識機制：使用特殊的共識法，迅速的選出區塊產生者以增加 TPS。比較特殊的有：「非同步圖形共識」，如 IOTA、HashGraph，與「隨機共識」，如 Algorand 的隨機演算法。[62 - 64] 但是只要節點數目多，擴容效果還是有限，因為主要瓶頸還是在網路上擴散交易給眾多節點。

2、分片（sharding） [71] ；將區塊鏈分成幾個獨立運作的「分片」（shard），平行運作，參與的礦工被分配到不同的「分片」，而「分片」間不太會有資料的交換。整個系統的 TPS 就是各個「分片」之 TPS 總和。但是為了解決 double spending 的問題，參與者的資產要被分配到不同的「分片」中，沒有辦法一次將不同 Shard 的資產一次轉給某人，資產在不同「分片」之間轉換需要花很多時間。一個智能合約只能在某一個「分片」之中，所以在不同「分片」的

智能合約無法溝通，其中的資產也不能轉換。如果同時分片量大，每個分片要募集足夠的礦工來抵抗 51%攻擊也是很困難的。

二、Layer two scaling solution

在主鏈的技術、架構、礦工等都沒有更動的情況下，在鏈下運用一些方法來加快整體能在主鏈上存證或證明的交易數目，目前有的技術種類很多，茲說明如下：

1、側鏈（side-chain）技術 [72]，也就是將資產先轉到其他區塊鏈，也就是私有鏈，在其他區塊鏈先處理後，再將結果轉回主鏈，也就是公有鏈。但是側鏈是個私有鏈，需要募集礦工，很難有全球共識。另外，有的是事先將交易記錄在側鏈，累積一段時間後，將一些區塊的雜湊值放到公有區塊鏈上存證，稱為錨定（anchoring）。這種方式的問題是要驗證側鏈的交易時，必須將所有於側鏈累積的大量交易交給驗證方，驗證方根據錨定的雜湊值進行確認，一般情境很難實現。目前 BlockChain Security Inc.（區塊科技公司）[73] 使用這種技術來進行資料存證。

另外，也有所謂的「relay-based」的側鏈技術，如 BTC-relay [74] 、Rootstock [75]，他的問題在於需要用 2-way peg 的協議機制來互相轉換，但比特幣區塊鏈這一方無法建立這種 relayer；另一種稱做「channel-based」 的側鏈技術，如 Lightning-network [76]、Raiden

[77]，是利用比特幣的擴充套件，其問題詳見下一段的說明。

2、以下說明數個對微支付(micropayment)需求進行鏈下擴容的技術，這些方式對資料存證的應用無法支援。

- Lightning Network/Raiden Network：這是被稱為 state channel 的鏈下支付系統，首先先將要支付的貨幣鎖在一個 channel 中，指定要支付給某固定位址，然後慢慢發簽章支付款項。幾本上只能做一對一的支付，這是最大的問題及限制。為了解決這問題，想出使用 hub 代付的方法。但是 hub 必須隨時在線，若是 hub 中的儲金不夠，也無法順暢運作。同時運作時要將 state channel 開開關關也會消耗主鏈的頻寬。
- Plasma：以太坊創始人 Vitalik Buterin 於 2017 年 8 月 11 日提出 Plasma 白皮書 [78]。參與者在鏈下先利用電子簽章交換支付代幣的證據，由 operator 定期打包上鏈。要交換的代幣必須先儲存到 Plasma 智能合約中，要由此智能合約提幣，要先提出請求，如果有 double spending，則由其他參與者提出證據取消提幣。但是 Plasma 如果發現 operator 有問題，可能無法提出證據證明，因為智能合約的證據大小有限制，參與者只能集體出逃(白皮書上稱為 mass-exit)，但是逃的晚的，一定會損失。因此 Plasma 有以下問題：

- 無法即時出金，要等一個星期。
- 有可能出問題時智能合約無法處理，只能集體出逃。

- Plasma Cash：因為 Plasma 的問題，基本上無法使用，所以 Vitalik Buterin 於 2018 年 3 月於法國 Paris 提出 Plasma Cash [79]，這是 Plasma 的升級版本。為了解決無法用密碼學證據證明有人作弊，限制要交換的代幣的面額（denomination）無法切割或合併。同時收幣者要由送幣者接受所有該幣交易的歷史記錄，所以密碼學證據會持續膨脹。Plasma Cash 有以下問題：
  - 無法即時出金。
  - 交換的代幣的面額（denomination）無法切割或合併。
  - 密碼學證據膨脹，交換幣的速度會受影響。
- zk-rollups：基本上和 Plasma 及 Plasma Cash 一樣，要交換的代幣要先儲存到一個智能合約。但是隨後在合約中的代幣交換是運用 zk-SNARK [80] 此零知識證明的技術來減少驗證交易所需要的礦工費。一個以太坊的交易需要 109 bytes 存在某區塊中（ECDSA 數位簽章佔 64 bytes、receiver address 20 bytes、其它佔 25 bytes）。但是使用 zk-rollups 只要將 8 bytes 存在合約中，sender 及 receiver 因為有對照表所以不用完整的 address。但是怎麼證明轉幣有收到 sender 的電子簽章呢？這是讓 operator 使用

zk-SNARK 的方法產生 proof（P 演算法），然後讓合約來執行 zk-SNARK 的 verification（V 演算法）。P 演算法需要大量的算力，但是在鏈下執行，產生的 proof 很小。V 演算法針對此 proof 算量很小，所以在智能合約上執行不會消耗太多 Gas（大約為 500,000 Gas cost）。

3、運用 batch compression 產生密碼學證據，將證據於公有區塊鏈存證以達到擴容的效果。這個方向可以支援區塊鏈資料存證。

- Microsoft Sidetree：這是 Microsoft 公司對於 decentralized ID 這個應用提出的擴容方案，將大量收集的數位 ID 使用 Merkle tree 進行 batch compression 後將最後的根雜湊值於比特幣的區塊鏈存放。這個擴容的方法於驗證存證資料時，必須進行 batch decompression，所有被打包的資料要交給驗證方，驗證方根據存證的雜湊值進行確認。一般情境很難實現，目前沒有產品正式問世。
- Cryptoewerk Horizon：這是 Connecting software 公司提出的擴容存證方案，基本上和 Sidetree 很像，驗證存證資料時，必須進行 batch decompression。支援驗證的 seal 只能針對單一筆資料進行驗證，無法驗證不存在。

第四節 研究動機與目的

「美國國家標準與技術中心」(National Institute of Standard and Technology，簡稱 NIST）於 1977 年的文件中提出的資訊安全三要素：CIA triad，即「機密性」(confidentiality)、「完整性」(integrity)、「可用性」(availability）[81]。「國際電信聯盟」(International Telecommunication Union，簡稱 ITU）於 X.800 標準協議所稱之網路安全所需達到的目標，有：機密性（confidentiality)，完整性（integrity)，可用性(availability)，授權管理(authorization 或稱 access control)，隱私性（privacy)，與不可否認性（non-repudiation）等六項 [82]。

而不僅僅是過去所提的的資訊安全三要素 CIA （confidentiality、Integrity、Availability）與網路安全六要素的問題而已。

其中應該包含有：

一、資訊交換過程與結果的確認，或稱為鑑別性 （authentication)，包含：

- 身分的確認
- 作業過程的確認
- 作業內容與結果的確認

以確保這筆資訊是真實且有效的（validity)。

二、作業或交易過程與結果資訊的可用性、機密性、完整性。確保這筆資訊是安全的（security)。

三、作業或交易過程與結果的不可否認性與可歸責性（accountability)，確

保資訊不會遭受破壞、日後得以追蹤並驗證(auditability)。其中包括「是否有」或「是否沒有」這筆資料。

要達到以上目的，需要資料存證。為了使得資料存證，能在各種的應用情境下，有效而快速的稽核，以達到作業結果的不可否認性與可歸責性，Hwang et al. 提出了「以定位摩克樹為基礎的違約證明」(tp-Merkle tree based Proof of Violation) 協議機制 [83]，解決資料存證的安全性與效能問題。但是資料存證要如何存放？是否有機會遭到竄改或破壞？這是資料存證的一大問題。

1.4.1 需要有安全的資料存證空間與快速稽核機制

如同本章第二節所述，如果要將資料存證放在可信任的第三方，所謂「可信任的第三方」有可能被質疑，也可能成為存證資訊遭到竄改或破壞的風險所在，若存證的資訊遭受竄改，則難以究責。因此筆者可以運用 Web3.0 的核心技術：去中心化的公有區塊鏈來存證，以防止存證資料遭受竄改、阻卻外來攻擊。

公有區塊鏈除了礦工是分布在全世界外，區塊鏈上的交易紀錄，也是來自全世界的交易，因此可以想見其資訊量與來源相當的多與雜。為了能夠方便搜尋相關資料，區塊鏈上的每一筆記錄，筆者通常會以 **key-value pair** 的結構儲存：

Blockchain record：< key，value >

key 是做為調閱資料的索引值，可以依照需要，沒有限制自行定義，value 是要存放的資料，例如：一筆交易資訊。

就以太坊的智能合約為例，見圖 3，智能合約包含三個部分：(1) 金庫

（deposites）（2）函式（functions）（3）交易記錄（key-value pairs）。

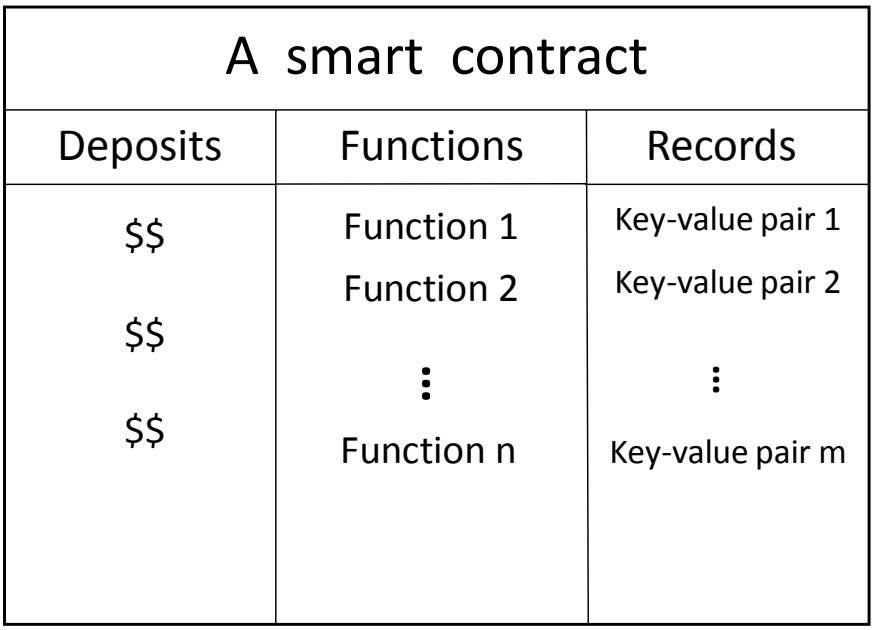


圖 3：以太坊能智能合約示意圖

但是公有區塊鏈存在著四大問題，如前面 1.3.3 節所述，如果我們將所有作業的存證資訊，即 key-value pair，皆打上公有區塊鏈，每打上一筆證據，也就會形成一筆區塊上的記錄，會收一次的礦工費；以近幾年平均的礦工費與以太幣（ETH）的美金價格來看，請參考表 1。

表 1：公有區塊鏈作業價目表

Aumme 1 ETH = USD 1,000

Aumme Gas price = 50 Gwei　　1Gwei = 10^{-9} ETH

	建立 ERC11155 合約	**Mint a token**	**ETH transfer**	**ERC20 transfer**	**Key-value pairs**
Gas consumption	4,222,561	77,420	21,000	51,000	100,000
ETH	0.21112805	0.003871	0.00105	0.00255	0.005
USD	$211.1	$3.9	$1.1	$2.6	$5.0

近兩年來，參考 blockchain explorer 的資訊，一個以太幣的價值，大約在 1,000 美元上下；而以太坊區塊鏈的礦工費以 Gas 計費，Gas 的價值會波動，使用者也可以提高 Gas 的價格（出價），讓自己的作業或交易優先被區塊鏈的礦工處理。

我們不難想像，光是一段簡單的日常交易，就可能產生來回上百次的交談訊息，為了防止日後發生爭議的歸責問題，每一個訊息都應存證，若全部打上公有區塊鏈儲存，那麼將會產生上百筆的礦工費，以表 1 來看，一筆 key-value pair 需要 5 元美金，假定這次的作業有 500 筆資料要存證，所以礦工費：500 × USD5 = USD2,500！極高的成本。日後若要稽核，一次多筆資料的查詢與讀取，雖然不需要礦工費，但是往往會讓礦工以為是遭到分散式阻斷攻擊（DDoS）而遭終止，所以這種方式顯然會造成大量資料存證與稽核上的困難。

因此，本論文使用「以定位摩克樹為基礎的違約證明」協議機制，證明了可以在兩種實際應用的情境下，達到安全而有效率的大量資料存證與稽核的目的，並可以解決上述的各項問題。

第五節 研究成果

本論文以「定位摩克樹」做為資料存證的機構，並研究了在兩種實際情境下的存證應用：

1、雲端系統執行環境的即時稽核（Real-time Auditing of the Runtime Environment for Cloud Computing Platforms）。

本項研究發表於期刊 JISE 2019。[84]

2、利用公有區塊鏈提供自動給付與賠償機制(Automatic Reward System Based on Public Blockchains)。本項研究發表於國際學術會議 IEEE ECICE 2022。[85]

以上的兩項研究，分別證明了在該情境下，能夠有效而快速的確認作業過程的不可否認性與可歸責性，阻卻外來攻擊。除了可以即時而快速的驗證作業內容外，防止因遭任何一種外力的資訊破壞，並可即時察覺，也可以提供有效的「詐欺證明」(fraud proof)，做為究責之用，並且可解決公有區塊鏈的擴容與隱私保護問題，擴大了 Dapp 的應用範圍。

就這兩項研究成果，於以下兩節分別摘要說明之。

1.5.1 雲端服務平台的稽核應用

資訊系統，除了應用軟體程式外，還包含作業系統，與其運行時所需要的大量動態函式庫，以及所需的硬體環境。

雲端服務（cloud services）有「軟體即服務」(Software as a Service，簡稱 SaaS)、「平台即服務」(Platform as a Service，簡稱 PaaS)、「基礎即服務」（Infrastructure as a Service，簡稱 IaaS）[9] 等三大類，目前雲端服務供應商所提供的雲端租賃服務已不勝枚舉，已涵蓋了大部分資訊系統運行所需。也因為雲端服務可免除自行建置與維護運行的大量成本，並可擁有最新最適的版本可供使用，也可免除需要規劃升級的成本，因此大受歡迎。

在 IaaS 與 PaaS 的服務類型中，向雲端服務供應商所租用的服務，我們通常

稱為「虛擬機器」(virtual machine，簡稱 VM)[86, 87]。對雲端服務供應商而言，理論上會確保 VM 的安全性與可用度，可是對使用者而言，會擔心系統是否會遭到竄改、破壞或遺失資料，以及服務提供商是否會誠實的面對服務端的失誤？另外有一種狀況，就是該租用平台遭到毀損，服務提供商使用了先前的備份回復系統，換句話說，用戶最新版本的資料可能因此遺失了，若是牽涉到軟體執行環境的版本，會造成很多的問題；服務提供商使用了先前版本的備份回復系統，這種情形我們稱之為「roll-back attack」[88]或稱為「replay attack」[89]，如果用戶無法即時發現這種情形，必然會造成用戶使用上、或在虛擬機器上執行軟體上發生諸多錯誤與問題。另外若是因為租用平台遭到電腦病毒感染，目前也沒有任何一種防毒軟體可以解除未知病毒。

因此筆者提出了一個創新的架構，能夠在雲端服務平台（也可以運行在單獨的資訊電腦系統，或雲端伺服器）執行即時且快速的執行環境的稽核，也就是在資訊系統執行環境中，當應用軟體執行前，該應用軟體執行時所需的相關檔案與其所需的動態連結檔（dynamic linked library files），會立即的完成該執行環境的完整性檢查（real-time integrity checking）[90]。

過去的相關研究中，有一種直覺式的解決方案，是將資訊系統的執行環境的所有檔案的雜湊值存放起來，當應用軟體執行前，先檢查所有檔案的雜湊值是否有與原先存放的雜湊值相符，以完成完整性的檢查 [91] 。以蘋果電腦（Mac）為例，若使用 Mac OS X E1 Capitan 10.11.5 版本，並安裝基本系統與常用的軟體，

則將會有 149,487 個目錄與 717,976 個檔案，如果將這些檔案取雜湊值存放在「遠端可信任的電腦」中，於每次執行軟體前，先檢查所有相關檔案的雜湊值是否有與原先存放的雜湊值相符，筆者發現，這種完整性檢查，會是相當沒有效率的，而且若遭受病毒感染或駭客破壞，原先檔案的雜湊值也可能遭到竄改，無法確保完整性檢查的可信度。而這種做法，很明顯的也無法防止「roll-back attack」，因為當系統還原成舊版本時，其相對的雜湊值也還原成舊版本了，因此也無法察覺系統是否已遭更動了。今天若是將所有檔案的雜湊值放在公有區塊鏈上，是不是可以解決這個問題呢？

如果我們將所有執行環境的檔案的雜湊值的 key-value pairs 全部打上區塊鏈，也就是 717,976 個檔案，參照表 1，一筆 key-value pair 需要 5 元美金，所以礦工費會是：USD5.0 × 717,976 = USD3,589,880 這樣的天價！而且，當相關檔案的版本更動，其相對的 key-value pair 也要更新，如此不僅成本太高，稽核時，要在區塊鏈上一次存取大量的 key-value pairs 也不可能做到。

因此，筆者運用「定位摩克樹」提出一個創新的、可即時且快速稽核 VM 執行環境完整性的架構，本論文於 2019 年當時發表的研究 [84]，將定位摩克樹稱為「完滿二元雜湊樹」（full binary hash tree，簡稱 FBHT）具有索引函數功能，FBHT 所需的空間也相當少（請參照 3.3.4 節，表 2）。在雲端的 VM 中，筆者運用一個代理程式，先將所有執行環境的檔案的雜湊值以「定位摩克樹」存放起來，而這些雜湊值的「狀態碼」（*status code*）存放在遠端電腦或公有區塊鏈上，以做

為該 VM 環境的即時稽核之用。「狀態碼」即是「定位摩克樹」的根雜湊值（root hash），所占空間很小，只有 32-byte。若執行環境的版本更動，只需要更新狀態碼存放在公有區塊鏈的 key-value pair 即可。

當應用軟體執行前，會先下載 32-byte 的狀態碼，代理程式會將所有檔案的雜湊值計算出來並產生所對應「定位摩克樹」的根雜湊值，最後比對根雜湊值與狀態碼是否相符即可。詳細的系統架構與運作方式，請參照第四章第一節的詳細說明。

筆者模擬真實的環境進行測試，實驗結果證實這個方案確實可行。

1.5.2 利用公有區塊鏈提供自動給付與賠償機制的應用

在雲端的交易環境中，單純的客戶與供應商間的交易是最簡的，例如雲端硬碟租用。而現實環境中的交易，經常會牽涉到多方的交易互動。例如版權交易：作者會透過代理商銷售，客戶向代理商購買；作者如何確保代理商回報的交易數字是正確的？如果違約，如何提出「詐欺證明」（fraud proof）要求賠償？

公有區塊鏈具有加密貨幣的交易功能，並且提供了可信任的去中心化的平台，因此又稱為「分散式帳本」。公有區塊鏈記載的交易內容公開透明，難以被竄改，因此可以提供交易多方一個可信任的平台。

在交易支付系統上，通常最大的問題在於代理商與資產擁有者，或者版權擁有者之間的信任問題，代理商使否有誠實地依銷售量支付應給的版稅？一般的資訊化作業並無法解決這個問題，因為操作方是代理商，版權擁有者無法稽核。

為了解決這些問題，讓公有區塊鏈能夠應用在微支付的交易上，並讓多方交易作業安全且在多方都可信任的環境下進行，也就是同時需要有快速稽核機制，而讓多方都能夠提出「詐欺證明」(fraud proof)，因此，筆者提出了一個自動給付與違約賠償的架構，足以解決現實環境中的多方交易模式。詳細的系統架構與運作方式，請參照第四章第二節的詳細說明。

筆者模擬真實的環境並進行測試，實驗結果證實這個方案確實可行，而且效率極好。

第六節 論文架構

本篇論文的第二章，就傳統資料採證的方式做一般性的介紹；第三章介紹過去研究中的資料採證與驗證方式並說明其問題，接著說明本研究所使用的存證方法「定位摩克樹」，並做詳盡的介紹，以及實際執行效能的實驗結果，並介紹「定位摩克樹」存證的相關研究成果；第四章將就本研究所使用的方法，提出兩個應用架構，並詳細說明其運作原理與實驗結果；第五章將就本研究總結並提出未來延伸研究的方向。

第二章、資料的採證與存證

第一節 資料存證

為了確保網路交易的可信度，就必須在交易經過的每一個部份，進行存證（attestation），日後才能有足夠的證據去驗證該交易是否正確或有效，如果證據資料足夠，甚至連哪一個環節出問題，都可以驗證的出來；除此之外，存證的資料，為能提供日後的追蹤、稽核，因此必需有不能被破壞、竄改（tamper）或遺失的特性（properties），此即為所謂的資料存證（data attestation）。[13, 14, 92]

資料存證技術的效能（performance），也直接影響著稽核驗證（auditing）[93]與鑑識（forensics）[14, 105]的效果。

第二節 傳統的資料採證方法

2.2.1 雜湊

「雜湊」（hash），是對資料的一種處理方法。將任何大小的資料，經過「雜湊函數」（hash function）的運算，會得到「固定長度的值」，這個值我們稱為「雜湊值」（hash value），或稱「信息摘要」（message digest），可以視為該資料信息的「指紋」（fingerprint）。而產生雜湊值的雜湊函數必須具有下列兩個性質：

（1）雜湊值無法經過雜湊函數反算出明文。

（2）明文資料變更，其雜湊值也會跟著變更（不同）。

如果不同的明文資料經過雜湊函數卻產生出相同的雜湊值，我們稱之為「雜湊碰撞」（hash collision），如果雜湊函數的「值域」不夠大，則容易造成雜湊碰

撞，而這種雜湊函數，通常無法使用在應用領域上。因此，我們在設計雜湊函數時，必須注意所使用的演算法，必須有足夠大的值域，使任何輸入的資料，能產生唯一的雜湊值，這種雜湊值，我們則稱之為「資料指紋」(data fingerprint)。雜湊值也可以應用在資料的檢索，以及資料的驗證領域上。

2.2.2 數位簽章

所謂「數位簽章」(digital signature)，是於 1976 年由 Whitefield Diffie 與 Martin Hellman 所提出的 [22]，其中包含數位簽章所需的密鑰交換系統技術[94]。數位簽章的目的，是為驗證資料或訊息傳遞的確認性，也就是用以確認這個訊息不是假冒或偽造的。雜湊函數也經常使用這類的加密演算法，或者是被用來作為數位簽章的方法。

例如：「再雜湊」或稱「雜湊鏈」(hash chain)。

2.2.2.1 雜湊鏈

將雜湊函數重複地用於同一明文或數位簽章，即將所得雜湊值再次傳遞給相同的雜湊函數計算其雜湊值。

例：$\boldsymbol{m}$ 代表明文，$\boldsymbol{h}(\boldsymbol{h}(\boldsymbol{h}(\boldsymbol{h}(\boldsymbol{h}(\boldsymbol{m}))))$ 代表對明文 $\boldsymbol{m}$ 做了 5 次雜湊，我們記為 $\boldsymbol{h}^5(\boldsymbol{m})$。雜湊鏈通常是用在身分的識別上面，我們稱為「鑑別」(authentication)。

2.2.3 以雜湊樹存證

「雜湊樹」又稱「摩克樹」(hash tree 或 Merkle tree) 是一個樹狀的資料結

構，每一個「節點」(node) 儲存雜湊值。「葉節點」(leaf node) 用來儲存資料的雜湊值，同源相臨的「子節點」(sibling node)的雜湊值「串接」(concatenation) 再取雜湊值，成為其「父節點」(parent node) 的雜湊值，如此由下往上用相同的雜湊函數計算出「根節點」(root node)的雜湊值，我們稱為「根雜湊」(root hash 或 master hash 或 top hash)。如圖 4。

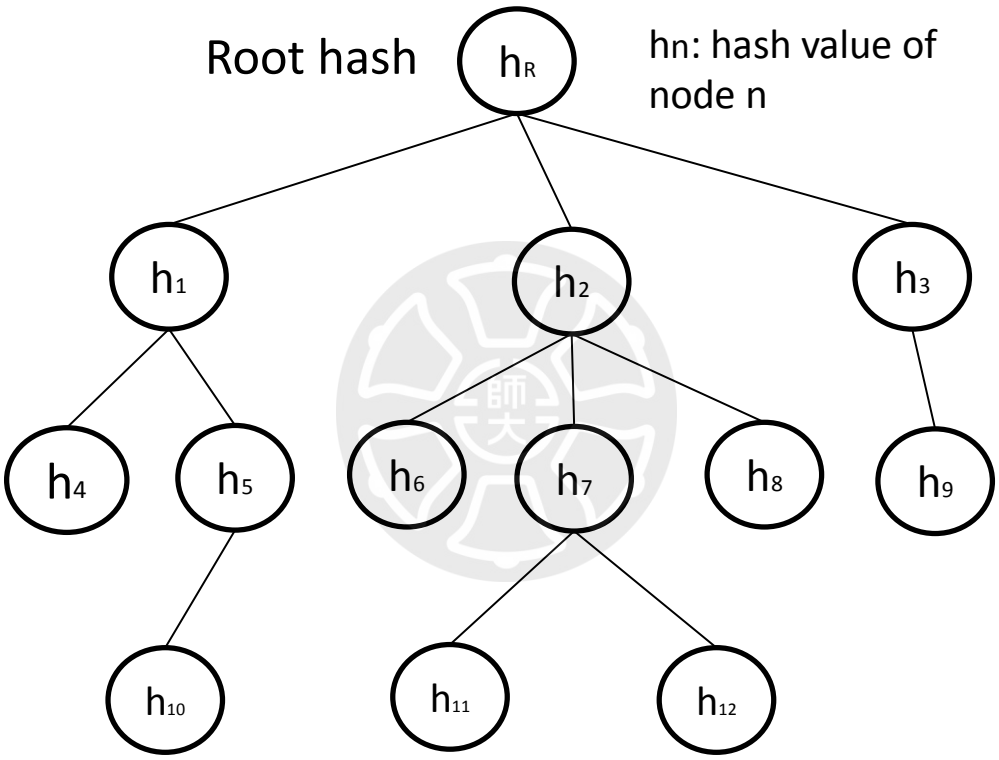


圖 4：Merkle tree。

這種雜湊的樹狀資料結構，是由瑞夫・摩克（Ralph Merkle）於 1979 年的博士論文 [45]中提出的，做為在「公鑰系統」(public key infrastructure)、或其他加密技術，記錄數位簽章或信息摘要之用，並且可以快速的檢核資料的完整性，因此又稱為「摩克樹」(Merkle tree)。「摩克樹」可以是多元的（m-ary)，也不一定

「完滿」（full）;「完滿二元雜湊數」（full binary hash tree，簡稱 FBHT）可以視為「摩克樹」的一個特例。

筆者以符號"||"表示 hash 值串接（concatenation）:

$h_5 = hash(h_{10})$　　$h_1 = hash(h_4||h_5)$

$h_7 = hash(h_{11}||h_{12})$　　$h_2 = hash(h_6||h_7||h_8)$

$h_3 = hash(h_9)$

所以 root hash = $hash(h_1||\ h_2||h_3)$

$= hash(hash(h_4||\ hash(h_{10}))\ ||\ hash(h_6||\ hash(h_{11}||h_{12})||h_8)$

$||\ hash(h_9))$

h_4、h_6、 h_8、 h_9、 h_{10}、 h_{11}、 h_{12} 任何一筆交易資料改變，其 hash 值就會改變，所以 root hash 值也會跟著改變，所以只要保有 root hash，就可以立即地驗證資料的完整性。

如果只要驗證局部資料是否正確，例如 h_{10} 的資料是否有遭受竄改或變更，只要下載與追蹤該資料的子樹片段做檢查即可，該「單一資料的子樹片段」，我們又可稱為「partial Merkle tree」（簡稱 pMT）、「Slice of Merkle Tree」（簡稱 Slice）或稱為「Merkle Proof」。如圖 5，h_{10} 的 Merkle proof 所得到的 root hash，一定與整個雜湊樹的 root hash 相同，因此只取出 Merkle proof，不需要檢驗所有存證的資料，所需要的空間更小，因此效率更快。

特別的是，無論是 m-ary 摩克樹、完滿二元雜湊樹，或者是使用 Merkle proof，

如果是被駭客植入木馬程式，都無法只取出 Merkle proof 稽核出來，因此無法察覺。如果是遺失檔案，則需要下載整個 Merkle tree 稽核才能察覺，因此無法只使用 Merkle proof 發現。這是因為，當我們取得資料的雜湊值以摩克樹存證時，會依序或隨機將資料指紋存放於葉節點，我們會知道每一個葉節點的駐標（subscript）以及資料指紋存放的駐標，但是偽造的資料指紋與其駐標也會偽造，因此單就 root hash 或 Merkle proof 的驗證是無法發現的，唯有將所有資料的指紋全部重新計算，並與原存證的整顆摩克樹進行比對驗證，才有辦法發現，所以當有大量或是長時間累積的履歷資料的存證，若使用 Merkle tree 來存證，並無法有效率的稽核，甚至在很多現實的情境下是無法實作的。

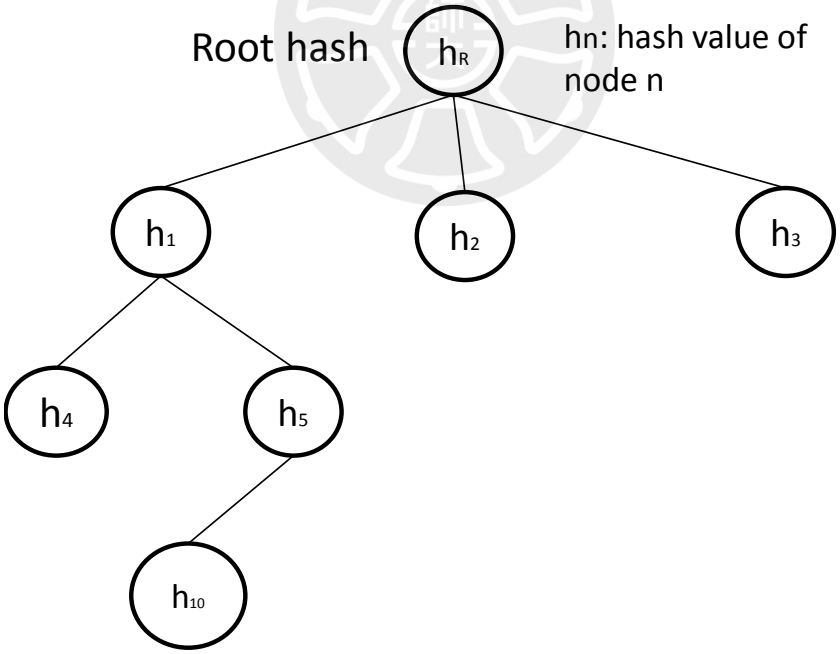


圖 5：h_{10} 的 Merkle proof

因此，使用摩克樹存證，以 root hash 驗證，只能驗證存在（proof of existence），無法驗證不存在（proof of inexistence）。

第三章、研究的資料採證與存證方法

第一節 C&L scheme

Hwang et al.（2013）[95] 提出的方法，應用於使用雲端儲存空間檔案的存取與更新，及其使用過程的資料存證與稽核，並利用摩克樹存證與重置存證空間，以便減少服務方與使用方所需的存證空間，提高稽核效率。

C&L scheme，是一種在交易過程中，用以互相確認所有過程是否存在的「協定」（protocol），是一種「4-step handshaking」，做到交易過程的不可否認性。

首先服務提供方依使用者上傳檔案的目錄結構，建立初始化的檔案雜湊樹，如圖 6，此時為初始化 epoch。每開始一段使用該服務的過程，會有一個 epoch id.，經過一系列的使用與更新資料過程後，由服務提供方儲存所有交易的證據，用戶端則只須存放最後一個證據與雜湊樹的 root hash，以避免用戶端存證資料過於龐大，例如手機的儲存空間較小。

每租用一項服務，一定需要註冊一個帳號用以存取服務，但 C&L scheme 允許不同「用戶端設備」（client device）使用同一個帳號做存取。當作業開始，每一個用戶端設備，都會有自己唯一的 client id，與屬該 client id 作業（operation）順序的序號，以區分不同用戶端設備使用相同的帳號存取，稱為「local sequence number」（簡稱 LSN），與「epoch id」，epoch id 用以宣告連續作業的階段，每一項作業中，所傳遞的執行訊息，均以 chain hashing 做數位簽章認證，以確認每個作業過程是有效的。當服務端因為存證資料累積太大，或其他需宣告該 epoch 結

束前，服務端會將新的結果，也就是依新的檔案目錄結構，建立新的相對應雜湊樹，同時用戶端也執行相同動作，經用戶端比對兩方雜湊樹的 root hash 是否相等，若相等，則通知服務端可以捨棄所有 chain hashing 證據，釋放存證空間，並由服務方宣告新的 epoch 開始，所有的登入用戶端的 LSN 重置為 1 開始。

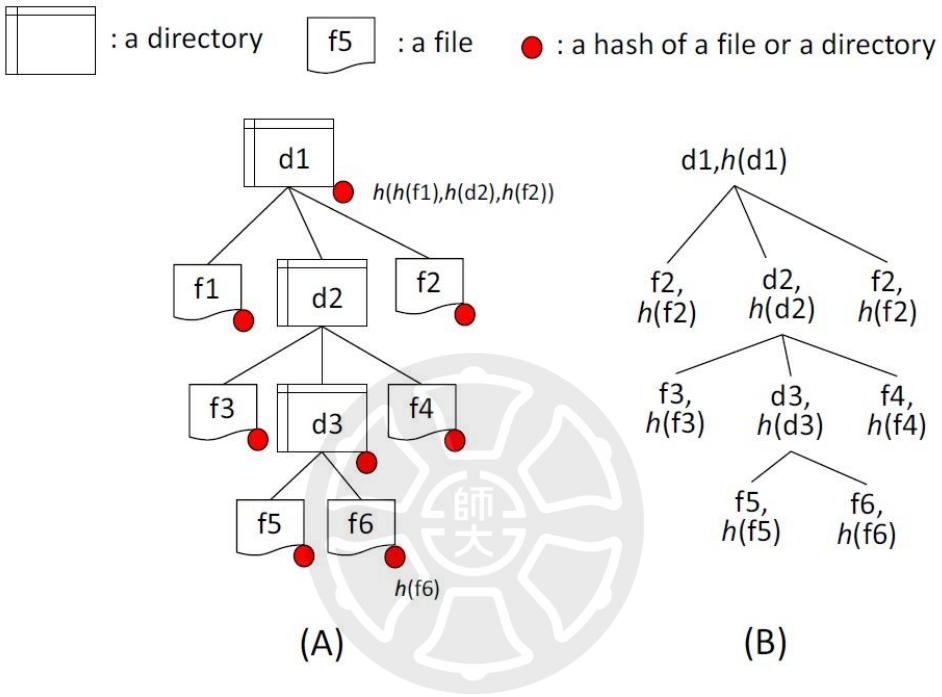


圖 6：以雲端硬碟為例，用以資料存證的雜湊樹（取自[95]）

由於利用 chain hashing 的數位簽章與 LSN 達到「相互不可否認性」(mutual non-repudiation）的效果，因此該協定稱為「C & L scheme」；該協議為提升存證以稽核效能，於每一個 epoch 中，用 m-ary 雜湊樹儲存所有經過確認過的作業所更新資料的指紋，以提升稽核效率。後來該協定稱又稱為「epoch-based Proof of Violation」協定。

C & L scheme 這個機制的主要問題，除了無法做到即時稽核與驗證外，每次的稽核驗證必須經過一段時間一連串的作業後(即所謂 epoch)，才能啟動稽核，

且每一次的稽核必須下載整顆摩克樹。

這是 Hwang et al.利用雜湊樹做資料存證，以達到雲端交易的相互不可否認性，因為 C & L scheme 無法做到即時稽核與驗證，因此之後 Hwang et al.又提出一個加強版的 C & L scheme，稱為「Proof of Violation」協定，簡成稱「PoV」協定，或稱「real-time PoV」協定，可以做到每一次作業後的立即稽核驗證。

更新版本的「real-time PoV」，詳見下一節的介紹。

第二節 Proof of Violation

所謂的 Proof of Violation 機制，包含有 three tuples：(1)predefined security properties；(2)predefined cryptographic proofs；(3)auditing scheme。

由於前一節所述的 epoch-based PoV 協定，除需要保存大量連續的密碼學證據以供驗證之用外，每一次的稽核驗證，必需下載整個依照現存檔案目錄結構所建立的雜湊樹，才能啟動稽核，盡管這當中包含大部分沒有異動過的資料，也必須重新計算 hash 值，所以相當沒有效率，也因此無法做到即時的稽核與驗證。於是 Hwang et al.（2014）[96] 又提出了一個新的方法，稱為「real-time PoV 協定」，該方法取消了 LSN 與 epoch id 的 properties，改用「同步伺服器」（synchronization server，簡稱 SYS）與「稽核伺服器」（auditing server，簡稱 AS），提供雲端硬碟服務的伺服器，稱為「儲存伺服器」(storage server，簡稱 STS)；用戶端設備只需存放最後一個證據與有參與更動過的檔案的 partial hash tree，即可以做到即時的稽核驗證。系統架構如圖 7。

其中，SYS 伺服器與 AS 溝通，並負責共用同一個雲端服務帳號的用戶端間，證據交換的工作，包含最後一個證據（chain hashing）與檔案系統雜湊樹的 root hash；AS 則負責存放所有交易資訊的證據，與整個檔案目錄結構相對應的雜湊樹，該雜湊樹的內容會經常異動，所有交易資訊的證據也會不斷累積，因此 AS 將會是經常異動且存放所有不斷累積資料存證的伺服器。

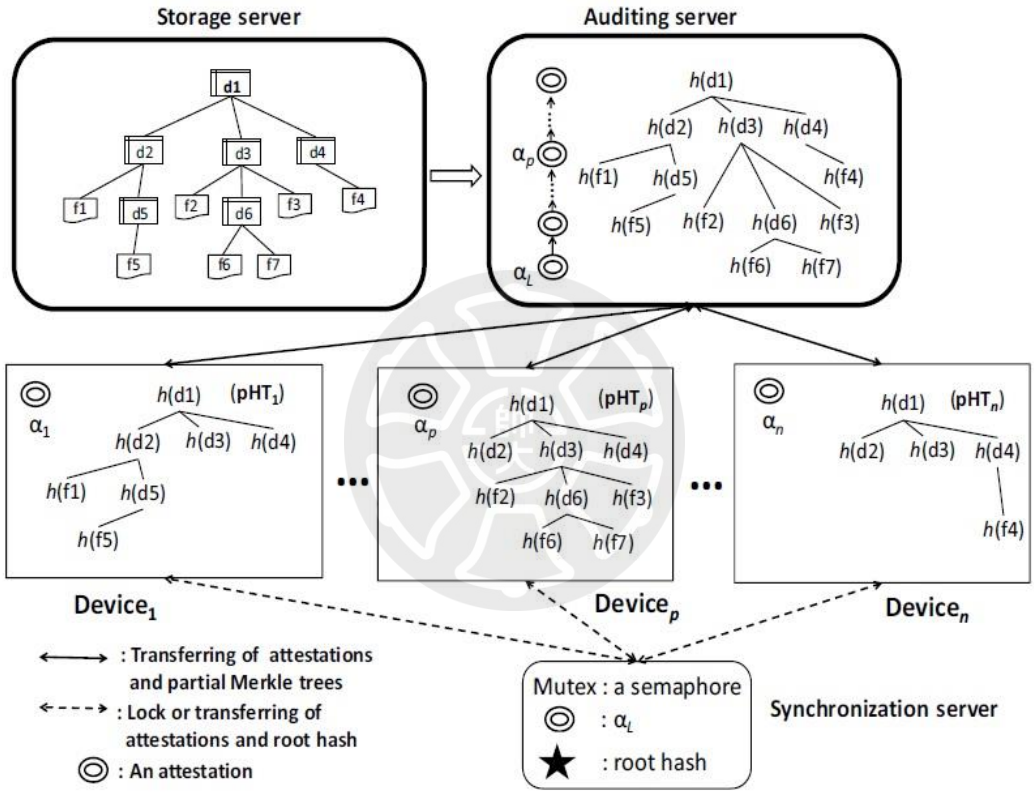


圖 7：Real-time PoV 的系統架構圖（取自[96]）

因此理論上，SYS 與 AS 都應該放在「可信任的第三方」中，以免存證資訊遭到竄改，但是由於其功能性的分配，SYS 可以交由用戶端的設備來管理，AS 則是需要一個「可信任的第三方」，如同第一章第二節所論及「可信任的第三方」反而可能是問題的所在，但若是將 AS 的資料打上公有區塊鏈，由於 AS 會是經常異動且存放所有不斷累積資料存證的伺服器，若由公有區塊鏈的礦工負責並將

存證資料打上公有區塊鏈，不僅速度慢，造成所謂「blockchain bloat」，參照第一章第四節的表 1，並會產生天價的礦工費，這會是「real-time PoV 協定」最大的問題。

第三節 定位摩克樹

前兩節介紹的「C & L scheme」與「real time PoV」，是 Hwang et al.先前所提出的方法，應用在雲端資料儲存的稽核與驗證，與一些類似的研究一樣，採用摩克樹來儲存交易資訊與資料的指紋，用以達到服務的安全性與不可否認性的目的，同時可以抵抗「roll-back attack」[88]與「replay attack」[89]。但是當雲端儲存空間如果被植入或嵌入一筆偽造的交易或是資料時，這兩種方法或其他使用摩克樹方法的類似研究，皆無法立即察覺，必需要下載整個摩克樹並針對所有葉節點檢驗，才能驗證出實際上不存在、或者是發現有被植入偽造的資訊；換句話說，驗證時必需下載整個摩克樹進行驗證，這會花費相當大的時間與空間，是非常沒有效率的。

Hwang et al.首先在 IEEE CloudCom 2016 的國際學術會議上，發表研究篇名為「Efficient Real-time Auditing and Proof of Violation for Cloud Storage Systems」，首先提出一個創新的資料結構，稱為「具有索引功能的二元完滿雜湊樹」（full binary hash tree with index function）[97]，同時在 ICBC 2018 的國際學術會議上發表研究篇名為「InfiniteChain: a multi-chain architecture with distributed auditing of sidechains for public blockchains」，以相同的資料結構，改稱為「索引摩克樹」（indexed Merkle tree）[98]，提出一個解決公有區塊鏈擴容問題的方案，並提出分散式稽核的概念。於 2020 年開始，正式定名為「定位摩克樹」（transaction-positioned Merkle tree，簡稱 tp-Merkle tree），所以「定位摩克樹」

是基於摩克樹，加上定位的功能的一種完滿二元雜湊樹的資料結構。

每一筆「交易紀錄」都包括有兩個部份：一個是用來決定存放在「定位摩克樹」位置的「索引值」(index value)的雜湊值，另一個是「交易資料」(data)的雜湊值，我們稱之為「key-value pair」; key-value pair 的資料結構表示為：<hash(key), hash(data)>，「key-value pair」的「key」是用來定位的「索引值」(index value)，「value」是「交易資料」(data)的雜湊值，皆採用 SHA-256 演算法取雜湊值，得出之雜湊值大小為 256 bits 即 32 bytes，因此每一個 key-value pair 的大小則會是 64 bytes。每一筆交易定位之後，取其 key-value pair 的雜湊值存放在定位的葉節點，因此每一個葉節點也會是 32 bytes 的大小。

將「交易紀錄」的索引值「key」放進定位函數Γ，則會得出該「交易」的「key-value pair」應存放在葉節點的位置。Γ函數表示式如下：

$$\Gamma(indexValue) = SHA256(indexValue)\ mod\ 2^{N-1} \quad \text{公式（1）}$$

N：表示 tp-Merkle tree 的樹高。

Γ函數計算後得到的葉節點位置，由於可能會發生葉節點位置的碰撞(collision)，因此當發生碰撞的葉節點，我們則以串列資料結構存放 key-value pair 的雜湊值，該葉節點則會存放這個 list of key-value pairs 的雜湊值連接(concatenated)後的雜湊值。

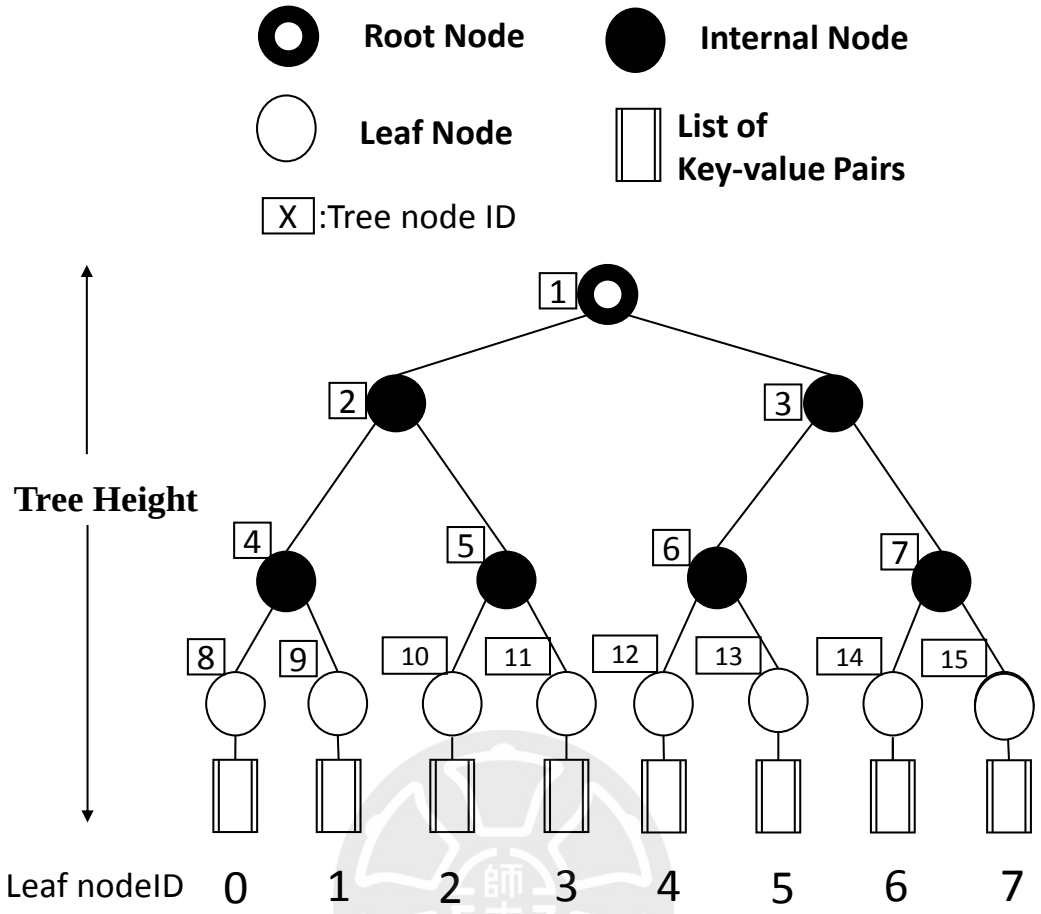


圖 8：樹高為 4 的定位摩克樹。

如果樹高為 N，則總共會有 2^N-1 個節點、2^{N-1} 個葉節點，每一個節點存放 32 bytes 的雜湊值。圖 8 是一個高度為 4 的定位摩克樹，共有 15 個節點與 8 個葉節點。每個內部節點（internal node）所存放的 hash 值，為其左子節點串接（concatenate）右子節點的 hash 值再取一次雜湊的值，由下而上計算到根節點（root node），即與摩克樹的計算方式相同。根節點所存放的 hash 值稱為 top hash、root hash 或稱為 master hash。root hash 便代表所有交易的指紋。

Ψ	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}
Array subscript	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

圖 9：儲存定位摩克樹的一維陣列Ψ。

如果我們將定位摩克樹以一維陣列儲存，令之為Ψ，如圖 9 所示：

若樹高為 N，則Ψ的大小為 2^N-1，令 leaf node ID 為 I，則 tree node ID：$X=I+2^{N-1}$，tree node ID：X 即是一為陣列Ψ的位置（array subscript）。

3.3.1 密碼學證據：Merkle proof

利用 tp-Merkle tree 驗證一筆交易，所需要的密碼學證據（cryptographic evidence），只需要取出該交易所在位置的定位摩克樹片段（slice）進行驗證即可，該「slice」我們稱之為「Merkle proof」，如圖 10。

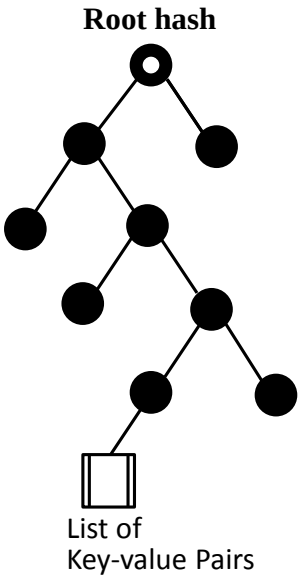


圖 10：Merkle proof 的一個例子

如果是採用傳統的摩克樹存放交易記錄資料，要驗證一筆記錄是不存在的，則必須驗證整顆摩克樹所有的節點；但是在「定位摩克樹」中，我們只要給定該交易的 ID，取出該 ID 的 Merkle proof 依序驗證，就可以確認該交易紀錄是否存在並為正確的。因此，定位摩克樹足以驗證任何交易資料的存在（proof of existence）和正確與否，以及是否根本不存在（proof of inexistence）、純屬虛構的。

若要取出某一交易的「Merkle proof」，其演算法如下：

Algorithm I: Extract a Merkle proof from an tp-Merkle tree which is stored in a one-dimensional array Ψ. Assume that the height of the tp-Merkle tree is *N*.

Input: *I*: a leaf node ID
Ψ: a one-dimensional array which stores a tp-Merkle tree
Output: m.proof(*I*)
M.proof(*I*) = ∅[1]
$X = I + 2^{N-1}$ // Translating leaf node ID to tree node ID
WHILE ($X \neq 1$) DO // Bottom up to the root node
IF X is an even number THEN
M.proof(*I*) = M.proof(*I*) ∪ Ψ[*X*] ∪ Ψ[*X*+1]
ELSE
M.proof(*I*) = M.proof(*I*) ∪ Ψ[*X*-1] ∪ Ψ[*X*]
END IF
$X = \lfloor X \div 2 \rfloor$[2] //Unconditional rounding
END WHILE
M.proof(*I*) = M.proof(*I*) ∪ Ψ[1]

[1] ∅ is an empty set.
[2] ⌊r⌋ means the floor of r, i.e. the largest integer less than or equal to r.

如果要導出某一交易的「Merkle proof」的 root hash，其演算法如下：

Algorithm II: Derive the root hash from a Merkle proof. Assume that the height of the tp-Merkle tree is *N*.

Input: *S*: A m.proof

I : A leaf node ID

Output: γ: Root hash of S

IF (Number of elements in S) ≠ (2**N-1*) THEN

RETURN ERROR

END IF

$X = I + 2^{N-1}$ // Translating leaf node ID to tree node ID

pt = 1

IF (*X* is an even number) THEN

γ = S[*pt*].

ELSE

γ = S[*pt* +1]

END IF

WHILE (⌊ *X* ÷ 2 ⌋ ≠ 1) DO

IF (*X* is an even number) THEN

γ = *hash*(γ | S[*pt* +1])

ELSE

γ = *hash*(S[*pt*] | γ)

END IF

pt = *pt* + 2

X = ⌊ *X* ÷ 2 ⌋

END WHILE

IF (X is an even number) THEN

γ = *hash*(γ | S[*pt* +1])

ELSE

γ = *hash*(S[*pt*] | γ)

END IF

3.3.2 增加或更新一筆資料到定位摩克樹

若要在定位摩克樹存放記錄，或是要變更一筆記錄，其做法的演算法如下：

Algorithm III: Update（or add）a hash value of a data. Assume that the height of the tp-MerkleTree is *N* .

Input: *Indexvalue*: The index value of the data.

value: The hash value of the data.

Ψ: A one-dimensional array that stores a tp-MerkleTree; PB（*i*）denotes the key-value pairs stored in the leaf node with an ID of *i*.

R: The new root hash.

I = Γ（ *Indexvalue*） // Obtain the leaf node ID

$X = I + 2^{N-1}$ // Translate the leaf node ID, *I*, into a tree node ID, *X*

WHILE （$X \neq 1$） DO // From the bottom of the tree up to the root node

IF *X* is an even number THEN

Ψ[X/2] = hash（Ψ[X] || Ψ[*X*+1]）

ELSE

Ψ[（X-1）/2] = hash（Ψ[*X*–1]）|| Ψ[X]）

END IF

X = ⌊ X / 2 ⌋ // Unconditional rounding

END WHILE

RETURN *R*

3.3.3 在定位摩克樹中驗證一筆資料

若要在定位摩克樹中，驗證一筆記錄，其做法的演算法如下：

Algorithm IV: Verify if a hash value of a data is valid. Assume that the height of the tp-MerkleTree is *N* nodes.

Input: *Indexvalue*: The index value of the data to be verified.

Ψ: A one-dimensional array that stores a tp-MerkleTree; KV（*i*）denotes the key-value pairs stored in the leaf node with an ID of *i*.

S: The latest downloaded root hash.

I = Γ（ *Indexvalue*） // Obtain the leaf node ID

$X = I + 2^{N-1}$ // Translate the leaf node ID, *I*, into a tree node ID, *X*

Y = Ψ[*X*]

WHILE （$X \neq 1$） DO // From the bottom of the tree up to the root node

IF *X* is an even number THEN

Y = hash（*Y* || Ψ[*X*+1]）

ELSE

Y = hash（Ψ[*X*–1]）|| *Y*）

END IF

X = ⌊ X / 2 ⌋ // Unconditional rounding

END WHILE

IF （*Y*≠*S*） THEN Report “Some hash values are not valid.”

ELSE

Compute the hash value of KV（*I*）, α.

IF （α≠Ψ[*X*]） THEN

Report “Some hash values are not valid.”

ELSE

Obtain the key-value pair of *Indexvalue* in KV（*I*）. Assume the hash value of *Indexvalue* is *h*.

Report that *h* is valid.

END IF

END IF

3.3.4 定位摩克樹的效能

為了確認「定位摩克樹」在不同資料量與不同樹高下所展現的效能與其關係，筆者首先先測試定位函數 Γ 的效能，是不是足以均勻的分配存放於葉節點的 key-value pairs；此外，筆者使用了不同的樹高與資料量對定位摩克樹的壓力測試，檢驗定位摩克樹於「建立初始樹」所需的時間與空間，和「更新或新增資料」與「稽核驗證資料」所需要的時間等的基本效能。

測試用的電腦規格：

Macbook Air 2014，中央處理器是為 Intel Core i5 1.4GHz 安裝 4 GB 1600 MHz DDR3 記憶體，作業系統是 OS X El Capitan 10.11.5。

測試用的資料規格：

每一筆 key-value pair 64 bytes（即索引雜湊值：32bytes，加上資料雜湊值：32 bytes，採用 SHA256 演算法），總共分為 5 萬筆、25 萬筆、50 萬筆，與 100 萬筆等資料，分別於樹高 4、6、8、10、12、14、16、18、20 等的定位摩克樹進行測試。

首先，筆者先在不同的樹高下，以不超過該樹高的葉節點數量放入 key-value pairs 做定位函數 Γ 的碰撞測試，測試結果如表 2。

表 2：不同數量的 key-value pairs 在不同樹高之定位摩克樹的碰撞情形

Height	# of Leafs	# of key-value pairs	Stock ratio	α	β
9	256	85	1/3	1.19	2
		170	2/3	1.42	3
		256	full	1.64	4
11	1024	341	1/3	1.17	4
		682	2/3	1.35	4
		1024	full	1.61	5
13	4096	1365	1/3	1.18	4
		2730	2/3	1.37	5
		4096	full	1.58	6
15	16384	5461	1/3	1.17	5
		10922	2/3	1.37	6
		16384	full	1.59	7
17	65536	21845	1/3	1.17	5
		43690	2/3	1.37	6
		65536	full	1.58	**8**

α: Average number of stored key-value pairs in a non-empty leaf node
β: Maximum number of stored key-value pairs in a non-empty leaf node

結果發現：當存證資料的數量未超過或等於葉節點的數量時，其最大碰撞率為「8」。

接下來進行建立初始化的定位摩克樹，考慮現實狀況的資料量，模擬建立不同樹高的定位摩克樹，若樹高為 N，則會有 2^{N-1} 個葉節點儲存交易資訊，若樹高為 20，則 $2^{20-1}=2^{19}=524288$，若未發生碰撞，至少可以存放 50 萬筆交易資訊。

實驗結果發現：以相同資料筆數，也就是固定的 key-value pais 數量，去建立不同樹高之定位摩克樹，所需的「時間」與「空間」並無太大差異，實驗結果請參照表 3。

表 3：建立不同樹高之定位摩克樹所需的時間與空間
(70 萬筆 key-value pairs)

Tree Height	Build up Time（s）	tp-Merkle tree（MB）
4	517.993	147.6
6	518.315	147.6
8	518.776	147.7
10	518.562	147.8
12	518.298	148.2
14	518.757	150.0
16	518.914	157.2
18	520.682	185.9
20	520.037	296.4

於是接下來筆者在不同的樹高下，用各種不同資料量做更新與稽核的效能測試，如下表 4 與圖 11 所示：

表 4：不同資料量在各個樹高下的更新資料所需時間(單位：ms)

Tree Height	Number of key-value pairs	Update time
4	50000	3.89
	250000	12.87
	500000	20.06
	1000000	34.12
6	50000	2.12
	250000	7.05
	500000	7.21
	1000000	11.58
8	50000	1.24
	250000	3.35
	500000	3.54
	1000000	7.29
10	50000	0.97
	250000	1.88
	500000	2.18
	1000000	2.23
12	50000	0.76
	250000	1.57
	500000	2.3
	1000000	2.64
14	50000	0.83
	250000	1.77
	500000	2.09
	1000000	1.89
16	50000	0.94
	250000	1.64
	500000	1.94
	1000000	2.24
18	50000	1.1
	250000	1.86
	500000	1.47
	1000000	3.27
20	50000	1.53
	250000	1.99
	500000	1.57
	1000000	3.26

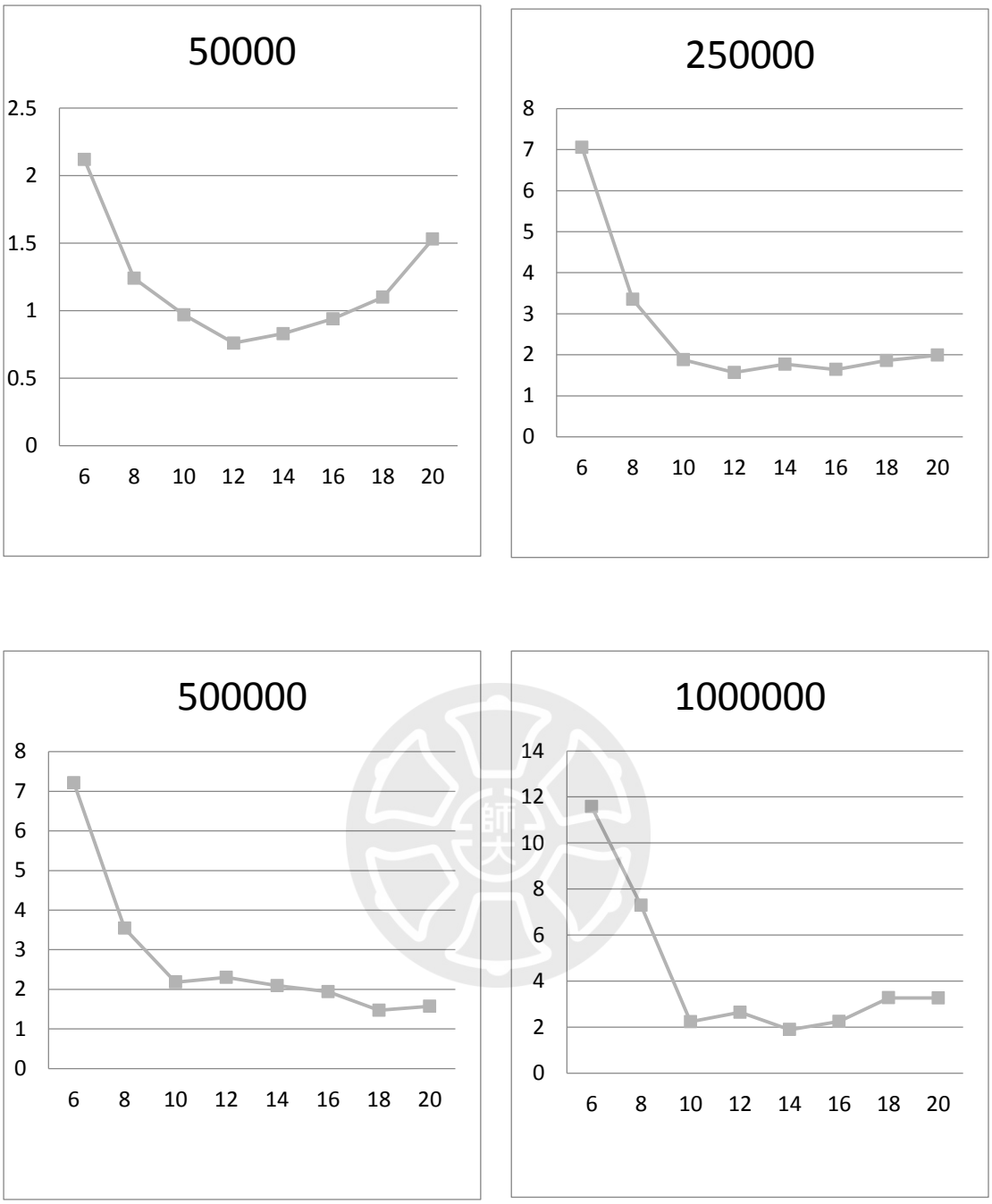


圖 11：不同資料量在各個樹高下的更新資料所需時間曲線圖
(單位：ms；縱坐標：所需時間，橫坐標：樹高。)

更新的效能測試結果，發現了一個現象：

當定位摩克樹，樹高超過 10（含）以上，無論資料筆數多寡，其更新資料所需的時間，幾乎呈現水平狀態。於是接下來筆者再用相同規格的各種不同資料量，做驗證的效能測試，如下表 5 與圖 12 所示：

表 5：不同資料量在各個樹高下驗證稽核所需時間(單位：ms)

Tree Height	Number of key-value pairs	Audit time
4	50000	2.77
	250000	10.90
	500000	17.11
	1000000	31.90
6	50000	0.83
	250000	3.19
	500000	5.10
	1000000	9.30
8	50000	0.28
	250000	0.92
	500000	1.65
	1000000	3.09
10	50000	0.11
	250000	0.26
	500000	0.47
	1000000	0.83
12	50000	0.08
	250000	0.06
	500000	0.23
	1000000	0.35
14	50000	0.07
	250000	0.08
	500000	0.12
	1000000	0.08
16	50000	0.03
	250000	0.09
	500000	0.07
	1000000	0.04
18	50000	0.09
	250000	0.09
	500000	0.08
	1000000	0.06
20	50000	0.09
	250000	0.1
	500000	0.07
	1000000	0.14

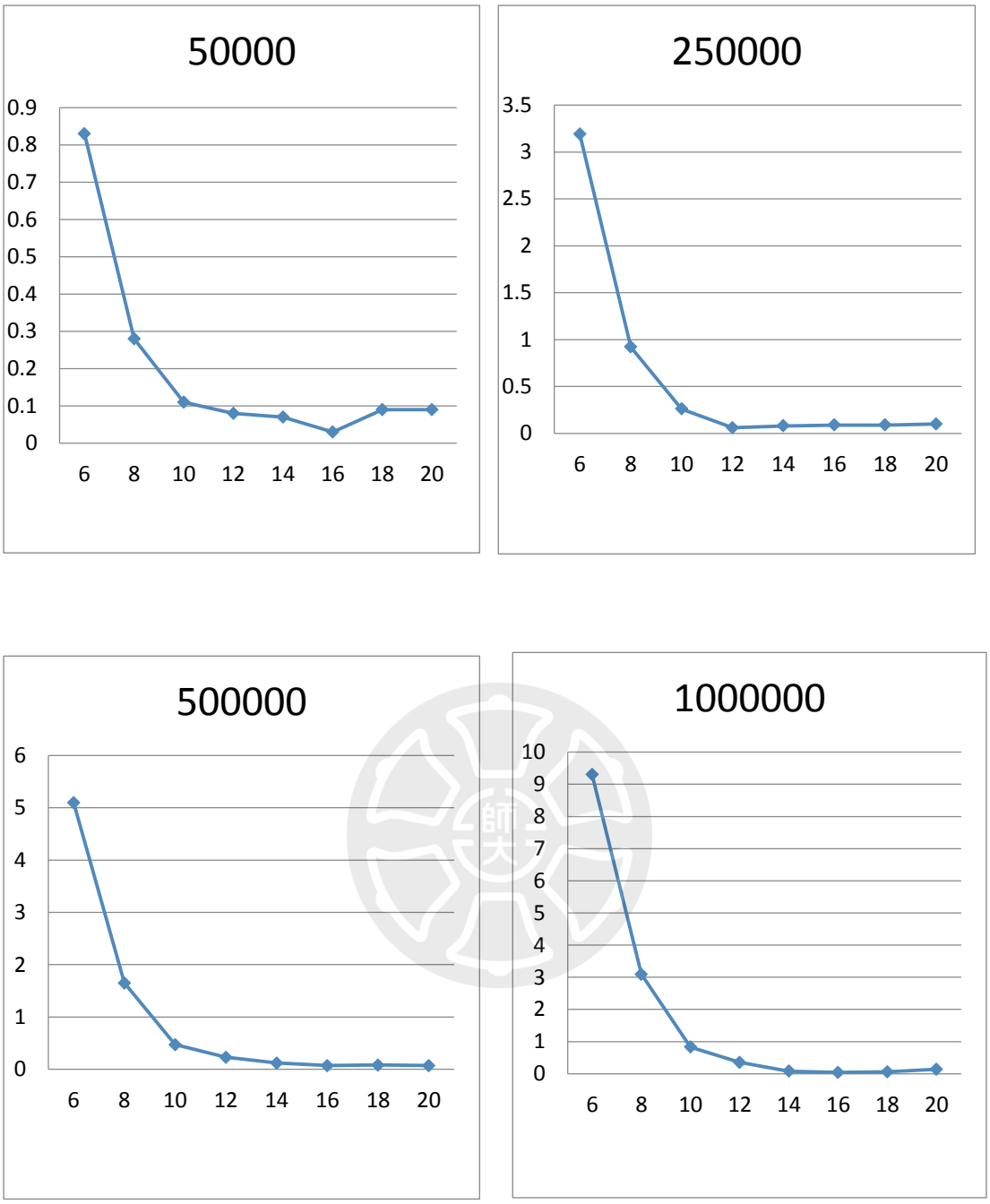


圖 12：不同資料量在各個樹高下驗證稽核所需時間曲線圖
(單位：ms；縱坐標：所需時間，橫坐標：樹高。)

效能測試結果，發現了一些現象：

一、當 leaf node 的數量足夠存放 key-value pairs 時，定位函數 Γ 足以均勻分配 key-value pairs，max collision 最高為 8，也就是會在 10（含）以下（請參考 4.1.2 節與表 5）。

二、當 list of key-value pairs 超過 10 以上或更高，碰撞與串接 hash 值所造成的時間損失顯著。

三、當定位摩克樹，樹高超過 10（含）以上，無論資料筆數多寡，其檢驗資料所需的時間，幾乎呈現水平狀態。

由以上的測試，得到以下結論：

一、建立定位摩克樹時，使用的樹高所產生的 leaf node 數量，避免小於存證數量太多。

二、定位摩克樹在樹高為 10（含）以上時，無論資料筆數多寡，其更新或儲存資料、與驗證資料，兩者所需時間，皆呈穩定能狀態。

第四節 定位摩克樹的相關研究

以下各節，是 Hwang et al.使用定位摩克樹，在選定的應用情境的實作研究，發表的成果，以下就選定的情境與解決的問題，分節簡述之。

3.4.1 雲端儲存空間檔案的存證與稽核

Hwang et al.發表於 2016 年在第 9 屆 IEEE Cloud Computing 學術會議的研究，名為：「Efficient Real-time Auditing and Proof of Violation for Cloud Storage Systems」[97]。

本研究主要目的，在於租用雲端儲存空間時，當存放於雲端的資料發生異常時，如何有足夠的證據歸責，以做為後續的申訴或求償的依據。系統架構如圖 13。

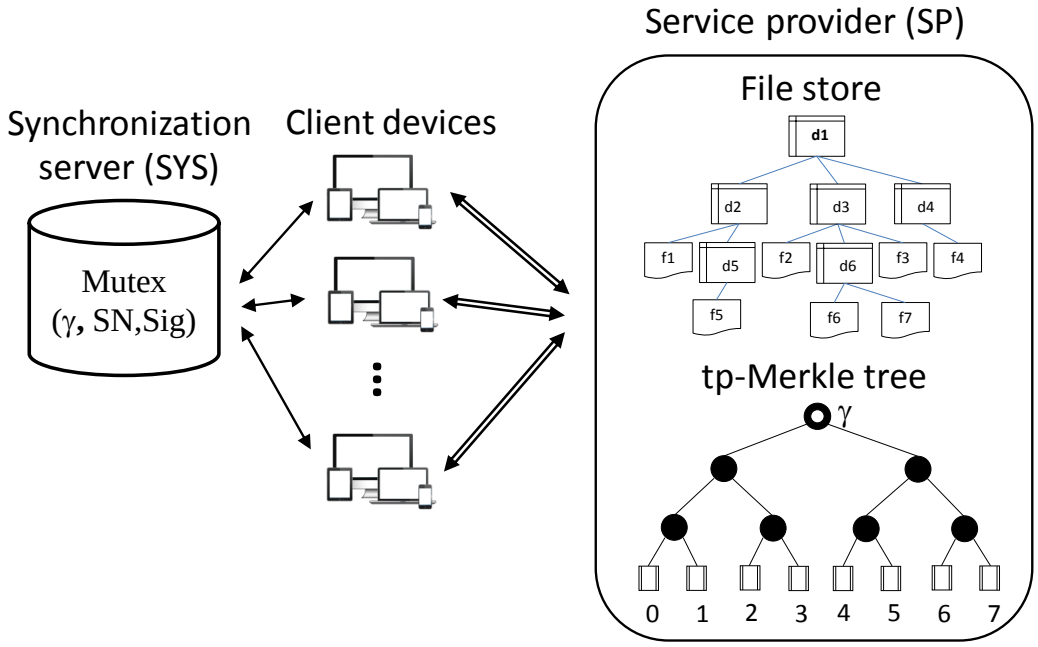


圖 **13：Efficient real-time PoV 系統架構圖。**（取自[97]）

該研究運用定位摩克樹，做為所有檔案資料的指紋存證，並由服務伺服器端負責保管並更新；而用戶端只需要存放三種證據：(1)該定位摩克樹最終版本的 root hash：γ，(2)PoV 協議最後一個作業訊息的序號：SN，(3)服務端對 root hash 的簽章：Sig。該三種證據形成一組 three-tuple 的證據：(γ，SN，Sig)由服務端產生，由用戶端設備存證，three-tuple 存證所需空間不到 1KB。

該研究的系統執行步驟如下：步驟（1）當用戶端的作業開始，用戶端先向服務端要求現行作業所需的 Merkle proof 進行立即稽核，以確認執行前資料是否無異常；步驟（2）稽核通過後則作業開始；步驟（3）當作業完成，則確認最終證據，通過稽核後，更新服務端的定位摩克樹至最新版，同時產生最新版的(γ'，SN'，Sig')更新用戶端，上傳更新資料；步驟（4）若稽核不通過，則(γ'，SN'，Sig')可以做為責任歸屬的證據(也就是 fraud proof)。

此架構同樣使用同步伺服器(SYS，請參照第三章第二節說明)，使得多個用戶端設備可以同時存取同一個雲端帳號，以共用雲端硬碟儲存檔案。

3.4.2 區塊鏈的擴容與分散式稽核：InfiniteChain

Hwang et al.發表於 2018 年在 International Conference on Blockchain(ICBC)國際學術會議的研究，名為：「InfiniteChain：A Multi-chain Architecture with Distributed Auditing of Sidechains for Public Blockchains」[98]。

由於公有區塊鏈（public blockchains）的去中心化的分散式架構，造就了 Web3.0 的核心(請參照第一章第一節的說明)，擁有資訊公開透明且難以竄改的特性，但是卻存在著擴容性、隱私性與應用環境的三大限制與區塊鏈三難（blockchains trilemma）的問題：(請參照第一章第三節 1.3.3 的說明)

一、有限的交易頻寬（bandwidth)：每秒至多能處理的交易量在 15 TPS 以下。

二、有限的交易負載（payload)：過多的交易會造成區塊鏈膨脹。

三、資訊缺乏隱私的保護（privacy protection)。

四、有限的應用環境(applications)：以加密貨幣為主，限制了 Dapp 的應用。

為了解決以上的問題，該研究提出了 Multi-chain 架構，如圖 14。

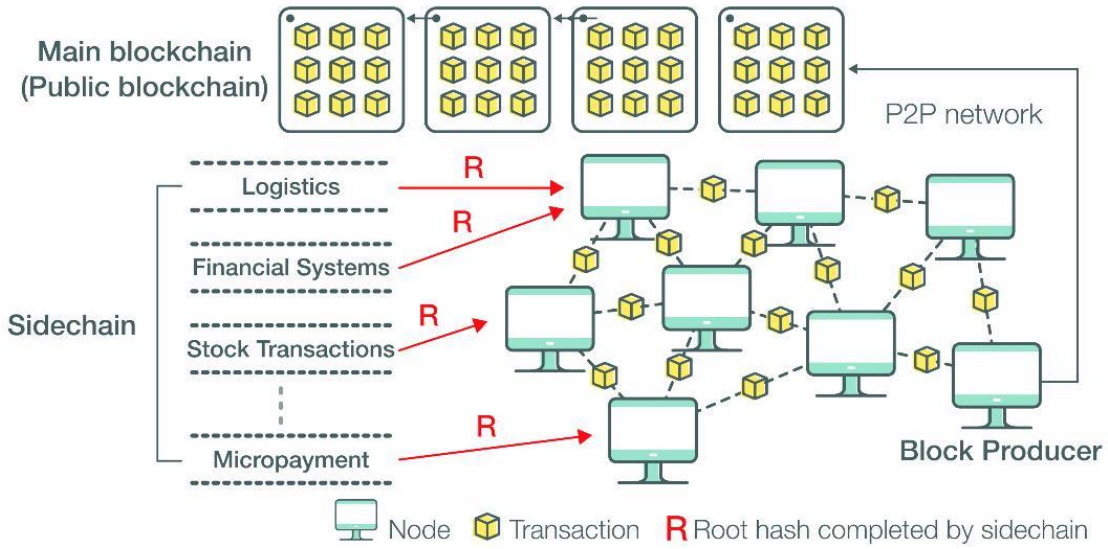


圖 14：Multi-chain 系統架構圖（取自[98]）

所有的交易活動皆在區塊鏈主鏈外處理，將每一段時間的每一串鏈下交易活動的交易指紋及其版本，記錄到主鏈的智能合約，因此每一串交易活動皆可分別稽核是否正確，該研究稱之為「分散式稽核」（distributed auditing），任何參與者皆可利用產生的很小的密碼學證據（cryptographic evidence）在此稱為「Merkle proof」驗證，若有錯誤，此「Merkle proof」可當「詐欺證明」（fraud proof），啟動預先定義在主鏈上智能合約的申訴機制。

每一個鏈下交易會有一個服務提供者，該服務者負責扮演代理人（agent）的角色，若是側鏈，則會有側鏈代理人（agent），與所有交易參與者以 PoV 機制溝通，並負責整合傳遞該鏈下交易的所有交易指紋給在主鏈上的稽核作業員（auditing node）。

由主鏈上的礦工，個別負責不同類型的鏈下交易（或側鏈，sidechain），該礦工也是負責該鏈下交易的稽核作業員（auditing node），必須負責產生該鏈下交

易的「定位摩克樹」，並將每次更新的 root hash 與版本碼（identification code）打上主鏈上的智能合約，這可以說是更有效率的進階版的錨定機制（anchoring，可參照第一章第三節 1.3.4 的說明）。因此每個鏈下交易或側鏈的「定位摩克樹」指紋，會在主鏈上同步，這麼做法，除了解決了主鏈擴容性、隱私性與應用上的問題外，並可以利用公有區塊鏈的可信任的平台，達成分散式稽核的目的，阻卻所有交易資料遭到外在的破壞、植入或竄改。

3.4.3 雲端儲存空間使用的自動賠償機制

Hwang et al.發表於 2020 年在第二屆 International Conference on Blockchain Technology（ICBCT 2020）國際學術會議的研究，名為：「Blockchain-based Automatic Indemnification Mechanism based on Proof of Violation for Cloud Storage Services」[83]。

由於原有的 PoV 機制（可參照第三章第二節的說明）是無法利用公有區塊鏈做到違約賠償的機制，因此必需修改用戶端與服務端交換訊息的格式如下：

$M^{i}_{request}$ = (CMD, SN^{i}, preRH, hash(M^{i-1}_{reply}) , SIG_{client})

M^{i}_{reply} = (hash($M^{i}_{request}$), Accept, Result, (Slices, Sets of PB-pairs), SN^{i}, RH, SIG_{SP})

其中：**CMD** 是檔案作業指令，如讀取、寫入、搬移、刪除等，並包含指令的引數（parameters）。

SN^{i} 是檔案作業的序號，由 1 開始。

preRH 是此次檔案作業開始前的 root hash。

SIG_{client} 是用戶端的數位簽章。

如果用戶端的 SN、preRH 和 SIG_{client} 都正確，服務端的回覆訊息的「**Accept**」值會設為「YES」，否則設為「NO」。

hash($M^i_{request}$)是用戶端作業要求訊息的雜湊值。

Result 是服務端就用戶作業要求訊息的執行結果，包含有：所存取檔案的雜湊值與執行成功或錯誤的訊息。

(Slices, Sets of PB-pairs)是此次作業所存取檔案的 PB-pairs 與 Merkle proof。

RH 是此次作業結束後所得到的更新 root hash。

SIG_{SP} 是服務端的數位簽章。

$M^i_{request}$ 與 $M^i_{request}$ 的 **SN^i** 是相同的。

註：PB pair 是指路徑值（pathname）與檔案二元碼值（binary code of file）分別的雜湊值，所組成的「key-value pair」。

以上除了有 chain-hashing 的機制 [96]，並增加了 Merkle proof 在訊息中。並由服務端，依照交換訊息的結果，將存放在用戶所租用的雲端空間的所有檔案目錄的最新版本的「定位摩克樹」，建立並存放存證，讓用戶端依交換訊息的結果，也就是服務端的最後一個 reply 訊息，做為即時稽核與申訴之用。系統架構圖請參照圖 15。

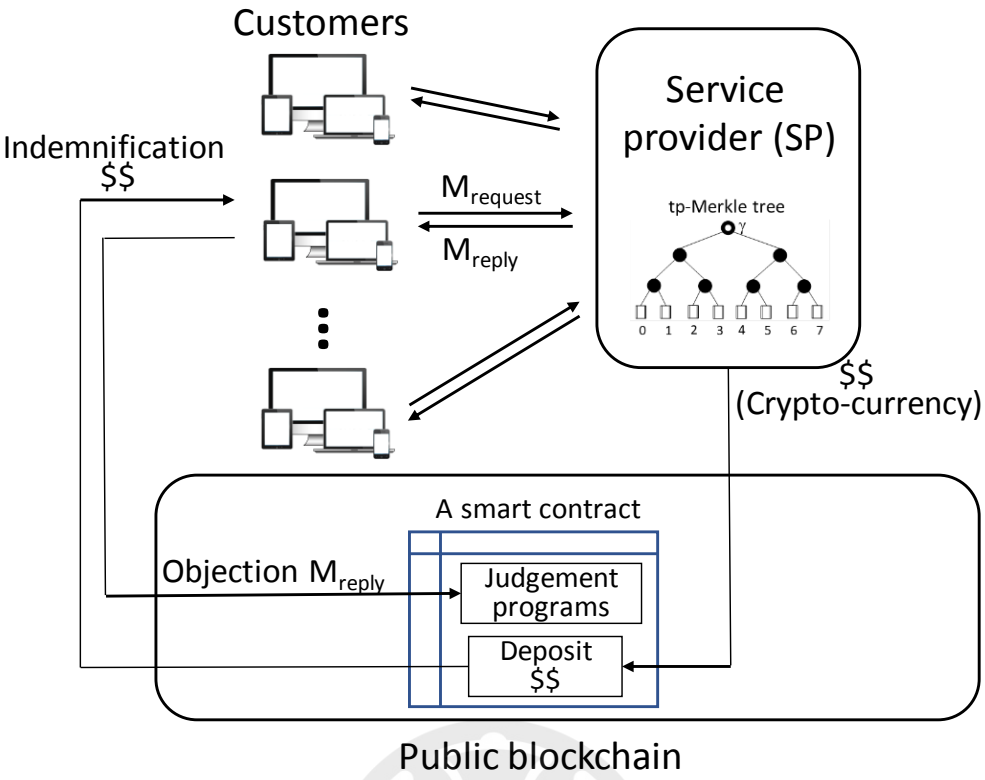


圖 15：雲端儲存空間的 blockchain-based AIM 系統架構圖（取自[83] ）

3.4.4 公共金鑰基礎建設的應用

Hwang et al.發表於 2021 年發表於 Journal of Security and Communication Networks 期刊的研究，名為：「A Semidecentralized PKI System Based on Public Blockchains with Automatic Indemnification Mechanism」[100]。

公共金鑰基礎建設（public key infrastructure，簡稱 PKI）簡稱「公鑰系統」，是目前廣泛被使用在網際網路上，用以辨認與確認真實身分的憑證系統；例如我國的自然人憑證 [37] 即為一例。任何人可以將個人身分資訊向 CA（certificate authority）伺服器註冊，取得 CA 伺服器所發放的個人「數位憑證」（digital certificate），往後在網路上的作業，就可以以此「數位憑證」當作證明個人身分

的憑據。

由於目前的公鑰系統,仍然會有 CA 伺服器或 CRL(certificate revocation list) 遭受攻擊與單點失效危險 [38]，過去也曾經有一些研究，利用公有區塊鏈存放數位憑證與 CRL [70, 101],用戶端只需要上區塊鏈確認數位憑證的資訊，無須與 CA 伺服器聯繫，但是這樣的速度慢，並會產生過多的礦工費，若用戶眾多，同時在區塊鏈上查詢資料，有可能讓區塊鏈礦工以為遭到「分散式服務阻斷攻擊」而遭終止查詢；可參照第一章第四節 1.4.1 的說明。

因此如何防止單點失效、阻卻攻擊，並可以有效存證以防憑證資訊損壞，而形成一個可以有效快速稽核與責任歸屬的機制，便是該研究的目標。

首先先將數位憑證的狀態，由三種：Good、Revoked、Unknown，擴增為四種：Add、Renew、Pause、Revoked。定義「定位摩克樹」key-value pair 的資料結構為：< hash（IndexValue），hash（certificate + status 的二元碼）>。

接下來定義網站端與 CA 服務端交換訊息的格式如下：

$M_{request}$ = [Operation, Pub(owner), IndexValue, CO , $SIG_{Pri(owner)}$]

M_{reply} = [$M_{request}$, Result, h(Certificate), Status, $SIG_{Pri(CA)}$]

其中：**Operation** 有：apply certificate、change status、replace key 等。

Pub(owner)是網站端的公鑰。**Pri(owner)**是網站端的私鑰。

IndexValue 是該憑證的 key-value pair 存放在定位摩克樹的索引值。

CO 是 **C**learance **O**der。

SIG 是該訊息的數位簽章。

h(Certificate)是該數位憑證的雜湊值。

Status 是該數位憑證的狀態。

CA 伺服器在經過一段時間的累積依回覆訊息的結果，建立最新版本的「定位摩克樹」存放並存證，公告在「星際檔案系統」（Inter-planetary file system，簡稱 IPFS）[102, 103] 上以供任何人稽核之用，並將 root hash 與 CO 打上公有區塊鏈的智能合約。owner 端只需存放最後一個 CA 回覆的訊息為證據，作為往後啟動稽核機制與申訴賠償之用。參照系統架構如圖 16。

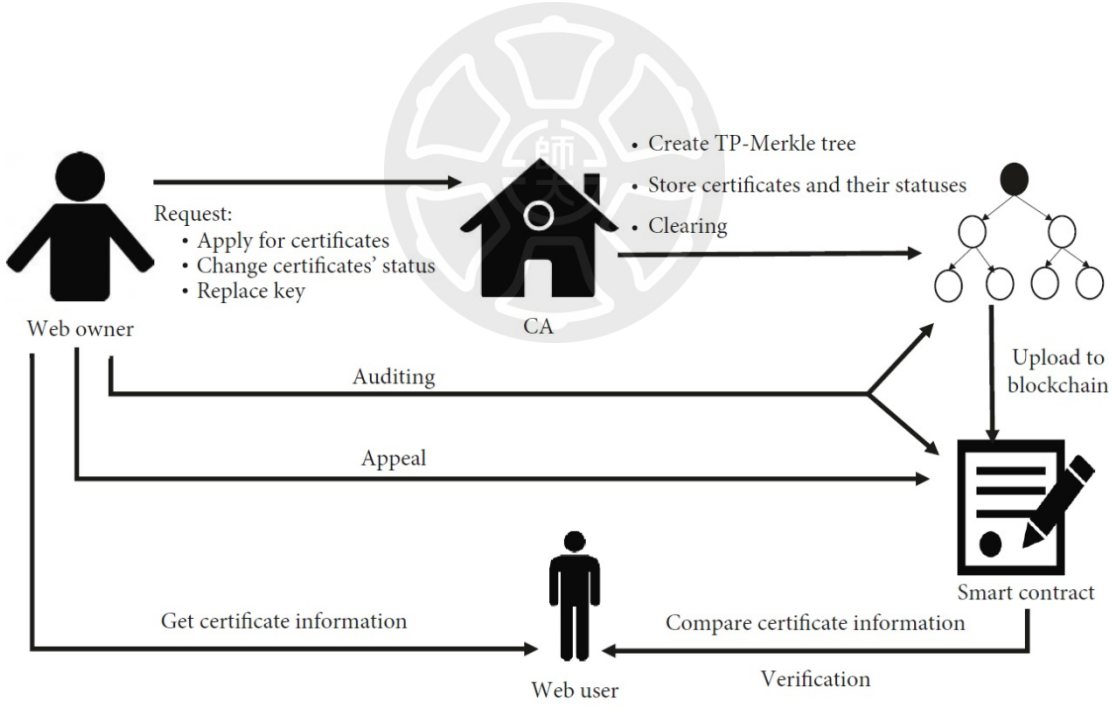


圖 16：Semi-decentralized PKI 系統架構圖（取自[100] ）

第四章、研究成果與實驗結果

本論文以「定位摩克樹」做為資料存證的機構，並研究了在兩種實際情境下的存證應用：

1、雲端系統執行環境的即時稽核(Real-time Auditing of the Runtime Environment for Cloud Computing Platforms)。

本項研究發表於 Journal of Information Science and Engineering 期刊 [84]。

2、利用公有區塊鏈提供自動給付與賠償機制(Automatic Reward System Based on Public Blockchains)。本項研究發表於第四屆 IEEE Eurasia Conference on IoT, Communication and Engineering 2022 國際學術會議，並獲得該會議的最佳論文獎 [85]。

以上的兩項研究，證明了分別在該情境下，能夠有效而快速的確認作業過程的不可否認性與可歸責性，阻卻外來攻擊。除了可以即時而快速的驗證作業內容外，防止因遭任何一種外力的資訊破壞，並可即時察覺，也可以提供有效的「詐欺證明」(fraud proof)，做為究責之用，並且可解決公有區塊鏈的擴容與隱私保護問題，擴大了 Dapp 的應用範圍。系統架構與實驗結果，詳見以下兩節。

第一節 雲端執行環境的即時稽核

本研究的目的，是為了解決雲端服務供應商所提供的雲端虛擬機器，稽核其執行環境的安全，以確保用戶在使用時期，執行環境避免因遭到惡意攻擊或損壞，

使軟體執行時有意外情形的發生，並做適當的存證，如果可以做執行環境的即時即時稽核，則可以確保用戶的資料不被損壞，所存放的證據也可作歸責之用。

一個軟體執行環境的運算平台，包含有某些硬體架構、作業系統，以及執行時所需要的函式庫。軟體作業執行環境的稽核，即是包含運算平台所有的軟體元件的完整性檢驗（integrity checking）。

雲端服務，提供租用者所需的資訊系統與其執行環境。雲端服務的租用者，會在雲端服務所提供的所謂「虛擬機器」（VM）上安裝作業系統（operating-system images）與應用軟體。例如 PaaS 服務，讓租用者可以輕易的開發、執行與管理程式，不需要去處理複雜的相關環境與載入軟體的基礎架構。IaaS 與 PaaS 服務，提供租用者某種運算平台的 VM，該 VM 的影像系統檔（images）由服務提供商所控制，租用者無法知道是什麼？這些「images」有可能因為內部的錯誤、惡意的安全性攻擊，導致損毀。惡意軟體或電腦病毒也有可能感染或被安裝在 VM 的 images 中；這些都是租用者無法控制或者能事先偵測到的。某些服務提供商支援 VM 的即時系統轉移（live migration），租用者是無法支配、或者被通知 VM 是否已經轉移到另一個硬體平台環境。而另一種可能的情境是，當 VM 的 images 遭到損毀，服務提供商會主動用前一版本的 images 備份來還原 VM 系統，而租用者無從得知其最新版的 images 已經遺失了！這就是筆者在第一章第五節的 1.5.1 所說的「roll-back attack」與「replay attack」，如果租用者無法立即發現這種「roll-back」攻擊，這種過期的 VM 執行環境現況，會導致很多

的問題。病毒偵測軟體無法察覺這樣的問題，而且沒有任何一種電腦防毒軟體可以解掉所有的電腦病毒，特別是新的電腦病毒或者是未知病毒。

本研究展示如何在雲端執行環境的 VM 與單機執行環境做有效率的稽核；簡單的說，要解決這些問題，要達到雲端執行環境的即時稽核的目標，就使要有一個有效率的存證與稽核機制，能夠在載入軟體前，做雲端虛擬機器或單機的執行環境的即時完整性檢查（real-time integrity checking for run-time environments）。

4.1.1 系統架構

筆者運用「定位摩克樹」提出一個創新的可即時且快速稽核的架構。首先會先安裝一個「雜湊值儲存區」，稱為「*hash value archive*，簡稱 HVA」，主要目的是用來儲存執行環境所有相關軟體元件的檔案的雜湊值。筆者所提出的系統架構是以定位摩克樹建構 VM 在運行時所有相關檔案與動態連結函式庫檔案等元件的雜湊值，每一個檔案的雜湊值以「key-value pair」記錄，「key-value pair」的資料結構如下：

<hash（pathname）, hash（file）>

「key-value pair」的「key」是用來定位的「索引值」（index value），即檔案所存放的「路徑位置名稱」（pathname），「value」是「檔案」的雜湊值，皆採用 SHA256 演算法取雜湊值，得出之雜湊值大小為 256 bits 即 32 bytes。每一個 key-value pair 的大小則會是 64 bytes。每一筆交易定位之後，取其 key-value pair

的 SHA256 雜湊值，存放在定位的葉節點，因此葉節點也會是 32 bytes 大小。

儲存於 HVA 的雜湊值與其所建立的定位摩克樹，可參照圖 9 與圖 10，定位摩克樹所使用來索引定位的**Γ 函數**表示式如下：

$$\Gamma(pathname) = SHA256(pathname)\ mod\ 2^{N-1} \quad \text{公式（2）}$$

N：表示定位摩克樹的樹高。

而所有的雜湊值以「定位摩克樹」建置。代理程式（agent）負責當軟體執行時，即時檢查 VM 環境的系統完整性，例如是否有遭破壞或遺失檔案。這個代理程式同時負責控制軟體的載入、執行、連結相關動態函式庫檔案的工作，因此我們假定這個代理程式式不會遭駭或破壞的 [104]。例如我們可以將「代理程式」存放在另一部安全的電腦中，需要使用時再複製過來，而且啟動「代理程式」時需要輸入密碼，因此駭客或惡意程式就無法啟動。其他可供保護「代理程式」以防遭駭的方法，可參照 [104] 該篇論文。「狀態碼」（*status code*）代表最新執行環境的正確狀態，即完整性（integrity）。本研究所用以稽核的「狀態碼」，就是定位摩克樹的「根雜湊值」，其所占空間很小，只有 32-byte，所以足以進行快速驗證。如圖 17。

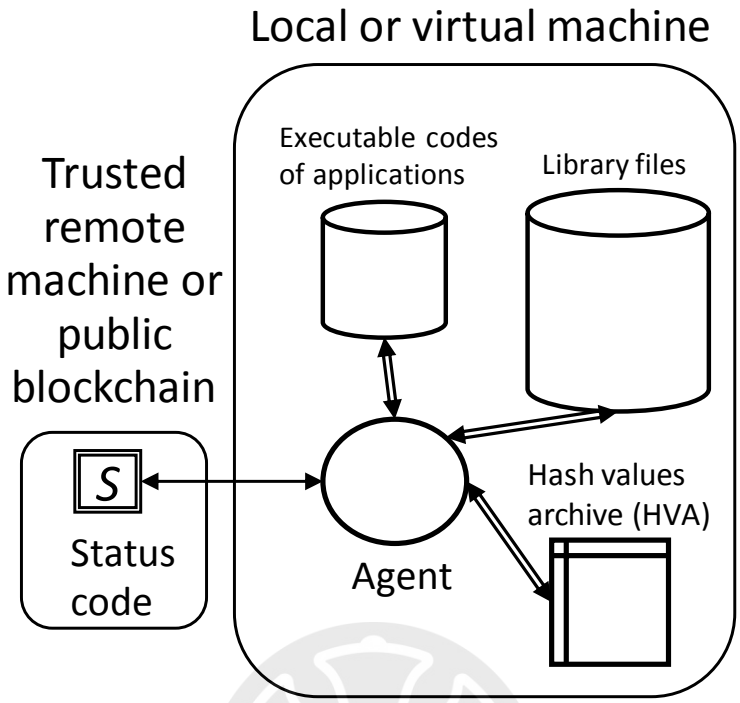


圖 17：雲端服務稽核系統架構圖。

本架構的執行步驟如下：

步驟（1）：代理程式從遠端信任的電腦或公有區塊鏈下載狀態碼 S；

步驟（2）：代理程式從 HVA 計算狀態碼 S'；

步驟（3）：代理程式比較 S 和 S'，以驗證 HVA 中的雜湊值是否正確。

步驟（4）：根據 HVA 中的雜湊值，檢查需要動態連結的應用軟體和函式庫檔案對應的可執行程式碼。

步驟（5）：如果驗證通過，則代理程式會載入、連結、並執行應用軟體。

「代理程式」驗證執行環境的軟體元件檔案的演算法如下：

Algorithm : Verify if a hash value of a file is valid. Assume that the height of the tp-Merkle tree is *N* nodes.

Input: *FPname*: The pathname of the file to be verified.

Ψ: A one-dimensional array that stores an tp-Merkle tree; PB(*i*) denotes the key-value pairs stored in the leaf node with an ID of *i*.

S: The latest downloaded status code.

(1) $I = \Gamma(FPname)$ // Obtain the leaf node ID

(2) $X = I + 2^{N-1}$ // Translate the leaf node ID, *I*, into a tree node ID, *X*

(3) $Y = \Psi[X]$

(4) WHILE ($X \neq 1$) DO // From the bottom of the tree up to the root node

IF *X* is an even number THEN

$Y = \text{hash}(Y \,||\, \Psi[X+1])$

ELSE

$Y = \text{hash}(\Psi[X-1] \,||\, Y)$

END IF

$X = \lfloor X / 2 \rfloor$ // Unconditional rounding

END WHILE

(5) IF ($Y \neq S$) THEN Report "Some hash values are not valid."

ELSE

Compute the hash value of PB(*I*), α.

IF ($\alpha \neq \Psi[X]$) THEN Report "Some hash values are not valid."

ELSE

Obtain the PB pair of *FPname* in PB(*I*). Assume the hash value of *FPname* is *h*.

Report that *h* is valid.

END IF

END IF

註：PB(*i*)是指在 tp-Merkle tree 下，第 *i* 個葉節點所存放的 key-value pair，PB pair 是指路徑值（pathname）與檔案二元碼值（binary code of file）分別的雜湊值，所組成的「key-value pair」。

說明：例如當要檢查的檔案，經Γ函數得到 $I = 3$、$X = 11$、$Y = \Psi[11]$，在第一代的 while loop 得到 $Y = \text{hash}(\Psi[10] \| Y)$，第二代的 while loop 得到 $Y = \text{hash}(\Psi[4] \| Y)$ and $Y = \text{hash}(Y \| \Psi[3])$，第三代的 while loop 得到 $Y = \text{hash}(Y \| \Psi[3])$，最後一次的 while loop 得到 $Y = \text{hash}(\text{hash}(\Psi[4] \| \text{hash}(\Psi[10] \| \Psi[11])) \| \Psi[3])$，因此最後也就是第（5）步驟便可以驗證 hash(PB(11)) 和 Ψ[11]是否相等？

這個架構之所以可以在 VM 中，執行載入任何應用程式前，快速地進行相關執行環境與連結檔案的檢查，而不會造成明顯的延遲，關鍵就在於只需下載非常小「status code」進行驗證即可。

這個「status code」可以儲存在遠端可信任的電腦或用戶端的電腦，或是打上公有區塊鏈儲存，以備即時稽核之用。

筆者另外考慮兩種直覺式的執行環境的完整性檢查方案：

（1）將所有的檔案的 PB pair 存放在 HVA 中，並計算 PB pairs 串接後的雜湊值。

（2）將所有的檔案的 PB pair 以 m 元雜湊樹儲存在 HVA 中。

這兩種做法，筆者也會在實驗中實作，與本研究所提出的架構作效能上的比較。

以下兩小節，分別說明這兩種直覺式方案的做法。

4.1.1.1 直覺式方案一：檔案的雜湊值以 PB pair 存放在 HVA

將每一個執行碼與函數檔以前面所說的 key-vale pair 也就是 PB pair 儲存在 HVA 中，其結構式為：(P_i , B_i)，P_i 是第 i 個檔案其路徑的雜湊值，B_i 是第 i 個檔案二元碼的雜湊值。假定 HVA 中有 N 個檔案的 PB pair 雜湊值，所以狀態碼 S = $hash(hash(P_1||H_1) || hash(P_2||H_2) ||\ldots|| hash(P_N||H_N))$，將所有的 PB pairs 雜湊值串接起來取雜湊值，而每次要執行相關檔案的完整性檢查，都要執行一次將所有的 PB pairs 雜湊值串接起來取雜湊值，再與狀態碼比對。

4.1.1.2 直覺式方案二：檔案的雜湊值以 m 元雜湊樹存放在 HVA

依照檔案存放的目錄結構，相對的建立 m 元雜湊樹，以存放每一個檔案的 PB pair 雜湊值，如圖 18、圖 19 所示。

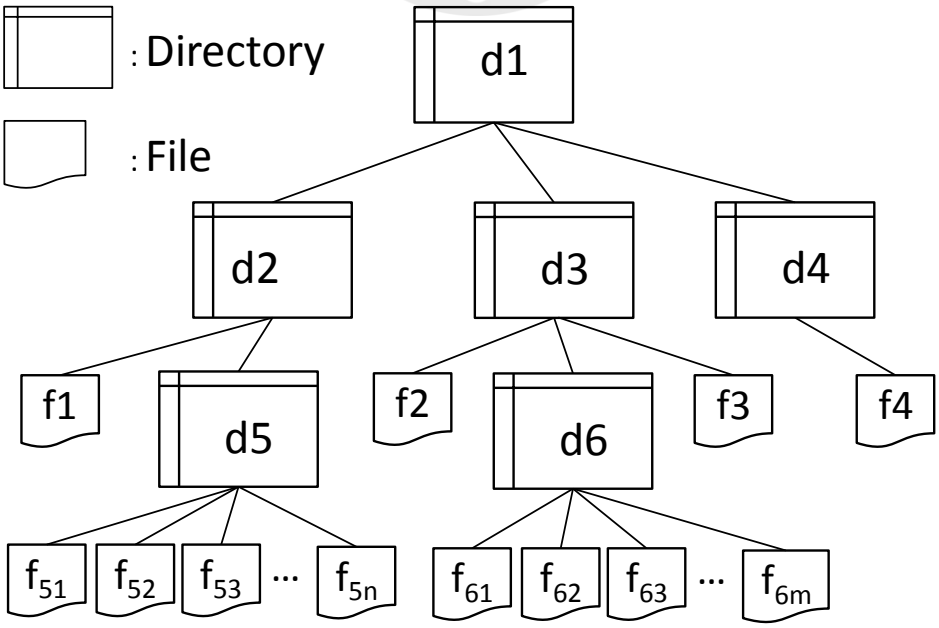


圖 18：檔案目錄結構的例子。

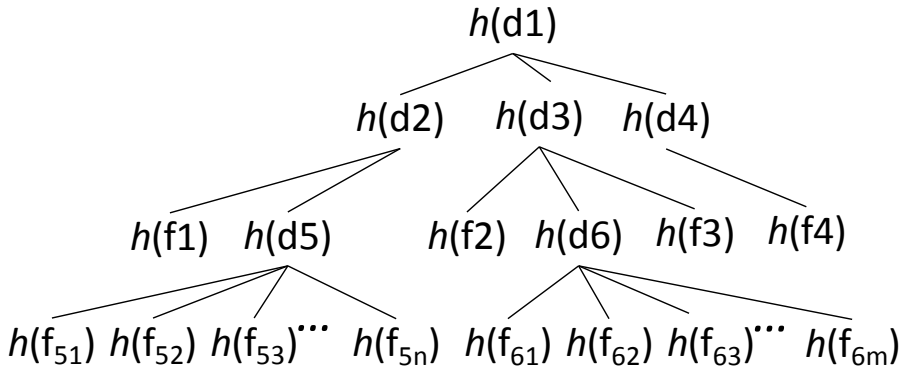
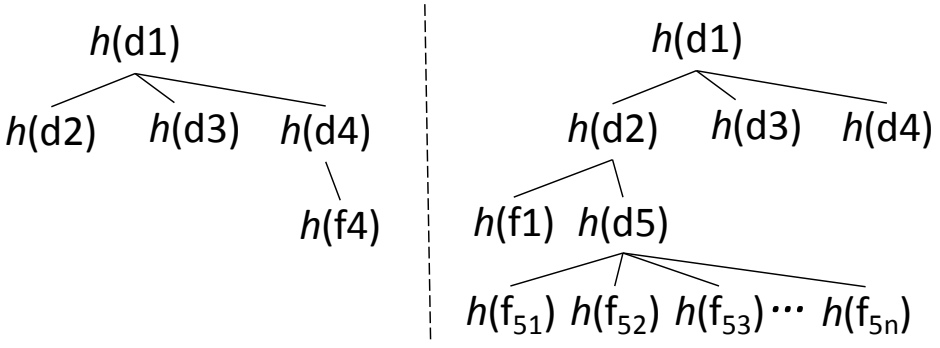


圖 19：相對於圖 18 檔案目錄結構的 m 元雜湊樹。

h(f)：代表 f 的雜湊值。目錄 d5 的雜湊值，代表在 d5 目錄內所有檔案的雜湊值串接起來取雜湊，即：*h*(d5) = hash(*h*(f51) || *h*(f52) ||…||*h*(f5n))；同樣的，*h*(d2) = hash(*h*(f1) || *h*(d5))，其餘依此類推。根雜湊也就是狀態碼 *S*，即 *h*(d1)。

使用 m 元雜湊樹的好處是，只需要 Merkle proof 就可以計算出根雜湊值，而無需下載整棵雜湊樹[93]。如圖 20，如果已知正確的根雜湊值 *S*（即本研究所稱的「狀態碼」），要確認 f4 是否遭竄改或破壞，只需下載並計算 f4 的部分雜湊樹、即 f4 的 Merkle proof 的根雜湊值 *S''*，比對 *S* 是否等於 *S''* 即可。即：*S''* = hash(*h*(d2) ||*h*(d3) ||*h*(f4))。



(a) (b)

圖 20：圖 19 的兩個部分雜湊樹，其根雜湊值(a)=(b)

4.1.2 實驗結果

筆者進行了一系列實驗來評估所提出架構的效能，筆者使用 Java 程式語言實作。信息摘要函數是「java.security.MessageDigest」，採用 SHA-256 演算法。

實驗用的電腦是 Macbook Air 2014，中央處理器是為 Intel Core i5 1.4GHz，安裝 4 GB 1600 MHz DDR3 記憶體，作業系統是 OS X El Capitan 10.11.5。硬碟內共有 717,976 個檔案，存放在 149,487 個目錄中。安裝的軟體包括 FaceTime、LINE、Pages、Mail、Maps、Calendar、Preview 和 Photos 等常用的應用軟體。

首先，包含筆者在內的團隊研究所提出的索引 Γ 函數，是否足以將大量函式庫檔案均勻分佈到定位摩克樹的葉節點中。以第三章第三節的 3.3.4 定位摩克樹的碰撞測試來看，表 2 所表示的最高碰撞率為 8，但是該測試所使用的 key-value pairs 的數量最高只有葉節點的總數量，尚不足以說明本研究的碰撞效能。因為發生碰撞的葉節點，在計算儲存有一些 PB pairs 的葉節點的 Merkle proof 的雜湊值時，需要分別計算這些 PB pairs 的雜湊值，然後將它們串接起來，並導出串接值的雜湊值，併計算所有沿路徑到根節點的內部節點所有節點的雜湊值，與其串接值的雜湊值。如果同一個葉節點中，儲存了太多的 PB pair，除了表示索引定位的 Γ 函數造成過多的碰撞外，則從該葉節點雜湊值運算，也需較多的時間，導致該 Merkle proof 所要計算的根雜湊，就會需要較長時間。

因此，首先依照筆者所用的環境，總共有 717,976 個檔案，因此筆者測試了在不同樹高底下，索引 Γ 函數的碰撞情形，如表 6。定位摩克樹的所有節點用一

維陣列儲存，相對的葉節點若發生碰撞，則另外開一個與該葉節點的相對應的 resizable 的一維陣列存放相碰撞的 PB pairs 的雜湊值。

表 6：不同樹高下索引 Γ 函數的碰撞情形

Height	No. of leafs	α	β
4	8	89747.25	89908
6	32	22436.16	22646
8	128	5609.27	5794
10	512	1402.32	1502
12	2048	350.58	421
14	8192	87.64	127
16	32768	21.91	43
18	131072	5.50	18
20	524288	1.84	10

Height: Height of an tp-Merkle tree
No. of leafs: Total number of leaf nodes in the tp-Merkle tree
α: Average number of stored PB pairs in a nonempty leaf node
β: Maximum number of stored PB pairs in a nonempty leaf node

這個實驗的結果表示，在葉節點的數量沒有嚴重不足的情形下，其最大的碰撞率會是「10」。

如圖 17 所示的系統架構，筆者用了四種方法來做完整性檢查效能的實驗與比較，包括有 4.1.1.1 的 key-value pair 方法、4.1.1. 2 的 m 元雜湊樹方法，與本研究所提出的定位摩克樹的方法外，筆者還單純地將所有檔案的 PB pairs 存放在 HVA 中，以純粹一對一的方式進行完整性檢查，此方法則稱為「pure integrity check」。「pure integrity check」和 key-value pair 方法，筆者使用 java.util.HashMap 儲存路徑與檔案的雜湊值；m 元雜湊樹方法，筆者使用 java.util.ArrayList 建構；筆者所使用的定位摩克樹方法，各節點用一維陣列儲存外，各葉節點的 PB pairs

筆者用 java.util.LinkedHashMap 儲存。

首先，筆者先比較這四種方法在 HVA 建構時間與所需儲存的空間的效能，如表 7。

Pure integrity check 與 key-value pair 所需要的 HVA 空間相同，因為兩者都需要將 717,976 個檔案的 PB pairs 儲存在 java.util.HashMap 物件中，唯一差別的是時間：516.1 秒 - 509.2 秒 = 6.9 秒，相差 6.9 秒是因為 key-value pair 方法，需要將所有的 PB pairs 串接後計算出狀態碼。m 元雜湊樹方法需要將 149,487 個目錄與 717,976 個檔案的雜湊值儲存到 java.util.ArrayList 物件中，需要較多的空間。而定位摩克樹方法，會依照樹高所產生的節點數不同，而需要不同的儲存空間；當樹高越高，內部節點數越多，因此所需空間越多。

表 7：四種方法的 HVA 建構時間與所需空間

Scheme		**Build-up time (s)**	**Memory required (MB)**
Pure integrity check		509.2	57.5
Key–value pair		516.1	57.5
m-ary hash tree		566.5	87.2
tp-Merkle tree	Height=4	518.0	57.5
	Height=6	518.3	57.5
	Height=8	518.8	57.6
	Height=10	518.6	57.7
	Height=12	518.3	58.1
	Height=14	518.8	59.9
	Height=16	518.9	67.1
	Height=18	520.8	133.5
	Height=20	520.1	206.9

接下來，筆者測試檢查一個檔案的完整性，以及當有一個檔案更動後，重新計算狀態碼並更新狀態碼，分別所需要的時間。pure integrity check 不需要計算狀態碼；key-value pair 方法需要全部的 PB pairs 都計算一次；m 元雜湊樹方法只需要計算遭檢查或更動檔案的「部分雜湊樹片段」（稱為 Merkle proof），但會有時大有時小，所以不同的檔案，所在目錄不同，其所需的時間會依照目錄的大小不同，需要不同的時間，因此完全依目錄結構而定；而定位摩克樹方法，除了樹高不同外，不同檔案所對應到的部分雜湊樹片段的大小，皆大約相同，只有在不同葉節點，其所存放的 key-value pairs 數不見得相同；當樹高超過 12（含）以上，其稽核與更新狀態碼所需的時間，分別小於 1 ms 與 2 ms。參照表 8 的實驗數據。

表 8：四種方法的檢查檔案與更動檔案所需的時間（單位：ms）

Scheme		**γ**	**δ**
Pure integrity check		Nil	Nil
Key–value pair		17.20	18.11
m-ary hash tree		6.83	18.51
tp-Merkle tree	Height=4	27.57	34.81
	Height=6	7.10	9.13
	Height=8	2.26	9.13
	Height=10	0.47	2.18
	Height=12	0.22	1.90
	Height=14	0.09	1.92
	Height=16	0.10	1.64
	Height=18	0.07	1.71
	Height=20	0.08	1.57

Height: Height of an tp-Merkle tree

γ: Average running time for verifying the hash value of a file according to status code (in ms)

δ: Average running time for calculating the new status code when a hash value of a file is updated (in ms)

最後筆者以常用的八種軟體，在軟體載入前自動對所需執行環境的連結庫檔案做完整性的檢查實驗，記錄其所需時間。我們可以用 Mac OS 的命令列指令「otool」加參數： -L <軟體名稱> 得知，每一種軟體在執行時，會使用到的動態連結的函式庫檔案有哪些？例如 FaceTime 執行時，會載入/System/Library 與 usr/lib 目錄下的動態連結檔共 64 個。

筆者用 overhead 來比較三種方法的效能，雖然三種方法皆需要透過網路下載「狀態碼」供稽核之用，但是「狀態碼」僅僅 32-byte 非常小，因此可忽略下載所需時間。overhead 的計算方式如下：

Overhead＝｛〔（該方法的執行時間）－（pure integrity check 的執行時間）〕

÷（pure integrity check 的執行時間）｝× 100%

實驗發現，4.1.1.1 的 key-value pair 方法相較於 pure integrity check 方法是非常沒有效率的，如果用 tp-Merkle tree 的方法，當樹高超過 12（含）以上，他相較於 pure integrity check 方法的 overhead 幾乎都在 1%以下。如表 9 所顯示的實驗結果。

表 9：軟體執行時包含完整性檢查所需要的時間(單位：ms)

Scheme		Facetime	Line	Pages	Mail	Maps	Calendars	Preview	Photos
No. of linked library files		64	26	36	46	37	44	30	68
Pure integrity check		2214.32	1241.71	3283.03	3059.24	767.9	1069.32	703.11	2481.5
Key–value pair (Overhead)		13251.24 498.43%	5839.72 370.30%	8985.14 173.68%	8985.14 193.70%	7089.80 823.27%	9146.04 755.31%	6577.50 835.49%	13990.68 463.80%
m-ary hash tree (Overhead)		2631.32 18.83%	1600.37 28.88%	3986.86 21.44%	3986.86 30.32%	1671.14 117.62%	1960.27 83.32%	1623.95 130.97%	5923.20 138.69%
td-Merkle tree	Height=4 (Overhead)	3664.80 65.50%	1853.77 49.299%	4031.34 22.79%	4048.33 32.33%	1590.14 107.08%	2081.20 94.63%	1441.42 105.01%	4031.62 62.47%
	Height=6 (Overhead)	2649.86 19.67%	1413.86 13.86%	3502.17 6.67%	3356.66 9.73%	1008.40 31.32%	1368.88 28.02%	919.38 30.76%	2922.44 17.77%
	Height=8 (Overhead)	2360.94 6.62%	1298.14 4.54%	3353.99 2.16%	3159.45 3.28%	847.26 10.33%	1166.02 9.04%	778.95 10.79%	2626.08 5.83%
	Height=10 (Overhead)	2247.48 1.50%	1256.75 1.21%	3296.95 0.42%	3079.50 0.66%	784.00 2.10%	1089.58 1.89%	730.28 3.86%	2515.72 1.38%
	Height=12 (Overhead)	2228.88 0.66%	1245.80 0.33%	3287.03 0.12%	3069.25 0.33%	772.44 0.59%	1077.40 0.76%	709.20 0.87%	2490.30 0.35%
	Height=14 (Overhead)	2222.06 0.35%	1242.53 0.07%	3285.48 0.07%	3063.10 0.13%	769.72 0.24%	1073.20 0.36%	709.20 0.87%	2487.20 0.23%
	Height=16 (Overhead)	2217.72 0.15%	1243.25 0.12%	3286.10 0.09%	3062.69 0.11%	770.40 0.33%	1070.26 0.09%	704.86 0.25%	2488.44 0.28%
	Height=18 (Overhead)	2220.82 0.29%	1243.17 0.12%	3285.17 0.07%	3062.69 0.11%	769.04 0.15%	1072.36 0.28%	704.55 0.22%	2485.96 0.18%
	Height=20 (Overhead)	2219.58 0.24%	1242.48 0.06%	3285.11 0.06%	3061.05 0.06%	769.72 0.24%	1071.94 0.25%	705.48 0.34%	2485.96 0.18%

4.1.3 相關研究

信任平台模組（trusted platform module，簡稱 TPM）是國際標準的一種安全加密處理器 [106, 107]。具有 TPM 功能的電腦，可以在啟動過程中檢查主機系統的完整性，使得在作業系統啟動之前、安全的開機過程中，或是在應用軟體中，

以硬體模組執行保護和偵測的機制。Sule et al.（2015）[108]展示了在網路上實現基於 TPM 的可信雲端運算上部署實作的原型。Berger et al.（2015）[109]提出了一個稱為「可擴容存證」（scalable attestation）的解決方案，以基於 TPM 技術結合安全啟動與信任啟動，解決雲端環境的完整性和監控的問題。然而，只有在啟動時執行完整性檢查是不夠的，因為惡意軟體或電腦病毒可以在任何時候感染執行環境。

Mishra（2010）的研究蒐集整理各種病毒偵測方法及其限制 [91]，並討論這些方法在完整性檢查上的應用及其優勢。在 Symantec Corporation 推出的商用 Norton AntiVirus 產品中，完整性檢查稱為「防疫接種」(inoculation)。然而，這種完整性檢查，無法抵禦在雲端運算環境的 IaaS 和 PaaS 服務模型中的 roll-back 攻擊，因為當惡意軟體或病毒在感染時，會同時修改系統中所存放的檔案與其雜湊值。其他相關的研究還有「進階環境入侵偵測系統」（advanced intrusion detection environment ，簡稱 AIDE），是一個檔案目錄的完整性檢查器 [110]。Yue et al.（2016）[111] 提出了一種保護雲端的虛擬機器 image 完整性方法，為了減少啟動所需時間，系統會定期的啟動掃描程序，來驗證整個 image 的完整性。Wang et al.（2014）[112] 提出了一個動態完整性驗證架構，利用一個可信任的第三方（TTP）來收集系統的完整性資訊，並從 TTP 遠端定期的掃描虛擬機器的完整性。然而，攻擊者可以在兩個連續檢測周期之間選擇攻擊時間以逃避掃描。Kaczmarek 和 Wrobel（2014）[113] 提出了一個實作在 Linux 核心的安全模組的

以實現檔案完整性驗證的系統，其初始檔案的雜湊值資料庫會存放在 Linux 核心空間，當 Linux 執行系統呼叫時，Linux 核心會啟動完整性檢查。然而，這種方案無法應用於雲計運算的環境，因為作業系統核心的 image 是由雲服務提供商所控制。

近年來，軟體式的入侵偵測系統（intrusion detection systems，簡稱 IDS）引起了廣泛關注 [114 - 119]。IDS 是一種監控並記錄系統環境與網路狀況的工具，用以收集有用數據（如可疑活動和環境情境），並分析該數據以檢測是否有遭惡意入侵的機制。任何一種入侵偵測系統，希望透過整合各種數據資料，以建立具有低錯誤率的有效預測模型。有一些研究便是專注於虛擬機器的入侵檢測。Garfinkel 和 Rosenblum（2003）[120] 提出了一種稱為「虛擬機器內在檢視法」（virtual machine introspection，簡稱 VMI），主機活動的狀況，由硬體狀態與 VM monitor 依照預存的結構知識庫所推斷的軟體狀態，進行分析。Ibrahim et al.（2011）[121] 提出 CloudSec 監控設施，為 IaaS 雲端平台上托管的虛擬機器提供主動、透明和即時的安全監控。CloudSec 利用 VMI 技術，對虛擬機器的實體儲存空間進行細微的檢測，而無需在虛擬機器中安裝任何安全防疫資訊碼。Hizver 和 Chiueh（2014）[122] 開發了一個即時的核心資料結構監控系統，運用作業系統的揮發性分析能力（一種開源的電腦鑑識框架）來簡化和自動化虛擬機器執行狀態的分析。Garfinkel et al.（2003） [123] 提出一種信任計算架構稱為 Terra，該架構使用「信任虛擬機器監視器」（TVMM）將一個防篡改的硬體平台，

劃分為多個獨立的虛擬機器。硬體和 TVMM 可以作為一個可信方，讓虛擬機器以加密方式識別其運行的軟體。然而，虛擬機器監控器是由雲端服務提供商維護和掌控，這些方案接無法讓用戶稽核雲端的平台。

Wei et al.（2009）[124] 提出了一個 VM image 管理系統，用於控制對 image 的存取、追踪其來源，並為用戶和管理者提供有效的 image 過濾器和掃描器，以偵測和修復安全漏洞。然而，在該系統中，需要定期對虛擬機器的 image 進行病毒掃描以偵測和修復漏洞。Haeberlen et al.（2010）[125] 提出了一種「可歸責的虛擬機器監視器」(accountable VM monitor)，它可以透過查詢虛擬機器中的日誌檔案、發送與接收的訊息，來偵測出故障的虛擬機器。Santos 和 Lopes（2014）[126] 提出了建構可靠虛擬機器的方案，它基於「trusted computing」與「model checking」兩項功能：「trusted computing」可對虛擬機器的軟體進行低階蒐證，「model checking」提供對軟體的高階配置屬性做自動驗證。Win et al. （2014）[127] 提出了一個虛擬安全解決方案，旨在為虛擬環境提供全面性的保護；當虛擬機器被新增時，「安全監視器」會產生記憶體內容的雜湊值並存儲。並會周期性的計算虛擬機器的執行程序表的雜湊值，並與原先儲存的雜湊值進行比較。然後這些雜湊值會傳遞到「控制監視器」，與先前獲取的值進行比較；這些分析結果可以知道是否存在惡意軟體攻擊。

Viswanathan 和 Mishra（2016）[128] 提出了一個軟體系統，運用作業系統的核心功能，進行檔案系統監控，偵測對動態和互動式網站內容的更改，並驗證

修改的真實性。Web 伺服器內的每個頁面的 SHA-1 雜湊值，使用 TPM 擁有者授權值，存放在 TPM 的 NV-RAM 記憶體中，然而，NV-RAM 是無法支援真實電腦的執行時環境。

第二節 利用公有區塊鏈的自動給付系統

在雲端的交易環境中，客戶與供應商間的交易是最單純的，例如雲端硬碟的租用。而現實環境中的交易，經常會牽涉到多方的交易互動。例如版權交易：作者會透過代理商銷售，客戶向代理商購買；而在費用的支付上，通常最大的問題在於代理商與資產擁有者，或者是版權擁有者之間的信任問題，代理商使否有誠實地依照銷售量支付應給付的版稅？作者如何確保代理商回報的交易數字是正確的？如果違約，如何提出「詐欺證明」(fraud proof) 要求賠償？一般的資訊化作業，並無法解決這個問題，因為操作方是代理商，版權擁有者無法稽核。

公有區塊鏈具有加密貨幣的交易功能，並且提供了可信任的去中心化的平台，因此又稱為「分散式帳本」。由於公有區塊鏈記載的交易內容公開透明，難以被竄改，因此可以提供交易多方一個可信任的平台。但是公有區塊鏈卻有擴容上的限制，這是由於去中心化共識的機制，導致交易速度慢，且每一筆交易記錄都需要礦工費，若是一筆交易作業需要多個記錄程，那麼在區塊鏈上造成過多的資料記錄，會造成交易成本的大幅提升，尤其是在微支付(micropayment)的交易上，因此全然將所有的交易作業搬上供有區塊鏈反而是不可行的；此外還有隱私權的

問題：所有記載在公有區塊鏈的資訊都是公開透明的；因此大大限制公有區塊鏈的應用範疇。

筆者以路跑賽為例，攝影師負責拍攝選手的路跑英姿供選手下載，主辦方從所收的報名費中，依每位攝影師所拍攝照片被下載的量支付版稅給攝影師，如果主辦方沒有依實際的下載量已付版稅，攝影師亦無從稽查。直覺上，這個問題，很容易以公有區塊鏈來解決，直覺式的解決方案之系統架構如圖 21。

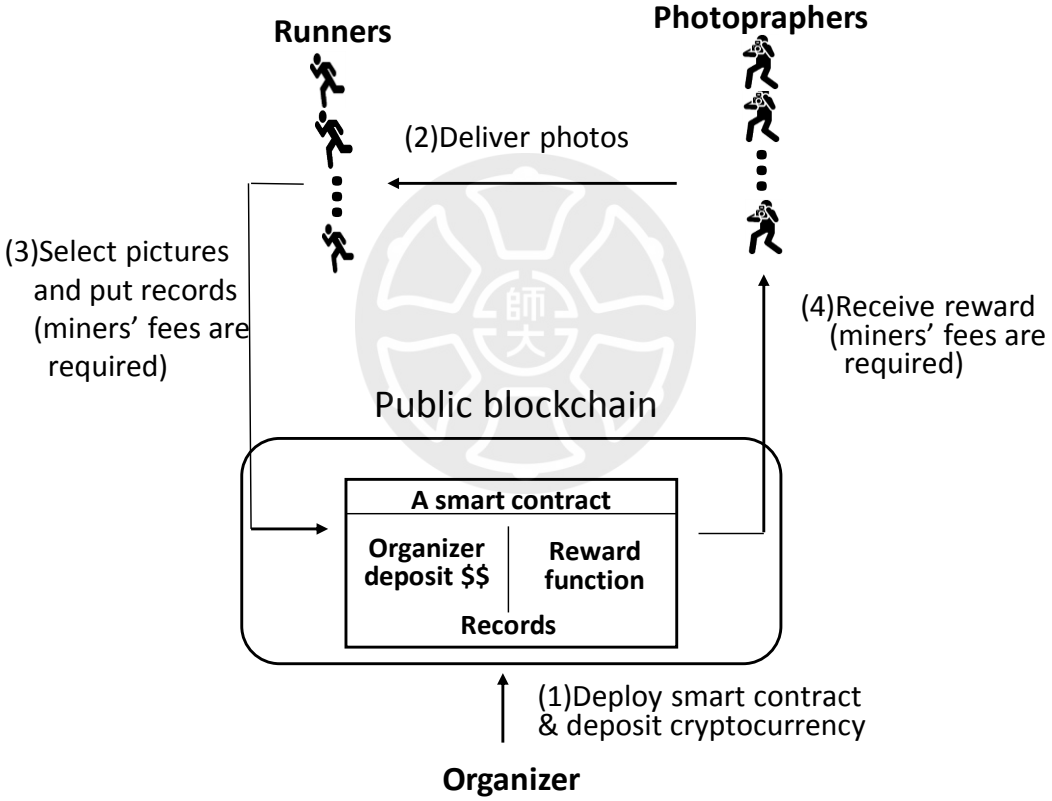


圖 21：直覺式方案–基於公有區塊鏈的自動支付系統

系統的執行步驟如下：

步驟（1）：主辦方先在以太坊公有區塊鏈部署智能合約，包含費用提領程序並將報名費中的一定比例打入以太坊的帳戶中，以供支付攝影師費用之用。

步驟（2）：每一位攝影師分別將照片傳給路跑選手供其觀賞與下載之用。

步驟（3）：每一位路跑選手將其所做的選擇儲存在智能和約中。

步驟（4）：每一位攝影師啟動提領程序，該程序將會依照路跑選手選擇的記錄，將費用打入每一位攝影師自己在以太坊的數位錢包。

假設在步驟（3）中每一筆交易的礦工費是δ1，步驟（4）中每一筆交易的礦工費是δ2，假設有選照片的路跑選手有 N 位，每位選手選擇並下載的找片數有 K 張，攝影師總共有 P 位，因此所需要的總礦工費則是：$N \times K \times \delta 1 + P \times \delta 2$。

很明顯地，這個直覺式的解決方案的架構，雖然解決了信任的問題，主辦方也難以做假，但是所需要在區塊鏈上記錄的交易次數過多，以致產生的礦工費會過高，如果需要路跑選手在區塊鏈上記錄照片的下載次數而支付礦工費，也不合理。因此，筆者提出了一個基於公有區塊鏈的創新自動給付與賠償機制，能夠將總礦工費，由 $N \times K \times \delta + P \times \delta$降到只需$\delta + P \times \delta$，而且最重要的是，路跑選手無需支付礦工費，只有每一位攝影師在提領照片的版費時，分別需要支付一次交易的礦工費而已，才合乎常理，而且本研究所提出的系統架構同時是一個可公開稽核的自動給付與違約賠償機制。

4.2.1 系統架構

為了解決這些問題，讓公有區塊鏈能夠應用在微支付的交易上，讓多方交易作業安全且在多方都可信任的環境下進行，並避免客戶在購買時需要支付高額的礦工費，必須要有一個去信任化（trustless）的多中心安全機制，也就是同時需

要有快速稽核機制，而讓多方都能夠提出「詐欺證明」(fraud proof)，因此，筆者提出了一個自動給付與違約賠償的架構，足以解決現實環境中的多方交易模式，而且可以避免不必要的礦工費支付，如圖 22。

筆者採用了 public blockchain-based Proof of Violation 機制，Proof of Violation 協議請參照第三章第二節的說明，以「定位摩克樹」存證，並將存證的「定位摩克樹」公布到網際網路上 P2P 的「星際系檔案系統」(Inter-planetary File System，簡稱 IPFS)以公開稽核機制，完成研究的系統架構。系統架構圖，請參照圖 21。

本研究架構以路跑賽為例子，攝影師提供路跑者的攝影照片，由主辦方（organizer）從收取的報名費中支付給攝影師相片費，支付費用的多寡，依照相片被下載的數量而定。筆者使用以太坊區塊鏈進行實作，主辦方利用以太坊的智能合約，將費用儲存在以太坊區塊鏈的帳戶中，並負責支付相片費用給攝影師，若被發現支付不實，則會啟動賠償機制。

依照 PoV 機制，筆者要先定義，跑者在選擇並下載照片後，與主辦方之間交換訊息的格式，同時需要有 chain-hash 的機制 [99]，定義如下：

$M^{i}_{request}$ = (PK_R, P_addr, IndexValue, Number_of_Downloaded_Photos, SIG_{runner})

M^{i}_{reply} = ($M^{i}_{request}$, $Reward_{accumulated}$, hash($Receipt^{j-1}$), $SIG_{organizer}$)

其中：**PK_R** 是跑者的公鑰。

P_addr 是攝影師的乙太坊地址。

IndexValue 是「回條」(Receipt)在定位摩克樹存證的索引值。「indexValue」

是由「PK_R」串接（concatenation）「P_addr」組成。

Number_of_Downloaded_Photos 是跑者選擇並下載的照片數。

SIG_{runner} 是跑者的數位簽章。

主辦方收到跑者的訊息後，會回覆訊息：M^{i}_{reply} 給跑者，M^{i}_{reply} 稱為「回條」（Receipt），該回條主辦方會同時存證，存證在「定位摩克樹」公布到網際網路上 P2P 的「星際系檔案系統」以公開稽核機制。

$M^{i}_{request}$ 是用戶端作業要求的訊息。

$Reward_{accumulated}$ 是到目前為止 P_addr 的這位攝影師所累計的報酬，因此該攝影師可以依此往前追溯 $Reward_{accumulated}$ 的累積值是否有誤。。

hash($Receipt^{j-1}$)是屬於 P_addr 的這位攝影師的前一次回條的雜湊值。

跑者會存放所有的回條，以作為未來即核與申訴之用。

首先，主辦方先在公有區塊鏈上佈署智能合約，包含開啟智能合約帳戶，以及違約賠償判別函式，並儲存一筆費用在智能合約的帳戶中；當跑者從攝影師方提供的照片中選擇想要到照片並下載，此時會回傳信息給主辦方，主辦方會依回條與其預先定義的機制累計費用，作為計算版權費之用，主辦方會將這些回條以定位摩克樹存證；攝影師一段時間後檢查存放回條的定位摩克樹並取得累計最高的回條，即「MaxR」（詳見系統執行步驟 7、8），再以此證據透過智能合約收取費用。

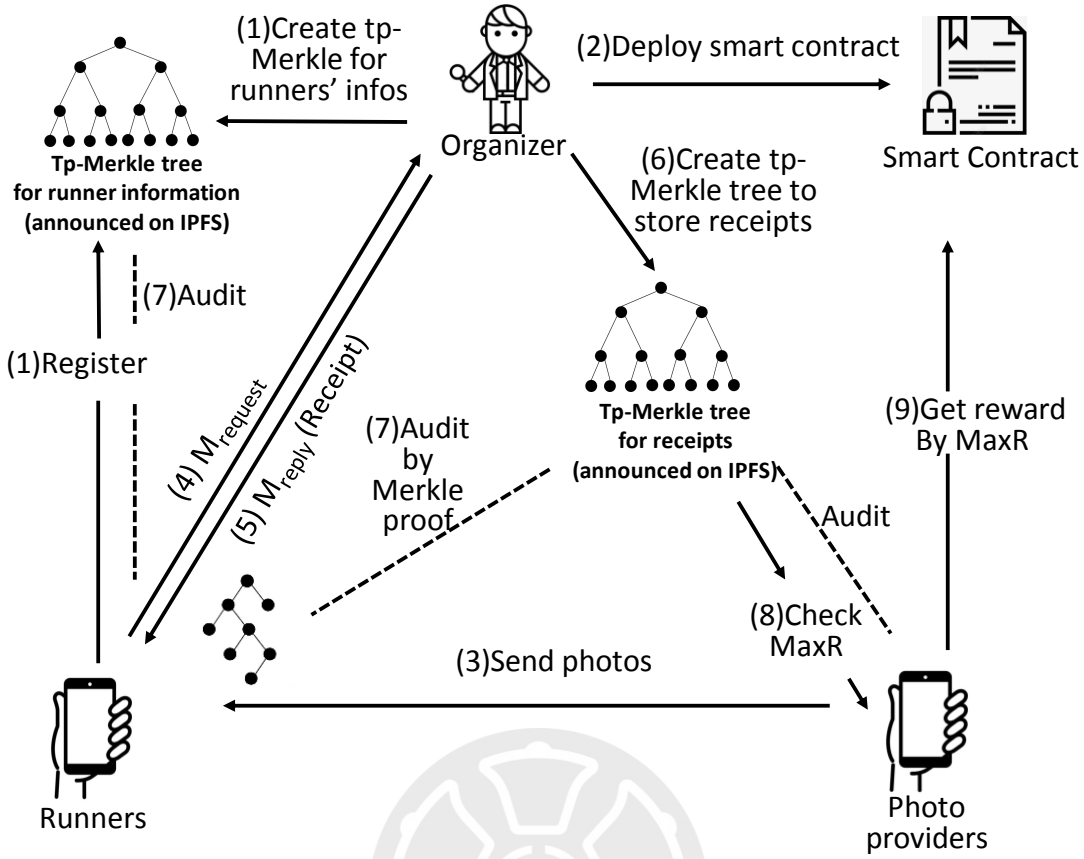


圖 22：自動給付與違約賠償機制系統架構圖

系統的執行步驟如下：

步驟（1）：跑者向主辦方註冊並繳付報名費。主辦方為每一個報名成功的跑者開各一個 key-value pair「<H(PK_R), H(ID)>」，並建立一個定位摩克樹存放所有已報名並註冊跑者的 key-value pair。「PK_R」和「ID」分別是每一位跑者的 public_key 和身分 id。這顆存放所有跑者註冊資訊的定位摩克樹會被公開存放在「星際檔案系統」上。

步驟（2）：主辦方運用智能合約，將報名費中的一定比例打入以太坊的帳戶中，以供支付攝影師費用或違約賠償之用。定位摩克樹的「根雜湊」同時放上區塊鏈。攝影師也可以利用智能合約檢查目前帳戶的餘額。

步驟（3）：攝影師透過專用的 app 將照片傳遞給跑者，供其選擇並下載。

步驟（4）：跑者選好所要的照片下載，同時傳送訊息「$M_{request}$」給主辦方。

步驟（5）：主辦方收到「$M_{request}$」，會回復訊息「M_{reply}」給該跑者。「M_{reply}」筆者稱之為「回條」（Receipt）。

步驟（6）：主辦方將所有的「回條」的雜湊值建立一個定位摩克樹存放，並將此定位摩克樹同樣公開存放在「星際檔案系統」上。這顆定位摩克樹的「根雜湊」同樣放上區塊鏈。

步驟（7）：每一位跑者都可以透過他手中的「回條」下載在步驟 6 所建立的定位摩克樹的「Merkle proof」進行稽核，如果發現該回條有被竄改、刪除或任何其他錯誤，則智能合約會自動啟動賠償機制。同樣的，每一位攝影師也可以執行類似的稽核，例如：追溯 MaxR 是否正確？

步驟（8）：每一位攝影師都可以下載在步驟 6 所建立的定位摩克樹進行查核，並依此取得屬於自己累積最高金額的回條，筆者假設這個回條為「MaxR」。

步驟（9）：每一位攝影師分別用自己的回條 MaxR 啟動智能合約中的支付程序，將費用打入自己在以太坊的數位錢包。

為了讓主辦方誠實記載所有記錄，就第（7）與第（8）步驟若經稽核不實，在智能合約上會啟動賠償機制。任何一方也都可以提出申訴啟動稽核與賠償機制，同時為了避免濫用申訴機制，申訴者必需申訴時，同時經由智能

合約打入一筆申訴保證金，若申訴失敗則會沒收保證金，申訴成功，除了會退回保證金外，同時會獲得申訴成功賠償金。

1、是否有將跑者上傳的回條誠實記錄的申訴(攝影師也可以執行申訴)：

步驟（1）：跑者先檢查報名註冊之定位摩克樹，確認資訊正確。

步驟（2）：跑者依手上的回條與存證回條的定為摩克樹進行稽核。

步驟（3）：跑者發現稽核有誤，將該回條發上智能合約申訴。

步驟（4）：跑者經智能合約打入申訴保證金，上傳回條與 Merkle proof。

步驟（5）：智能合約檢查數位簽章是否正確？，若通過，則進行步驟 6，若未通過則駁回申訴，並沒收申訴保證金（不退回）。

步驟（6）：智能合約檢查回條的 key-value pair 以及 Merkle proof，並與智能合約的 root hash 進行比對檢查。

步驟（7）：若未通過，則申訴成功。申訴者經由智能合約獲取賠償金並退回申訴保證金。

2、MaxR 累積是否正確的申訴：

步驟（1）：攝影師（或其他稽核申訴者）從 IPFS 下載儲存回條的定位摩克樹以及回條進行稽核。

步驟（2）：攝影師（或其他稽核申訴者）依照 $\text{hash}(\text{Receipt}^{i-1})$ 查核 $\text{Reward}_{\text{accumulated}}$ 是否累積有誤。

步驟（3）：若經攝影師（或其他稽核申訴者）發現稽核有誤，將該回條與

Merkle proof 發上智能合約進行申訴，同時打入申訴保證金。

步驟（4）：智能合約檢查回條與 Merkle proof 數位簽章是否正確，正確則進行步驟 5，若錯誤則駁回申訴，並沒收申訴保證金（不退回）。

步驟（5）：智能合約檢查回條的 key-value pair 以及 Merkle proof，並與智能合約的 root hash 進行比對檢查。

步驟（6）：若未通過，則申訴成功。申訴者經由智能合約獲取賠償金並退回申訴保證金。

4.2.2 實驗結果

筆者實驗所使用的電腦作業平台為 Microsoft Windows 10 企業版，執行在 Intel 微處理器 Core i5-7400 3.4GHz、記憶體 16GB。使用以太坊的 Ropsten test network 實作 [129]，以 Solidity 語言撰寫智能合約。回條以 json 撰寫，因此一筆回條大約是 614 bytes；定位摩克樹則以 Python 語言撰寫儲存。

跑者、攝影師需要稽核時，有可能需要下載所有回條、key-value pairs 以及定位摩克樹，因此筆者先實作在一般現實的情形下，定位摩克樹在不同的樹高下，以及不同數量的回條，所需的空間是否是目前電腦或智慧型手機都可容許的範圍？如表 10、表 11。實驗結果顯示是可以的。

表 10：不同高度的定位摩克樹所需空間

Height of tp-Merkle tree	Storage space of tp-Merkle tree (MB)
15	2.4
17	9.7
19	38.8
20	77.6
21	155.2

表 11：回條所需儲存空間

Amount of Receipt	Storage space of Receipt (MB)
50,000	38
100,000	76.7
500,000	387.3
1,000,000	774.9

本研究針對定位摩克樹的效能測試（請參照第三章第三節之 3.3.4），發現當發現定位摩克樹高超過 10 以上，其效能最好；但是在本情境的實驗結果發現，葉節點所儲存的 key-value pairs 數量在 10（含）以下時，其執行效能最好，且不會消耗過多的 Gas。筆者做了稽核時間的測試，證明確實是如此，如表 12、表 13。

表 12：5 萬筆回條所需的稽核時間

Height of tp-Merkle tree	Maximum collision times	Time of scan & audit Receipt (s)
15	25	4.95
17	11	4.88
19	7	4.95
20	5	5.02
21	4	5.08

表 13：100 萬筆回條所需的稽核時間

Height of tp-Merkle tree	Maximum collision times	Time of scan & audit Receipt (s)
15	306	110.69
17	93	118.91
19	36	146.62
20	22	157.48
21	10	165.13

此外，本研究提出的系統架構，對於礦工費的消耗需要注意，因為礦工會依照支付 Gas 的大小決定交易執行的優先順序，Gas 以 Gwei 為單位，1Gwei = 10^{-9} ETH。本研究架構所需要在以太坊區塊鏈的基本作業包含：佈署智能合約、將金額打入智能合約帳戶（存款）、打上 root hash、攝影師支領費用、回條是否記錄正確申訴等，其所需礦工費，如下表(Assume 1 ETH = USD 1,000)。

表 14：佈署智能合約所需的礦工費

Gas consumption	Gas Price	ETH	USD
3391882	Fast (20 Gwei)	0.06784	67.84
	Use(10 Gwei)	0.03392	33.92
	Average (4 Gwei)	0.01357	13.57
	Cheap (3 Gwei)	0.01018	10.18

表 15：將金額打入智能合約帳戶所需的礦工費

Gas consumption	Gas Price	ETH	USD
21,459	Fast(20 Gwei)	0.000429	0.429
	Use(10 Gwei)	0.000215	0.215
	Average(4 Gwei)	0.000086	0.086
	Cheap(3 Gwei)	0.000064	0.064

表 16：上傳 root hash 所需的礦工費

Gas consumption	Gas Price	ETH	USD
43956	Fast (20 Gwei)	0.000879	0.879
	Use(10 Gwei)	0.000440	0.440
	Average (4 Gwei)	0.000176	0.176
	Cheap (3 Gwei)	0.000132	0.132

表 17：攝影師支領費用所需的礦工費

Height of tp-Merkle tree	Ψ	Gas consumption	ETH		USD	
			Gas price (3 Gwei)	*Gas price (20 Gwei)*	*Gas price (3 Gwei)*	*Gas price (20 Gwei)*
15	1	198166	0.000594	0.001982	0.5945	1.982
	10	254755	0.000764	0.002548	0.764	2.548
17	1	212465	0.000637	0.002125	0.637	2.125
	10	268835	0.000807	0.002688	0.807	2.688
19	1	224267	0.000673	0.002243	0.673	2.243
	10	283197	0.000849	0.002832	0.8496	2.832
20	1	235560	0.000706	0.002356	0.7067	2.356
	10	281276	0.000844	0.002813	0.844	2.813
21	1	238281	0.000715	0.002383	0.715	2.383
	10	288410	0.000865	0.002884	0.865	2.884

Ψ is the number of key-value pairs in the leaf node.

表 18：未記錄回條之申訴所需的礦工費

Height of tp-Merkle tree	Ψ	Gas consumption	ETH		USD	
			Gas price (3 Gwei)	*Gas price (20 Gwei)*	*Gas price (3 Gwei)*	*Gas price (20 Gwei)*
15	1	193263	0.00058	0.001933	0.58	1.933
	10	251157	0.000753	0.002512	0.753	2.512
17	1	207562	0.000623	0.002076	0.623	2.076
	10	265237	0.000796	0.002652	0.796	2.652
19	1	219365	0.000658	0.002194	0.658	2.194
	10	277678	0.000833	0.002777	0.833	2.777
20	1	230867	0.000693	0.002309	0.694	2.309
	10	279599	0.000839	0.002796	0.839	2.796
21	1	233379	0.0007	0.002334	0.7	2.334
	10	284812	0.000854	0.002848	0. 854	2.848

Ψ is the number of key-value pairs in the leaf node.

表 19：未正確計算 MaxR 之申訴所需的礦工費

Height of tp-Merkle tree	Ψ	Gas consumption	ETH		USD	
			Gas price (3 Gwei)	*Gas price (20 Gwei)*	*Gas price (3 Gwei)*	*Gas price (20 Gwei)*
15	1	361500	0.001084	0.003615	1.085	3.615
	10	461810	0.001385	0.004618	1.385	4.618
17	1	389901	0.001169	0.003899	1.17	3.899
	10	489858	0.001469	0.002652	1.47	4.899
19	1	416330	0.001249	0.004163	1.249	4.163
	10	518726	0.001556	0.005187	1.556	5.187
20	1	426449	0.001279	0.004264	1.279	4.265
	10	523824	0.001571	0.005238	1.572	5.238
21	1	444246	0.001333	0.004442	1.333	4.443
	10	538101	0.001614	0.005381	1.614	5.381

4.2.3 相關研究

Hasan & Salah（2018）[34, 35] 運用乙太坊區塊鏈，提出一個去中心化的「送交證明」（Proof of Delivery）架構，讓買方在購買實體或數位產時，確認送達並自動付費；但是當爭議（dispute）發生時，仍需要一個公正的第三方擔任離鏈的仲裁者。Yitzchak et al.（2018）[130] 同樣的運用乙太坊區塊鏈，提出一個激勵式 IoT 設備的軟體更新網路架構，稱為「Proof of Distribution」。目前存在於網際網路上 IoT 節點已超過百億個以上，如此的龐大數量很容易成為網路駭客的攻擊目標，因此為了避免 IoT 設備有安全性漏洞，因此 IoT 大廠會定期發布安全性更新，俗稱「補丁」（patches），以避免設備遭到攻擊與破壞，造成維護的困擾。為了激勵「參與者」（distributor）參與更新 IoT 軟體的佈建，當 IoT 設備更新完成，會透過智能合約產生一個「zero-knowledge contingent payment」（ZKCP）的「零知識證明」交給參與者，參與者以此 ZKCP 到智能合約領取酬勞。然而，產生「零知識證明」 需要複雜的數學運算，經由智能合約產生，會消耗較多礦工費（以太坊區塊鏈稱為：「Gas」），如果 IoT 軟體更新的量與次數多，也會消耗過多的 Gas，顯然不見得划算。

Feng Liu，Zhefu Feng 與 Jiayin Qi（2022）[131] 提出一項「基於區塊鏈技術具有多方認證的數位資產平台」（Blockchain-Based Digital Asset Platform with Multi-Party Certification，簡稱 BDAP），提供應用程式介面（application interface，簡稱 API），可與企業與銀行間的供應鏈金融（supply chain finance，簡稱 SCF）

整合。所用的主要技術有：threshold ECDSA 做加密與數位簽章，以及可驗證拜占廷容（verifiable Byzantine fault tolerant，簡稱 VBFT）共識演算法等安全性架構，以足以與現行的 SCF 整合，達到多方認證，然而它屬於私有鏈，並無整合連結公有鏈，其公信力仍然會讓人質疑。

第五章、結論與未來探討

Web 3.0 時代的元宇宙空間（Metaverse）[132]，已經讓過去雲端運算與其可提供的服務類型，益加的多樣化與廣泛，虛與實之間，人類的生存、生活與活動所需，不僅是能在元宇宙中重現，反而有逐漸的擴增與多樣化的現象，像是自 2002 年起迄今的「Second Life」虛擬世界 [133] 即為一例。

在 Web 3.0 的元宇宙空間中，可以在去中心化的平台上，建立各種類中心化的服務，或者是去中心化的應用軟體。然而目前 Web 3.0 的去中心化平台效能不彰，擴容性差，以目前的這種困境之下，如何突破此種情形，而提出一個有效率而具公信力的監督機制，並能應用在各種實際情境之下，便是本研究所要達到的目的。

國際電信聯盟 ITU 所定義的國際標準 X.800 提到，網路安全所需達到的目標有六項：

一、機密性（Confidentiality），為使資料不受到未經授權者所知悉或竊取，可以利用加密演算法，如 RSA、DES 等加密演算法，或安全的通道技術，如虛擬私人網路 VPN、Secure Soket Layer(SSL)等技術達到。

二、完整性（Integrity），為使資料不受到未經授權的篡改，可以利用數位簽章、雜湊函數、資料的加解密或防火牆等技術達到。

三、可用性（Availability），為使資料不會因外在因素或人為因素導致不見或停擺，可以使用容錯系統或備援系統來達到。

四、授權管理（Authorization 或稱 Access Control），資料的使用必須有合法的授權，因此可以用使用者登入帳號、密碼與權限管理來達到。

五、隱私性（Privacy），與機密性有類似的技術予以達成。

六、不可否認性（Non-repudiation），包含了完整性與鑑別性（authentication）兩種性質，可以利用數位簽章、數位憑證等技術達到。

但是事實上，目前沒有任何一個技術可以證明 100%沒有漏洞，無法突破，因此，如果在所有的交易過程中都能夠存證（data attestation），日後爭議發生時，才得以稽核（auditing）並歸責（accountibility），才是資訊安全管理的完整架構，並解決前述問題。

第一節 研究結果與討論

由前面的研究成果，筆者得到了定位摩克樹 tp-Merkle tree 有以下三點特性：

一、存證空間小，每一筆資料只要 64bytes 存證。

二、更新與驗證速度快。

三、可以驗證交易或資訊的存在（proof of existence），與驗證不存在（proof of inexistence），或是否遭到竄改（tampered）。

如果我們有一百萬筆交易資訊要存證，2^{20} 等於 1,048,576，換句話說，以定位摩克樹存證，樹高為 21，才會有 2^{20} 個葉節點可以存放一百萬筆交易資訊的存證指紋，定位摩克樹的總節點數則是 2^{21}-1 個節點。每一個資料是以一個 key-value pair 存證，一個 key-value pair 佔 64bytes，每個定位摩克樹的節點需 32bytes，因

此總存證的空間會是：64bytes x 100 萬 +（2^{21}-1）x 32bytes ≈ 125MB。若我們假定一筆交易資訊大約是 1KB，1KB x 100 萬 ≈977MB，因此我們可以說，清算（clearance）一百萬筆交易資訊的存證空間僅需 125MB，因此存證的 overhead ≈ （125 ÷ 977）x 100% ≈ 12.8%。

由第三章第三節 3.3.4 得知，當定位摩克樹的樹高超過 10 以上，其效能幾乎成水平狀態，但是由研究之實驗結果得知，每個葉節點平均存放 key-value pairs 在 10 個以下，其稽核效能最好。2^{17} =131,072，因此改用樹高為 18 的定位摩克樹來清算 100 萬筆交易的存證資料，每個葉節點平均存放的 key-value pairs 數則不會超過 10 個。不過一個 key-value pair 仍然是佔 64bytes 不會變，因此總存證的空間會變成：64bytes x 100 萬 +(2^{18}-1)x 32bytes ≈ 69MB，因此存證的 overhead ≈ （69 ÷ 977）x 100% ≈ 7%，所以 overhead 只有 7%而已。

當一百萬筆交易資訊利用定位摩克樹予以清算存證，則僅需將 Merkle root 上傳區塊鏈，因此僅需一筆礦工費，其中上傳的 Merkle root，會包含 clearance order 與 timestamp，形成一個 key-value pair，稱為 clearance record，存放上智能合約，方便稽核時查詢存證指紋之用。筆者所使用的存證資料結構如下：

$$< \quad R_i，ID_i，i，Ch_i >$$

R_i 是 Merkle root。

ID_i 是索引值，用於搜尋，通常是指 clearance 時的 timestamp。

i 是指 clearance order，由智能合約產生。

CH_i 是 chain-hashing，指到前一個 clearance record，由智能合約產生。

存放在公有區塊鏈上智能合約存證的 clearance record，若經過 chain-hashing 的串接，則稽核驗證程式只要在公有區塊鏈上得到最末一個 clearance record，就可以驗證前面的 clearance record 記錄是否有誤了。

在第一章第三節的 1.3.3 所提到的公有區塊鏈存在著四大問題：

一、如何增加區塊鏈的交易頻寬，及交易速率？

二、如何讓大量的交易資訊得以儲存在區塊鏈上，避免『區塊鏈膨脹』？

三、除了加密貨幣以外的其他交易資訊，儲存在區塊鏈上能否有隱私保護？

四、區塊鏈如何能有效應用在真實世界的各種作業或交易場景？

以上述利用定位摩克樹可以輕易的清算一百萬筆的交易，這一百萬筆交易的清算帳冊僅需一次的上鏈與一次的礦工費，而且只需做存放所謂「clearance record」的動作，因此解決了第一與第二個問題。由於這一百萬筆交易屬於鏈下交易，可以運用各種加密演算法保護隱私性，只有交易指紋的清算帳本會上鏈儲存，交易內容不會公開在公有區塊鏈上，因此第三個問題也解決了。

本論文做了兩項研究，一是雲端虛擬機器的執行環境的完整性即時稽核，若用於其他雲端服務，在於租用者與雲端服務提供商之間的交易問題，可供雲端服務提供商確保服務的可用性與安全性，當爭議發生時如何歸責與賠償？顯然第一項研究也可以應用在此類型的情境。

第二項是類似有服務代理商的交易，產品擁有者如何確保代理商回報銷售數

字是正確的？得以正確的收取費用？顯然此研究也解決此類問題，類似的情境有：Youtube 影片的點閱率、商業網站頁面的點閱率、廣告的效益率…等等，如圖 23，也可以用類似的架構解決。

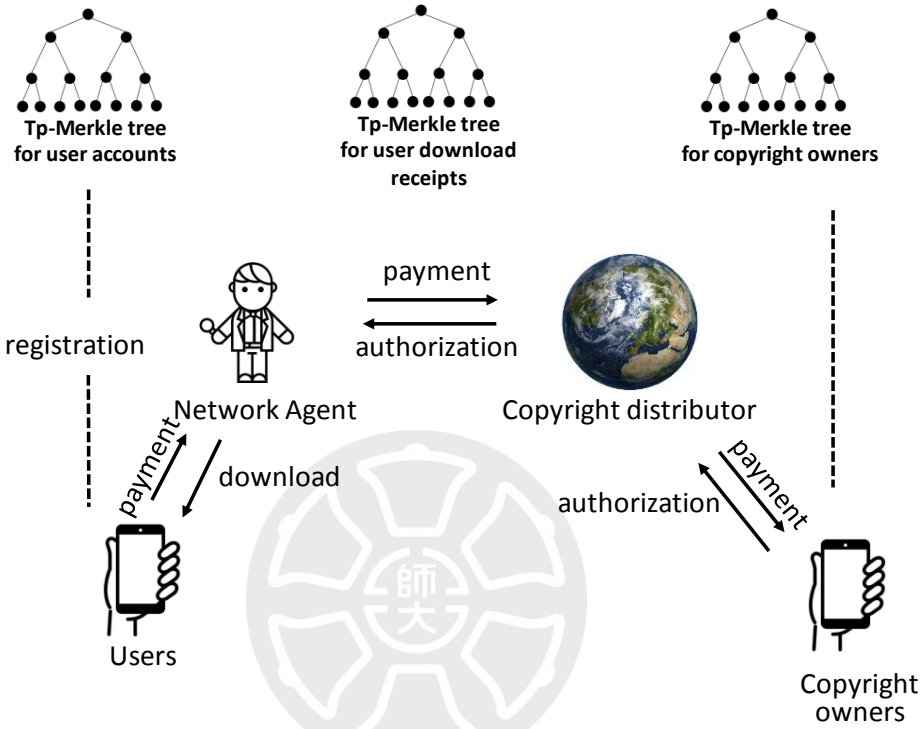


圖 23：數位著作權的多重代理銷售架構。

因此本論文的研究，主要貢獻有：利用定位摩克樹，解決了公有區塊鏈的擴容性與隱私性的問題外，並運用 Web 3.0 去中心化的平台：公有區塊鏈，解決雲端服務平台的完整性問題、銷售與支付費用問題，其中也包括微支付，且當爭議發生時，得以究責並自動賠償，由此並進一步希望將適用的情境一般化（generalization），且無須任何的所謂「公正第三方」的參與介入，提供一個可多方信賴的雲端作業環境。

第二節 未來探討

就目前公有區塊鏈所遭遇的困境，有許研究者提出了 layer one 與 layer two 的解決方案，如第一章第三節的 1.3.4 所述，而定位摩克樹是否適合用於側鏈技術（sidechain）的應用，或改良 sidetree protocol，值得後續研究者的繼續探討。

也因為定位摩克樹可以有效的驗證交易資訊的存在與驗證不存在，並可一次清算大量的交易資訊並存證與提供即時或日後的稽核，因此在元宇宙的生態系中（ecosystems），是否能符合其他的虛擬應用情境，也是值得後續研究者的繼續探討。

此外，負責「定位摩克樹」的中介者（mediator），該身分我們會稱為是「安全協定作業員」（security protocol operator，簡稱 SPO），例如：代理商、分銷商，需負責資料的存證與上傳公有區塊鏈，若該 SPO 同時是公有區塊鏈的礦工，則可避免單點故障（single point of failure，簡稱 SPoF）的問題；但是多數的情境下並不會是公有區塊鏈的節點，則如何避免定位摩克樹存證與稽核服務的 SPoF 的問題，值得後續研究者探討，例如使用微服務（microservices）[134]或高併發系統（high concurrency）[135 - 137]的開發架構。

參考文獻

[1] Gralla, P. (1998). *How the Internet Works* (4th ed.). Indianapolis, USA: QUE.

[2] Zeng, W., Zhao, Y., Ou, K., & Song, W. (2009). Research on cloud storage architecture and key technologies. *The 2th Conference on Interaction Sciences: Information Technologies, Culture and Human*. November 24-26, Seoul, Korea.

[3] Leon, A. （2008）. *Enterprise Resource Planning* （2nd ed.） . New Delhi, India: Tata McGraw-Hill.

[4] Nemana, S. （2017）. *ERP Implementation* （1st ed. eBook）. Emirate of Dubai, United Arab Emirates: Shyanmala Nemana.

[5] 管郁君、黃敏祐（1998）。企業特質與網際網路應用狀況之關聯。管理學報，17（1），119-147。

[6] Yuan, R., & Strayer, W. T. (2001). *Virtual private networks: Technologies and solutions*. Boston, USA: Addison-Wesley Professional.

[7] Osterwalder, A. （2004）. *The business model ontology： A proposition in a design science approach.* PhD Thesis, University of Lauanne, Lausanne, Switzerland.

[8] Chaffey, D. (2011). *E-Business and E-Commerce Management: Strategy, Implementation and Practice* (5th ed.). England: Pearson.

[9] Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing.* Retrieved from nvlpubs.nist.gov

[10] Amazon EC2, available from https://aws.amazon.com/ec2/

[11] Flavián, C., Ibáñez-Sánchez, S., & Orús, C. (2019). The impact of virtual, augmented and mixed reality technologies on the customer experience. *Journal of Business Research, 100, 547-560.*

[12] Gritti, C. O¨ nen, M., Molva, R., Susilo, W., & Plantard, T. (2016). Device identification and personal data attestation in networks. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 9(4), 1-26*.

[13] Berger, S., Goldman, K., Pendarakis, D., Safford, D., Valdez, E., & Zohar, M. (2015). Scalable attestation: A step toward secure and trusted clouds. *The 2015 IEEE International Conference on Cloud Engineering.*

[14] Garfinkel, S. L. (2010). Digital forensics research: The next 10 years. *Digital Investigation, 7, 64-73.*

[15] Carew, H., & Damodaran, M. (2008). X-windows, GUI programming, and Microsoft windows. *Issues in Information Systems, 9(2), 551-559.*

[16] Aghaei, S., Nematbakhsh, M. A., & Farsani, H. K. (2012). Evolution of the World Wide Web: from Web 1.0 to Web 4.0. *International Journal of Web & Semantic Technology (IJWesT), 3(1), 1-10*.

[17] Gant, J. P., & Gant, D. B. (2002, January). Web portal functionality and state government e-service. *Proceedings of the 35th Hawaii International Conference on System Sciences*.

[18] ICQ, available from https://icq.com/

[19] SixDegrees.com , available from http://sixdegrees.com/

[20] Bitcoin, available from https://bitcoin.org/

[21] Guest editors' introduction: Foundation of peer-to-peer computing. *Computer Communications, 31(2), 187–189*.

[22] Diffie, W., & Hellman, M. E. (1976). New drections in cryptography. *IEEE Transactions on Information Theory, IT-22(6), 644-654*.

[23] J., David, J., Davies, & Irvine, M. （1999）. *Cyber space: Virtual Reality and World Wide Web.* New York: Crabtree Pub. Co.

[24] McLuhan, M. (1962). *The Gutenberg Galaxy: The Making of Typographic Man*. Toronto, USA: University of Toronto Press.

[25] Perez, D., Werner, S. M., Xu, J., & Livshits, B. (2021). Liquidations: DeFi on a knife-edge. *The 25th International Conference on Financial Cryptography and Data Security, March 1-5*.

[26] Wang, Q., Li, R., Wang, Q., & Chen, S. (2021). *Non-Fungible Token (NFT): Overview, evaluation, opportunities and challenges.* Retrieved from https://doi.org/10.48550/arXiv.2105.07447 October 25, 2021

[27] *Amazon S3 Service Level Agreement.* Retrieved from aws.amazon.com

[28] *Windows Azure Pricing and Service Agreement*, Retrieved from http://www.microsoft.com/windowsazure/pricng/

[29] iCloud, available from https://icloud.com/

[30] Dropbox, available from https://www.dropbox.com/home

[31] Google Drive, available from https://drive.google.com/start#home

[32] Kremer, S., Markowitch, O., & Zhou, J. (2002) An intensive survey of fair non-repudiation protocols. *ComputeCommun., 25, 1601–1621*.

[33] Popa, R. A., Lorch, J. R., Molnar, D., Wang, H. J., & Zhuang, L. (2011). Enabling security in cloud storage SLAs with CloudProof. *USENIX Annual Technical Conference.*

[34] Hasan, H. R., & Salah, K. (2018). Blockchain-based proof of delivery of physical assets with single and multiple transporters. *IEEE Access, 6, 46781-46793.* doi:10.1109/access.2018.2866512

[35] Hasan, H. R., & Salah, K. (2018). Proof of delivery of digital assets using blockchain and smart contracts. *IEEE Access, 6, 65439-65448.* doi:10.1109/access.2018.2876971

[36] Perlman, R. (1999). An Overview of PKI trust models. *IEEE Network,*

November/December 1999.

[37] 台灣自然人憑證說明，https://moica.nat.gov.tw/what.html.

[38] Ellison, C., & Schneier, B. （2000）. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal, 16(1), 1 – 7.*

[39] 陳琪（1993）。美國反托辣斯法之簡介。公平交易季刊，1（4），45-67。

[40] Bruce, S. (2016). *Data and goliath: The hidden battles to collect your data and control your world.* New York: W. W. Norton & Company.

[41] Bloom, P. (2019). *Monitored: business and surveillance in a time of big data.* London: Pluto Press.

[42] Sternberg, R. J. (2000). *Pathways to psychology study guide 2nd ed.* Belmont: Wadsworth Publishing.

[43] Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system.* Retrieved from www.bitcoin.org

[44] Swan, M. (2015). *Blockchain: Blueprint for a new economy*. O'Reilly, Sebastopol, CA, USA, 2015.

[45] Merkle, R. (1979). *Secrecy, authentication, and public key systems.* Electrical Engineering, PhD Thesis, Stanford University, Stanford, California.

[46] Stallings, W. (2014). *Cryptography and network security: Principles and practice(6th ed.)*. Harlow, England: Pearson Education Limited.

[47] Bamakan, S. M. H., Motavali, A., & Bondarti, A. B. (2020). A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Systems With Applications, 154:113385*.

[48] Bach, L. M., Mihaljevic, B., & Zagar, M. (2018). Comparative analysis of blockchain consensus algorithms. *The 41st International Convention on*

Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018, Opatija, Croatia, May 21-25, 2018.

[49] A report by the UK Government Chief Scientific Adviser (2016). Distributed Ledger Technology: beyond block chain. *Government Office for Science (UK). January 2016.*

[50] Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). *International journal of information security, 1(1), 36-63.*

[51] Bitcoin Magazine, available from https://bitcoinmagazine.com/

[52] Buterin, V. (2014). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.* Retrieved from ethereum.org

[53] Buterin, V. (2014). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Retrieved from https://bitcoinmagazine.com/business/ethereum-next-generation-cryptocurrency-decentralized-application-platform-1390528211

[54] Ethereum, available from https://etherscan.io/

[55] Nick Szabo (1994). *Smart Contracts*. Retrieved from https://www.fon.hum.uva.nl/

[56] Solidity, 2021. Retrieved from https://docs.soliditylang.org/en/v0.7.5/

[57] Parizi, R. M., Amritraj, A., & Dehghantanha, A. (2018). Smart contract programming languages on blockchains: an empirical evaluation of usability and security. *Lecture Notes in Computer Science, Springer, Cham, Switzerland, pp. 75–91, 2018.*

[58] Litecoin, available from https://chainz.cryptoid.info/ltc/

[59] VisaNet: 65000TPS,資料來源：台灣聯合信用卡中心. Retrieved from: https://www.nccc.com.tw/wps/wcm/connect/zh/home/KnowledgeSharing/PaymentCardKnowledge/organizationIntroduction

[60] Real time Blockchain TPS, see https://www.blockchain.com/explorer

[61] Hafid, A., Hafid, A. S., & Samih, M. (2020). Scaling blockchains: A comprehensive survey. *IEEE Access, 8, 125244 - 125262.*

[62] IOTA, available from https://www.iota.org/

[63] HashGraph, available from https://hedera.com/learning/hedera-hashgraph

[64] Algorand, available from https://algorandtechnologies.com/

[65] Chaganti, R., Boppana, R. V., Ravi, V., Munir, K., Almutairi, M., Rustam, F. Lee, E., & Ashraf, I. (2022). A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges. *IEEE Access, 10,96538 - 96555.*

[66] Wani, S., Imthiyas, M., Almohamedh, H., MAlhamed, K., Almotairi, S., & Gulzar, Y. (2021). Distributed denial of service (DDoS) mitigation using blockchain—A comprehensive insight. *Journal of Symmetry 2021,* https://doi.org/10.3390/sym13020227

[67] Sohan, Md. S. H., Mahmud, Sikder, MA. B., Hossain, F. S., & Hasan, Md. R. (2021). Increasing throughput and reducing storage bloating problem using IPFS and dual-blockchain method. *The 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST 2021), January 5-7, 2021, Dhaka, Bangladesh.*

[68] Benaloh, J., & de Mare, M. (1994). One-way accumulators: a decentralized alternative to digital signatures. *In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, 765: 274–285. Springer, Heidelberg (1994),* https://doi.org/10.1007/3-540-48285-7_24

[69] Maxwell, G. *CoinJoin: bitcoin privacy for the real world.* Retrieved from: http://bitcointalk.org Accessed 10 Apr 2023.

[70] Yu, Y., Wang, Y., Hu, Y., Cheng, S., Tu, Y., Hu, X., Du, P., & Wei, B. (2020, November). Blockchain-based PKI system and its application in

Internet of Things. *Proceedings of the 2020 IEEE 4th Conference on Energy Internet and Energy System Integration （EI2）, Wuhan, China, November 2020*.

[71] Yu, G., Wang, X., Yu, K., Ni, W., Zhang, J. A., & Liu, R. P. (2020). Survey: Sharding in blockchains. *IEEE Access, 8 , 14155 - 14181*.

[72] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Tim´on, J., & Wuille, P. (2014). *Enabling blockchain innovations with pegged sidechains.* Retrieved from: http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains

[73] BlockChain Security Inc., available from https://chainsecurity.asia/.

[74] *BTC-Relay, A bridge between the Bitcoin blockchain & Ethereum smart contacts.* Retrieved from: http://btcrelay.org/ Accessed 10 Apr 2023

[75] *Rootstock Whitepaper.* Retrieved from http://www.the-blockchain.com/docs/Rootstock-WhitePaper-Overview.pdf Accessed 10 Apr 2023.

[76] *Lightning Network: Scalable, Instant Bitcoin/Blockchain Transactions*. Retrieved from: https://lightning. network. Accessed 10 Apr 2023.

[77] *Raiden Network - Fast, cheap, scalable token transfers for Ethereum.* Retrieved from: https://raiden.network/ Accessed 10 Apr 2023.

[78] *Plasma whitepaper*, Retrieved from https://www.plasma.io/ Accessed 10 Apr 2023.

[79] Plasma Cash, available from https://www.learnplasma.org/

[80] Westerkamp, M., & Eberhardt, J. (2020). zkRelay: Facilitating sidechains using zkSNARK-based chain-relays. *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2020) September 7-11,2020.*

[81] *nbsspecialpublication500-19: Audit and Evaluation of Computer Security*.

Retrieved from http://nvlpubs.nist.gov

[82] Recommendation X.800(1991), *Data Communication Networks: Open System Interconnections; Security Structure and Applications*. Retrieved from https://www.itu.int/

[83] Hwang, G.-H., Tien, P.-C., & Tang, Y.-H. (2020). Blockchain-based automatic indemnification mechanism based on proof of violation for cloud storage services. *The 2nd International Conference on Blockchain Technology 2020*.

[84] Hwang, G.-H., Huang, K.-Y., Liao, B.-S., Yuan, Y.-L., & Chen, H.-F. (2019). Real-time auditing of the runtime environment for cloud computing platforms. *Journal of Information Science and Engineering, 35, 323-339.* doi: 10.6688/JISE.201903_35(2).0005.

[85] Hwang, G.-H., Huang, K.-Y., & Li, C.-C. (2022). Automatic reward system based on public blockchains. *The 4th IEEE Eurasia Conference on IoT, Communication and Engineering 2022*.

[86] Smith, J., & Nair, R. (2015). The architecture of virtual machines. *IEEE Computer, 38(5), 32–38*.

[87] VirtualBox, available from https://www.virtualbox.org/

[88] Feng, J., Chen, Y., Summerville, D., Ku, W. S., & Su, Z. (2011). Enhancing cloud storage security against roll-back attacks with a new fair multi-party non-repudiation protocol. *IEEE Consumer Communications and Networking Conference(CCNC)*, pp.521-522.

[89] Shraer, A., Keidar, I., Cachin, C., Michalevsky, Y., idon, A. C & Shaket, D. (2010). Venus: Verification for untrusted cloud storage. *The ACM Cloud Computing Security Workshop* .

[90] Stefanov, E., van Dijk, M., Oprea, A., & Juels, A. (2012, December). Iris: A scalable cloud file system with efficient integrity checks. *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*.

[91] Mishra, U. (2010). Methods of virus detection and their limitations. *SSRN eJournal, August, 2010. Retrieved from:* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1916708

[92] Abera, T., Bahmani, R., Brasser, F., Ibrahim, A., Sadeghi, A.-R., & Schunter, M. (2019). DIAT: Data integrity attestation for resilient collaboration of autonomous systems. *Network and Distributed Systems Security (NDSS) Symposium 2019, February 24-27 2019, San Diego, CA, USA.* https://dx.doi.org/10.14722/ndss.2019.23420

[93] Kamara, S., & Lauter, K. (2010). Cryptographic cloud storage. *Financial Cryptography Workshops, pp. 136-149.*

[94] Merkle, R. C. （1988） . A digital signature based on a conventional encryption function. *Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology,* pp. 369-378.

[95] Hwang, G.-H., Peng, J.-Z., & Huang, W.-S. （2013） . A mutual nonrepudiation protocol for cloud storage with interchangeable accesses of a single aAccount from multiple devices. *The 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications （IEEE TrustCom 2013）, Melbourne, Australia,* 16-18.

[96] Hwang, G.-H., Huang, W.-S., & Peng, J.-Z. (2014). Real-time proof of violation for cloud storage. *Cloud Computing Technology and Science(CloudCom 2014), 2014 IEEE 6th International Conference.*

[97] Hwang, G.-H., & Chen, H.-F. (2016). Efficient real-time auditing and proof of violation for cloud sorage systems. *IEEE 9th International Conference on Cloud Computing.*

[98] Hwang, G.-H., Chen, P.-H., Lu, C.-H., Chiu, C., Lin, H. C., & Jheng, A. J. (2018). InfiniteChain: a multi-chain architecture with distributed auditing of sidechains for public blockchains. *Proceedings of the International Conference on Blockchain, pp. 47–60, Seattle, WA, USA, June 2018.*

[99] Hwang, G.-H., & Yeh, S.-Y. (2016). Proof of violation for availability in

cloud computing. *The 15th IEEE/ACIS International Conference on Computer and Information Science (IEEE/ACIS ICIS 2016), June 26-29, 2016, Okayama, Japan.*

[100] Hwang, G.-H., Chang, T.-K., & Chiang, H.-W. (2021). A semidecentralized PKI system based on public blockchains with automatic indemnification mechanism. *Security and Communication Networks, 14, Article ID 7400466, 15 pages* https://doi.org/10.1155/2021/7400466 ,SCI.

[101] Kfoury, E. F., Khoury, D., AlSabeh, A., Gomez, J., Crichigno, J., & Bou-Harb, E. (2020). A blockchain-based method for decentralizing the ACME protocol to enhance trust in PKI. *Proceedings of the 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), pp. 461–465,Milan, Italy, July 2020.*

[102] IPFS, available from https://ipfs.io/

[103] Benet, J. (2014). IPFS-content addressed, versioned, P2P file system. Retrieved from https://arxiv.org/abs/1407.3561

[104] Dunn, A. M., Hofmann, O. S., Waters, B., & Witchel, E. (2011). Cloaking Malware with the Trusted Platform Module. *Proceedings of the 20th USENIX conference on Security, Pages 26-26, San Francisco, CA, 2011*.

[105] Editors: Chow, K.-P., & Shenoi, S. (2010). Advances in digital forensics VI. *Proceedings of the International Conference on Digital Forensics, Hong Kong, China, January 4-6 2010, Revised Selected Papers.*

[106] Trusted Computing Group. *TPM main pecification.* Retrieved from https://www.trustedcomputing group.org/tpm-main-specification/

[107] Yu, F., Zhang, H., Zhao, B., Wang, J., Zhang, L., Yan, F., & Chen, Z. (2016). A formal analysis of Trusted Platform Module 2.0 hash-based message authentication code authorization under digital rights management scenario. *Security and Communication Networks, 9, 2802-2815.*

[108] Sule, M.-J., Li, M., Taylor, G. A., & Furber, S. (2015). Deploying trusted

cloud computing for data intensive power system applications. *in Proceedings of the 50th Interna- tional Universities Power Engineering Conference, 2015, pp. 1-5.*

[109] Berger, S., Goldman, K., Pendarakis, D., Safford, D., Valdez, E., & Zohar, M. (2015). Scalable attestation: A step toward secure and trusted clouds. *in Proceedings of IEEE In- ternational Conference on Cloud Engineering, 2015, pp. 185-194.*

[110] Advanced Intrusion Detection Environment (AIDE), available from http://aide.sourceforge.net/.

[111] Yue, X., Xiao, L., Zhan, W., Xu, Z., Ruan, L., & Liu, R. (2016). An optimized approach to protect virtual machine image integrity in cloud computing. *in Proceedings of the 7th International Conference on Cloud Computing and Big Data, 2016, pp. 75-80.*

[112] Wang, C., Liu, C., Liu, B., & Dong, Y. (2014). DIV: Dynamic integrity validation framework for detecting compromises on virtual machine based cloud services in real time. *China Communications, 11, 15-27.*

[113] Kaczmarek, J., & Wrobel, M. R. (2014). Operating system security by integrity checking and recovery using write-protected storage. *IET Information Security, 8, 122-131.*

[114] Altwaijry, H., & Algarny, S. (2012). Bayesian based intrusion detection system. *Journal of King Saud University Computer and Information Sciences, 24, 1-6.*

[115] Wee, Y. Y., Cheah, W. P., Tan, S. C., & Wee, K. (2011). Causal discovery and reasoning for intrusion detection using bayesian network. *International Journal of Machine Learning and Computing,1, 185-192.*

[116] Xiao, L., Chen, Y., & Chang, C. K. (2014). Bayesian model averaging of bayesian network classifiers for intrusion detection. *in Proceedings of IEEE 38th Annual International Computers, Software and Applications Conference Workshops, 2014, pp. 128- 133.*

[117] Hu, W., Gao, J., Wang, Y., Wu, O., & Maybank, S. (2014). Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics,44: 66-82.*

[118] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016). Malware detection with deep neural network using process behavior. *in Proceedings of IEEE 40th Annual Computer Software and Applications Conference, 2016, pp. 577-582.*

[119] Mira, F., Huang, W., & Brown, A. (2016). Novel malware detection methods by using LCS and LCSS. *in Proceedings of the 22nd International Conference on Auto- mation and Computing, 2016, pp. 1-6.*

[120] Garfinkel, T., & Rosenblum, M. (2003). A virtual machine introspection based architecture for intrusion detection. *in Proceedings of Internet Society Symposium on Network and Distributed System Security, 2003, pp. 1-16.*

[121] Ibrahim, A. S., Hamlyn-Harris, J., Grundy, J., & Al, M. (2011). CloudSec: A security moni- toring appliance for virtual machines in the IaaS cloud model. *in Proceedings of IEEE 5th International Conference on Network and System Security, 2011, pp. 113-120.*

[122] Hizver, J., & Chiueh, T.-C. (2014). Real-time deep virtual machine introspection and its applications. *ACM SIGPLAN Notices, 49: 3-14.*

[123] Tal, G., Pfaff, B., Chow, J., Rosenblum, M., & Boneh, D. (2003). Terra: A virtual machine- based platform for trusted computing. *ACM SIGOPS Operating Systems Review, 37,193-206.*

[124] Wei, J., Zhang, X., Ammons, G., Bala, V., & Ning, P. (2009). Managing security of virtual machine images in a cloud environment. *in Proceedings of ACM Workshop on Cloud Computing Security, 2009, pp. 91-96.*

[125] Haeberlen, A., Aditya, P., Rodrigues, R., & Druschel, P. (2010). Accountable virtual machines. *in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, 2010, pp. 119-134.*

[126] Santos, N., & Lopes, N. P. (2014). Leveraging trusted computing and model checking to build dependable virtual machines. *in Proceedings of the 10th Workshop on Hot Topics in System Dependability, 2014, pp. 1-6.*

[127] Win, T. Y., Tianfield, H., & Mair, Q. (2014). Virtualization security combining mandatory access control and virtual machine introspection. *in Proceedings of IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014, pp. 1004-1009.*

[128] Viswanathan, N., & Mishra, A. (2016). Dynamic monitoring of website content and alerting defacement using trusted platform module. *in N. Shetty, N. Prasad, N. Nalini, ed., Emerging Research in Computing, Information, Communication and Applications, Springer, Singapore, 2016, pp. 117-126.*

[129] Ropsten Testnet, 2021, https://ropsten.etherscan.io/.

[130] Leiba, O., Yitzchak, Y., Bitton, R., Nadler, A., & Shabtai, A. (2018). Incentivized delivery network of IoT software updates based on trustless proof-of-distribution. *Paper presented at the 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).*

[131] Liu , F., Feng, Z., & Qi, J. (2022). A blockchain-based digital asset platform with multi-party certification. *Applied Science, 12(11): 5342.* doi: https://doi.org/10.3390/app12115342.

[132] Stephenson, N. T. (1992). *Snow crash.* New York, USA: Bantam Books, June 1992.

[133] Second Life, available from https://secondlife.com/

[134] Larrucea, X., Santamaria, I., Colomo-Palacios, R., & Ebert, C. (2018). Microservices. *IEEE Software, 35(3), 96–100.* https://doi.org/10.1109/MS.2018.2141030.

[135] Crafa, S. (2015). The role of concurrency in an evolutionary view of programming abstractions. *Journal of Logical and Algebraic Methods in*

Programming, 84, 732–741.

[136] Chen, H., Wang, Q., Palanisamy, B., & Xiong, P. (2017). DCM: dynamic concurrency management for scaling n-tier applications in cloud. *IEEE 37th International Conference on Distributed Computing Systems*.

[137] Fan, P., Liu, J., Yin, W., Wang, H., Chen, X., & Sun, H. (2020). 2PC: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform. *Journal of Cloud Computing: Advances, Systems and Applications, 9(40).* https://doi.org/10.1186/s13677-020-00183-w.