國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：　黃冠寰　　博士

以部分雜湊樹達成有效率的雲端儲存系統即時稽核

Instant Auditing of Cloud Storage Access
by Cache Partial Merkle tree

研究生：　　黃偉賢　　撰

中華民國　　102　　年　6　月

國立臺灣師範大學資訊工程研究所

碩士學位論文

以部分雜湊樹達成有效率的雲端儲存系統即時稽核

Instant Auditing of Cloud Storage Access
by Cache Partial Merkle tree

經考試合格特此證明

審查教授：_____

_____

_____

_____

_____

_____

指導教授：_____

研究所所長：_____

中華民國　　年　　月

摘　要

在現今，人們使用雲端的服務越來越普及，例如雲端儲存(Cloud Storage)，但是用戶將自己比較重要的資料，放在不可相信的雲端上(untrusted server)會擔心所讀到的檔案是不正確或者不是最新的，或是遭受任何攻擊(例如 roll-back attack 或是 replay attack)。因此，很顯然用戶和服務提供商之間存在的許多問題。我們需要一個計劃來解決這些問題。

有一作法是將使用者資料備份，當有爭議時再去比較，但是此做法並不能保證哪一份是最新的以及其正確性，反而是增加更多的資源。在本篇論文中提出了一個機制，將用戶整個資料夾以及檔案以 hash tree 方式儲存，稱為 Merkle tree，以及利用 hash tree 產生之 root hash 來確保整個架構的唯一性，而雲端服務商保存著每次交換訊息所保留的證據，並交換 root hash，確保雙方狀態是一致的，用戶讀到的檔案也為正確以及最新。

然而以上狀況在單一用戶是可行的，但是當有用戶有其他設備使用時，目前資料夾的狀態就必須更新至其他設備，以確保在做更新時，其他設備也是最新的。一般做法是對於還未更新的設備，將證據傳給他們(broadcasting)，但是此舉會增加其他設備不停訪問而增加多餘的時間，並且也會產生許多問題。

我們解決方法不需要使用 broadcasting，而是有一個同步伺服器的機制，讓其他設備先暫時無法向 server 溝通，必須等待設備 A 與 server 完成一整個完整的

運作，才可以解開同步伺服器，避免造成 broadcasting 不完備的情形。雖然同步伺服器的機制會增加其他設備等待的時間，但是藉此讓許多設備透過同步伺服器之中交換證據，等到要做操作時才會更新設備所儲存之 Merkle tree，可以降低了許多時間，以達到真正的即時稽核(true instant audit)，也就是說，當用戶以及使用者有出錯時，馬上就可以偵測到。並再改善用戶驗證時需儲存的 Merkle tree，每個設備只需存部份 Merkle tree(pMT)，若存取時沒有另一部分的 Merkle tree，才需要更新，這樣一樣可以達到 true instant audit，減少用戶儲存負擔。相關的實驗數據結果可以呈現各種交換程序所需要的時間，來證明這個機制的可行性以及優點。

關鍵字：雲端儲存、雲端安全、雜湊樹、即時稽核

ABSTRACT

Nowadays cloud service is becoming more and more popular. One of the most important applications is the cloud storage. However, storing important data in cloud storage may suffer serious security risks. For example, the service provider can launch roll-back attack which is to restore lost files using a backup of an early version of them and their associated digital signatures. Then, the service provider can deny that the user's latest version of files have been lost. Therefore, we need a scheme to have the client device be able to audit if a file obtained from the service provider is valid.

In this paper, we first show that the intuitive solution of instant auditing by applying Merkle tree is inappropriate. Then, we propose an instant auditing communication protocol that can guarantee mutual nonrepudiation between the service provider and user and each client device only has to keep a partial Merkle tree of its account and its last attestation. All the client devices can audit if the obtained file is valid after every file writ operation without requiring broadcast their attestation to all other client devices.

The experimental results demonstrate the feasibility of the proposed scheme. A service provider of cloud storage can use the proposed scheme to provide instant auditing guarantee in their service-level agreement.

Keywords: cloud storage, cloud security, hash tree, Instant auditing

誌　謝

首先感謝我的指導教授黃冠寰，在這兩年的指導之下，深感受益良多，每當研究遇到問題時，教授都會跟我仔細地推敲，找出其中的道理。出國參加研討會時也對我多加照顧，讓我能夠打開眼界，並拿到了最佳論文獎，證明辛苦的努力有了成果。再來要感謝口試委員婁德權教授以及孫宏民教授，謝謝兩位教授給予我的碩士論文許多改進的意見，讓這個研究可以更加完整。我也感謝我的父母，讓我自己決定許多事情，並培養我負責任的態度，也全力支持我取得碩士學位，使我能在讀書當下，不需煩惱學校以外的事情。另外，我要感謝這兩年遇到的學長和同學，謝謝哲生學長以及宇程學長，總是從業界帶回很多新的觀念以及知識，並不吝與我們分享跟討論。謝謝李祺學長，在我有疑問的時候會細心給我許多建議以及方法，在參與課程的製作上也學到了很多。謝謝恆毅學長及柏宏學長，在剛進實驗室的時候不管課業以及研究上我有很多不懂的地方，你們都很願意教我，讓我可以快速地適應環境。最後要感謝我的同學們，翊展、震宗與思鋒，因為你們的陪伴，我的碩士生涯過得很精采，真懷念我們一起寫作業，一起討論，一起奮鬥，一起歡笑，一起到處闖蕩的時光。也感謝我的學弟裕偉，為大家帶來許多歡樂，我感謝這些日子我身邊的所有朋友，因為你們的支持，我今天才能有這樣的成就，才可以順利地度過這兩年的研究時光。

黃偉賢 誌於

國立臺灣師範大學資訊工程研究所

民國一百零二年七月

目錄

附表目錄

附圖目錄

第一章 簡介

1.1 雲端儲存

雲端儲存是一種將資料儲存在所承租的服務提供商並在雲中的虛擬化的模型。服務供應商所經營的大型數據中心，用戶將他們的數據放在這些中心，並可透過跟服務提供商購買或租用儲存容量。雲端儲存服務可通過訪問 Web 服務 API，或基於 Web 的用戶界面。熱門的雲端儲存系統，包括 Google Drive[1]，Dropbox[2]，SugarSync[3]，SkyDrive[4]，box[5]。

而將重要數據保存在雲端將會有嚴重的安全隱患。在雲儲存之中，用戶是不能相信雲儲存服務提供商，且並不知道從雲儲存中所檢索的文件是最新的。服務供應商可以洩露機密數據，修改數據，或者將不一致的數據回傳給用戶。此外，還需擔心惡意安全攻擊(roll-back attack[8] 或是 replay attack[9])。服務提供商是有可能恢復文件的早期版本或使用備份的文件與簽名，然後將舊版本的資料傳給用戶，若用戶沒有防範，這些攻擊將無法避免。而外部的敵人可以非法訪問雲儲存供應商的服務器，服務供應商的員工也可能從內部攻擊。即使雲儲存提供商實現強大的安全性措施，這些安全漏洞也是可能發生的，因此我們需要一個計劃，來解決這些問題。

在本篇論文中，將介紹我們的機制來達到真正的即時稽核(true instant audit)，也就是說，當用戶讀取到的檔案使用者有出錯時，馬上就可以偵測到。並且不僅

是在單一的設備，或用戶有多個設備也可以達到目的。並減少用戶儲存負擔。相關的實驗數據結果可以呈現各種交換程序所需要的時間，來證明這個機制的可行性以及優點。

1.2 稽核

用戶在雲端存取檔案，會擔心檔案是不是最新以及是否正確，因此為了確保檔案的完整性以及最新狀態，我們需要一個計劃，讓服務商證明無罪，或是用戶有其證據去證明對方有罪，我們將此驗證行為稱為稽核(auditing)[10]。

而稽核可分為時期稽核(Epoch based auditing)以及即時稽核(instant auditing)，時期稽核為用戶可以在設定的期間或是雙方產生的證據達到一定容量的時候，再去做稽核與服務商更新最新狀態，這個方法可以減少計算 Merkle tree[11]以及 root hash 的時間，這個部份使用鏈結雜湊(chain hashing)以及 C&L[12]方法可以做到。

但如果用戶有十分重要之檔案，在有與服務商簽合約的情形下，是不能容許出現任何錯誤，更不能還要等到更新時才發現，因此用戶必須要即時知道存取檔案的當下，就知道檔案或是整個資料夾的正確性，因此即時稽核是非常重要的。即時稽核一般的作法是使用 Merkle tree，用戶將整個資料夾(圖 一.1)裡的檔案快取並以 bottom up 方式由最底下的檔案算到最頂端後的根雜湊(root hash)的值，用戶存著整個 Merkle tree(圖 一.2)以及最後做存取所留下的 root hash，在每一次與服務商更動伺服器上的檔案(例如新增、寫入和刪除檔案等等)時，都需更新

Merkle tree 以及 root hash，以確保用戶以及服務商彼此目前狀態是一致的。但是這樣的作法會有一些問題的。第一，服務商無法同時支援許多的用戶或者設備，在現在，一個用戶同時擁有許多設備去存取檔案越來越普及，所以在現今的雲端儲存，就必須要支援多用戶的存取，例如用戶使用 Dropbox 服務，他不僅在個人電腦可以使用服務，而且在智慧型手機上也可以使用該服務，所以這樣會有以下的問題，若是目前最新使用的設備，在與服務商完成存取的動作後，必須要將最後的 root hash 交給其他的設備，否則若放在服務商，其他設備在做更新時並不會知道目前狀態是否為最新，或是會不會被服務商所竄改，所提供的是舊版本檔案的問題。若是直接將最後的 root hash 傳給其他設備，則必須等待全部用戶都上線更新才能確保下一個設備在做存取時是最新的狀態，這是非常浪費時間的。另外也無法支援 non-repudiation，當雙方有糾紛的時候，只憑著 Merkle tree 以及 root hash 是無法保證是誰將檔案搞丟的，用戶與服務商之間並沒有證據可以證明彼此誰是正確的。

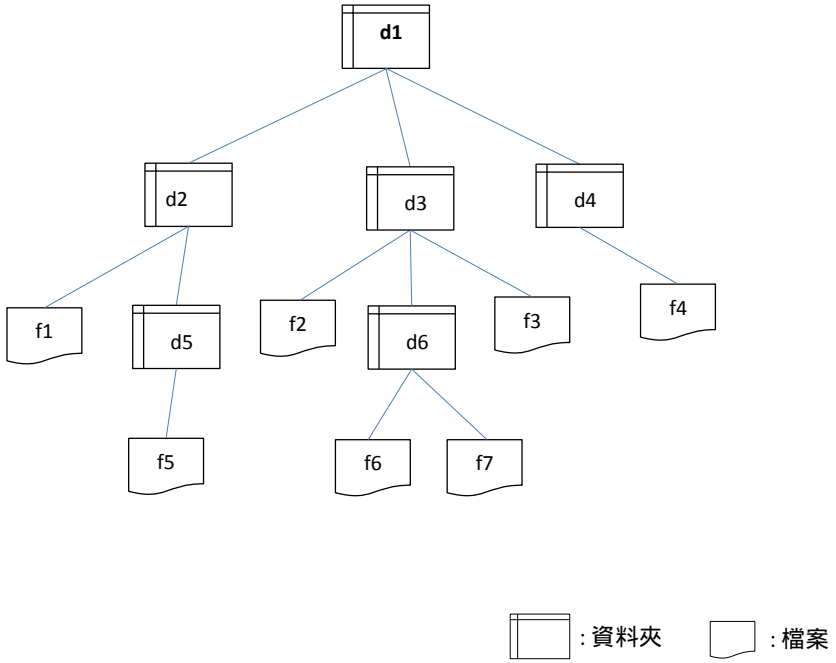


圖 一.1：資料夾

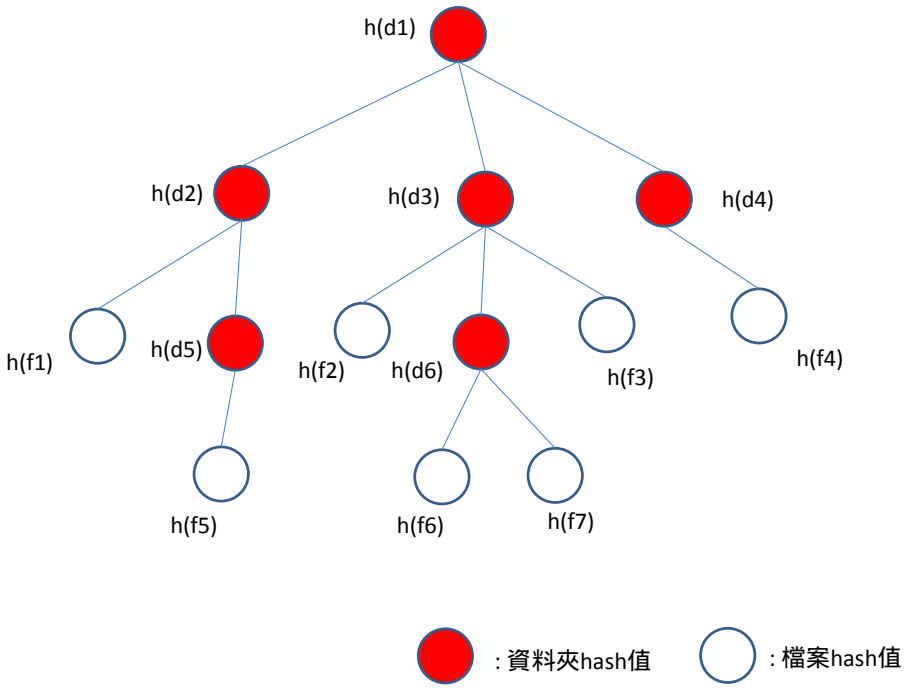


圖 一.2：資料夾的 Merkle tree

1.3 目標

我們的目標是能支援多個用戶或設備存取雲端服務商，在這情況下，讓每個用戶都能確保資料的完整性。並且用部分的 Merkle tree 減輕用戶設備的空間負擔，用同步伺服器解決多個用戶使用 broadcasting 交換證據所會產生的問題，降低需要把 last 證據傳給其他用戶而需等待的時間，並用鏈結雜湊來保證用戶與服務商之間的不可否認性。

1.4 章節介紹

這篇論文首先在第二章我們會開始介紹我們即時稽核的架構，講解我們如何支援多個用戶存取，並使用同步伺服器解決用戶之間需要使用 broadcasting 交換證據的問題。以及更進一步改善 Merkle tree，用戶只需要存部分的 Merkle tree 即可，進而減輕用戶儲存空間負擔，最後討論一些相關的問題。第三章呈現在每個步驟上實驗數據的結果，證明我們在做即時稽核時，每個步驟所需花的時間並不會太大。第四章為相關研究，而第五章為結論以及未來展望。

第二章 即時稽核架構

在現今，一個用戶有許多設備的情況是非常普遍的，所以我們就必須考慮，同時有許多的設備向服務商存取的情況。在這個章節中將會呈現當用戶有許多設備時，都能達到我們的目的。並且達到即時稽核，以及用戶與服務商之間同步交換 Merkle tree 以及 root hash 的機制下並沒有辦法解決用戶與服務商之間的不可否認性問題，因此我們提出一個機制，如圖 二.1 所示。

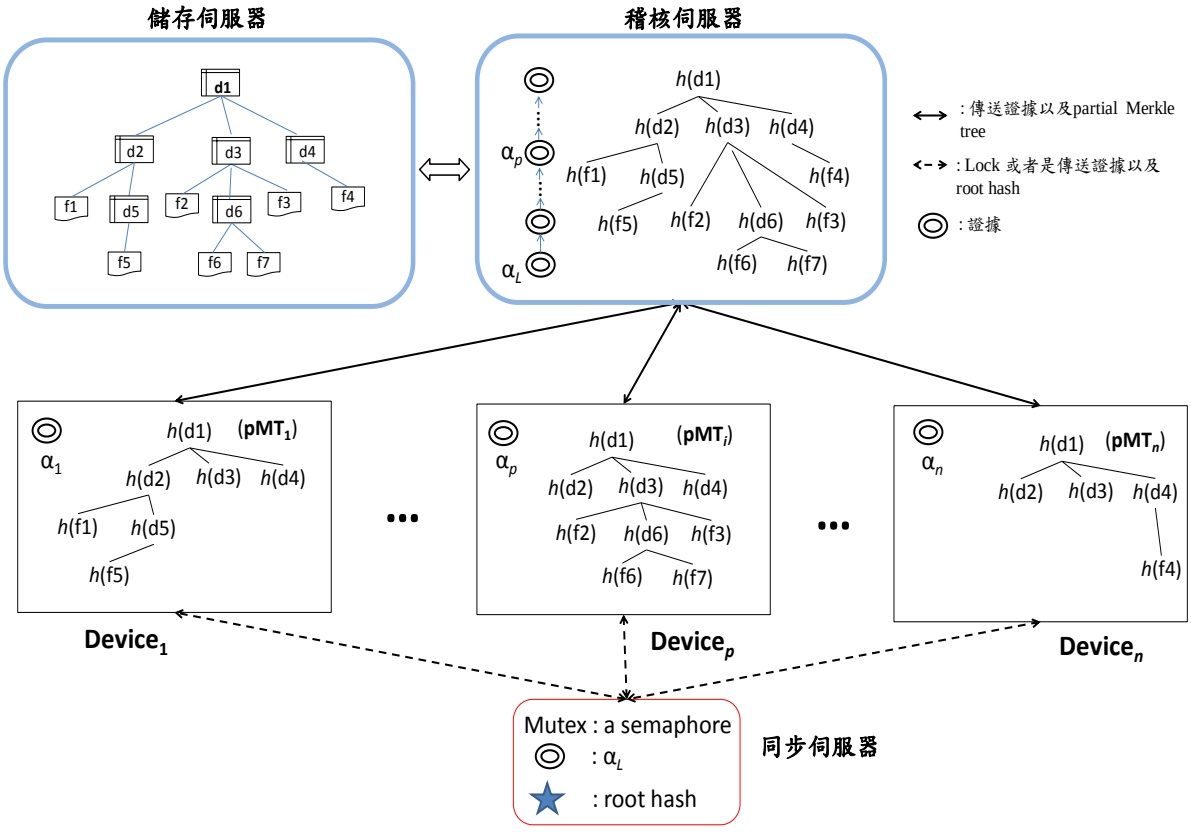


圖 二.1：即時稽核系統架構

首先儲存伺服器(storage server)儲存用戶的資料夾以及檔案；稽核伺服器(Auditing server)作為與多個用戶交換證據以及部份的 Merkle tree，以達到與服務商完成同步的步驟；每個用戶都會儲存最近常用的資料夾所相關的部分的 Merkle tree，以及最後存取所留下的證據，用來與稽核伺服器更新目前最新的狀態。最後同步伺服器(Synchronization server)儲存最後一個用戶存取後的證據以及 root hash，以防止服務商更改檔案或者稽核時需要的資料。

2.1 Partial Merkle tree

接著在這個章節裡，我們將會開始介紹，用戶與服務商之間同步交換 Merkle tree 以及 root hash 的機制。為什麼無法解決不可否認性的問題，以及講解為何要改進 Merkle tree 以及其方法。

首先當一個用戶開啟使用服務後，用戶開始上傳檔案以及資料夾，例如圖一.1，接著將計算每個檔案的 hash 值並記錄下來，如圖 一.2 所示，以 hash tree 的架構，從樹的底層的葉，將檔案以及資料夾一層一層疊上來，例如 *h(d2) = h(h(f3), h(d3),h(f4))*。而最後頂端之 hash 值稱之為 root hash。只要底下某一個節點有所變動，root hash 就會有所變動。接著我們將整個資料夾的架構記錄起來，稱之為 Merkle tree，我們實作的範例可參考附錄裡的圖 A.1，而這樣的結構儲存在用戶端，若是在用戶跟服務商存取檔案過後雲端供應商傳回 root hash，與用戶

所儲存的 Merkle tree 所算出 root hash，若是相同，即可驗證雙方資料夾結構的一致性。然而這樣的作法會有三個問題：第一、無法支援多個用戶，因為在用戶做完檔案操作，並更改 Merkle tree 後，必須要將此 Merkle tree 傳給其他的用戶，以確保下一個用戶在作操作時，手上的 Merkle tree 為最新的狀態，但是此舉是非常耗時間的，因為必須等到其他用戶都上線更新；第二、當檔案數量越多，用戶儲存的 Merkle tree 也會越來越大，不管在任何計算以及儲存上都會造成負擔；第三、就算以上兩點就解決了，但是若是服務商傳回的 root hash 與自己不同，則無法得知用戶與服務商雙方誰對誰錯，因為可能是用戶自己忘記更新檔案，又或者是服務商耍賴更改資料，想要敲詐用戶，所以雙方必須要有一個雙方承認的證據，在有糾紛的時候，可以提出來稽核。我們的詳細解決方法在下一節會說明。

為了減少用戶在儲存 Merkle tree 空間上的負擔，我們提出了 partail Merkle tree(pMT) ，讓用戶只需儲存部分的 Merkle tree。想像一種真實的情況，如圖 一.1 所示，這是一名用戶的資料夾，並且儲存他的 Merkle tree 以做為驗證(圖 一.2)。今天這名用戶可能會在這個月內因為工作只動到 d2 資料夾下面的檔案，假設 d3 下面的檔案以及資料夾數目非常可觀，而使用者也不常用那個資料夾的話，其實在 Merkle tree 中，如圖 二.2 所示，只需紀錄 d3 的 hash 值，而不需將 d3 下的整個結構記錄在 client 端，因為我們在驗證時，若是 d3 有錯，整個 root hash 就會錯，可以再省下使用者的負擔。

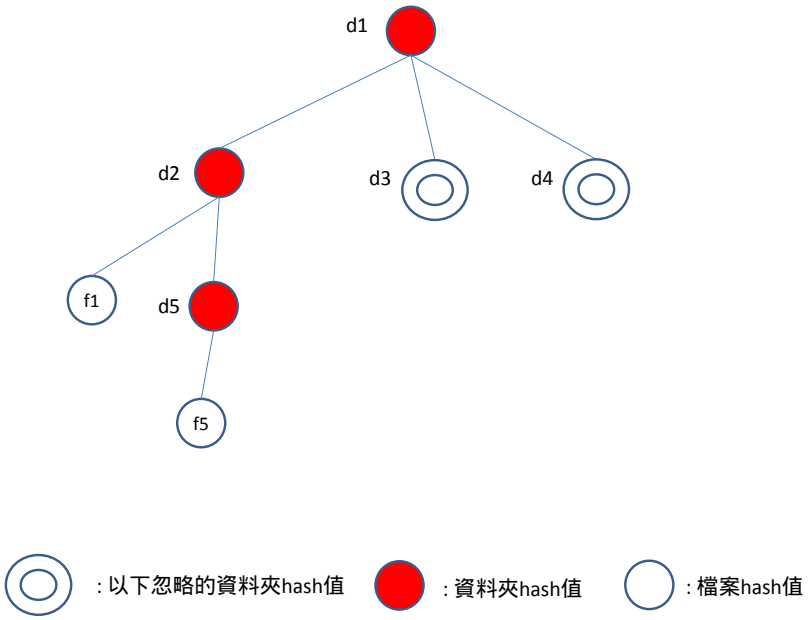


圖 二.2：部分的 Merkle tree

2.2 即時稽核溝通架構

接下來我們會呈現用戶或裝置如何與同步伺服器有效率的即時稽核，並且達到不可否認性。當每次用戶 P 向服務商發出請求，都必須遵循以下的機制步驟：

步驟一： Lock(Mutex)

步驟二：向同步伺服器取得 root hash(RH)以及最後的證據(α_L)，確認目前是否最新狀態

步驟三：若取得的 RH 不同，則需要向稽核伺服器更新最新狀態。假設目前同步伺服器的最後證據為 α_L，而用戶 P 手上的證據為 α_p，則將(α_p, α_L) 送給稽核伺服器。

步驟四：稽核伺服器將用戶未更新的區段，也就是 α_p 到 α_L 之間的證據交給用戶

步驟五：確認 $\alpha_L \rightarrow \alpha_p$ 是否正確的鏈結

步驟六：並取出在這段證據之中，是寫入而且在 pMT_p 之中，即更新 pMT_p ，更新最新狀態後算出 root hash 確認是否與 RH 相同，接著才能做存取的動作。

步驟七：我們假設設備 P 要執行檔案的運作到用戶 U 的帳戶，以下是與稽核伺服器交換證據的步驟：

- 當設備 P 想要存取他帳號裡的檔案，他會送一個請求訊息 (request message) Q_i, 則 $Q_i=(OP_i, [OP_i]_{pri(U)})$ 給稽核伺服器。其中 OP_i 為請求的格式，例如新增檔案、更新檔案內容或者讀取檔案。
- 當服務供應商收到 Q_i 後，要藉由驗證 $[OP_i]_{pri(U)}$ 電子簽章，確認是否為正確的請求，接著就送一個回應訊息(*request message*) R_i，則 $R_i = (Q_i, [Q_i, CH_{i-1}]_{pri(AuditingServer)})$ 給用戶，其中 CH_{i-1} 為第$(i-1)$個鏈結雜湊，且 $CH_{i-1}=[Q_{i-1},CH_{i-2}]_{pri(AuditingServer)}$。
- 當用戶 P 收到了 R_i 之後，用戶藉由 Q_i 和 $CH_{i-1}=[Q_{i-1}, CH_{i-2}]_{pri(AuditingServer)}$ 去驗證其數位簽章是否為正確。如果 R_i 與 R_{i-1} 順序為正確，則用戶 P 回傳(*reply-response message*) RR_i 給服務供

應商，其中 $RR_i = (R_i, [R_i]_{pri(U)})$。

- 稽核伺服器執行 Q_i ，假設執行結果為 L_i ，服務供應商回傳一個確認訊息(*acknowledgement message*) ACK_i 給用戶，其中 $ACK_i=(L_i, RR_i, [L_i, RR_i]_{pri(AuditingServer)})$，則用戶必須保存。

步驟八：如果 是 Q_i 為讀取 f 且 pMT_p 有 f 之雜湊值(hit)：

- 儲存伺服器將 f 傳給用戶
- 由用戶的 pMT_p 找到 f 的雜湊值，確認 Ack_i 中的 hash 值是否相同，完成讀取。
- unlock 同步伺服器

否則 (也就是 f_1 雜湊值不在 pMT_p 之中)：

- 和稽核伺服器下載 $pMT(\{f_\}})$
- 將 $pMT_p = pMT_p \cup pMT(\{f_\}})$
- 向儲存伺服器下載 f_1
- 由用戶的 pMT_p 找到該檔案的雜湊值，確認 Ack_i 中的 hash 值是否相同，完成讀取。
- unlock 同步伺服器

步驟九：如果 Q_i 為寫入 f_1 且 pMT_p 有 f 之雜湊值(hit)：

- 將改寫的檔案傳到伺服器

- 將欲改寫的檔案雜湊過後更新 pMT_p
- 根據 pMT_p 重新計算新的 root hash(RH')
- 更新同步伺服器的 α_L 以及 RH 後 unlock

否則 (也就是 f 雜湊值不在 pMT_p 之中)

- 和稽核伺服器下載 pMT({f})
- 將 $pMT_p = pMT_p \cup pMT(\{f_j\})$
- 將改寫的檔案傳到伺服器
- 將欲改寫的檔案雜湊過後更新 pMT_p
- 根據 pMT_p 重新計算新的 root hash(RH')
- 更新同步伺服器的 α_L 以及 RH 後 unlock

在步驟七之中，我們使用了接下來將加了數位簽章的證據鏈結起來，根據 C&L 機制[12]，如果用戶可以取得最後的證據以及 root hash 的話，在多個裝置操作的機制裡，是可以避免回捲攻擊的。在我們的機制裡，使用 semaphore Mutex 讓同時間只有一個裝置可以開始我們的機制，因為只有一個裝置可以 lock semaphore 去執行我們機制，所以我們可以保證 CIWF。

接下來我們講解每個檔案的操作都是可以成功的完成稽核。在步驟六裡面，因為所有的證據都是正確鏈結的關係，設備 p 每次都能成功的更新他的 pMT_p，

我們有以下幾種狀況：

- 如果檔案操作為讀取檔案 f 且 f 的雜湊值有在 pMT_p 裡面，一定可以保證所讀到的檔案 f 是否為正確。
- 如果檔案操作為讀取檔案 f 且 f 的雜湊值卻沒有在 pMT_p 裡面，設備 P 下載包含 f 以及所有相關更動資料夾，因為在步驟二中，每次都會獲得最新的 root hash，就可以辨別更動後的 pMT_p 是否為正確。接著，就可以根據 pMT_p 來辨別讀到的檔案是否為正確。
- 如果如果檔案操作為寫入檔案 f，只要確保證據是否正確被鏈結，pMT_p 是否更新為最新，最後把證據以及更動後的 root hash 送給同步伺服器儲存。

2.3 即時稽核架構討論

接下來在這一節關於即時稽核架構我們提出了幾個問題來討論。

為什麼這個機制可以保證不可否認性(non-repudiation)，檔案完整性(data integrity)，寫入的循序性(write serializability)，以及讀取的新鮮性(read freshness)？(簡稱 CIWF[8])。在步驟七之中，我們使用加了數位簽章的證據並按照順序鏈結起來，根據 C&L scheme，如果用戶可以取得最後的證據以及 root hash 的話，在多個裝置操作的機制裡，是可以避免回捲攻擊的。在我們的機制裡，使用 semaphore Mutex 讓同時間只有一個裝置可以開始我們的機制，因為只有一個裝

置可以 lock semaphore 去執行我們機制，所以我們可以保證 CIWF。

為什麼要將最後的證據以及 root hash 放在同步伺服器？首先我們必須要假設同步伺服器不可被攻破，如果被侵入或是有人放了錯誤的證據或者是 root hash，是有可能讓稽核伺服器發動回捲攻擊的。如果不將這些資料放在同步伺服器，假如服務商修改稽核伺服器上的資料，將舊的版本的證據或是 root hash 交給用戶，也就是回放攻擊，因為用戶不知道其他用戶更新了哪些部分，所以是不能相信服務商的。因此最後一個做存取的用戶必須將最後做完的證據以及 root hash 放在同步伺服器。

在這個機制之中，如果不使用同步伺服器交換證據的話，那就要使用 Broadcasting 的做法，在其中一個設備要做操作之後必須要將證據傳給其他用戶，其解決方法並不太完備，接著想像一種情況，當設備 A 對 server 請求寫入檔案了以後，完成後尚未將最後的證據傳送出去時，設備 B 就要向 server 做下一個 operation，則設備 B 可能得到錯誤的 file 卻不知道，當這類的事情沒有避免，之後又有更多的 Device 做操作，則其他設備都會得到舊版本的檔案或證據，整個機制將無法保證正確。並且若是透過服務商將證據傳給其他用戶，則會有兩個問題，(1) 無法避免服務商傳回來的證據是否確實為最新的版本，在設備 A 沒有上線的情況之下，是很難求證以及相信服務商的。(2) 若是設備 A 直接傳給其他用戶，則必須等待全部用戶都上線更新完成，才能確保下一個設備在做存取時是最

新的狀態，但是這是非常浪費時間的。

因此，如果能做到在每次其中一個用戶做完操作之後，其證據能馬上且安全的送到其他用戶，避免上述的問題的話，即可不需要同步伺服器幫忙。

第三章 實作與實驗數據

在這個章節我們實作了一系列的實驗來證明此架構的可行性以及詳細執行時間，包含產生 root hash、Merkle tree 的時間和各種步驟所需要花的時間。

首先我們看到我們實驗所使用的三個不同的帳戶，如**表 三.1** 所示分別是(1) 帳戶 A，屬於檔案數目以及容量小的範例，(2) 帳戶 B 為檔案數目眾多，但是每個檔案容量皆不大，(3) 帳戶 C 為實際使用 20 年的一個資料夾，所以有參考價值。

表 三.1：實驗數據中的三個範例帳戶

	帳戶 A	**帳戶 B**	**帳戶 C**
Number of files	44	22968	28404
Number of folders	6	188	1718
Total size	666 MB	32.9 MB	6.9 GB

表 三.2：從未 hash 過的檔案所產生 root hash 所需的時間(單位:秒)

帳戶 A	3.208
帳戶 B	8.893
帳戶 C	120.003

(A) PC

帳戶 A	15.744
帳戶 B	875.211
帳戶 C	978.889

(B) Hadoop system

表 三.3：直接從檔案的雜湊值產生 root hash 所需的時間(單位:秒)

帳戶 A	0.023
帳戶 B	3.874
帳戶 C	32.423

(A) PC

帳戶 A	1.823
帳戶 B	198.572
帳戶 C	347.568

(B) Hadoop system

接下來的實驗我們在兩個平台上面實作：(1) Windows 7 電腦 (2) Hadoop 系統[17]，而檔案都是自動複製以及分散到 Hadoop 系統之中。其中(1) Windows7 電腦規格 CPU 為 3.1GHz Intel Core i5 2400 四核心，6GB 的記憶體，500GB 的硬碟空間。(2) Hadoop cluster 規格為由五台 PC、五個節點組成。每個節點都有 3.0GHz Intel Core 2 Quad processor 的 CPU、2 GB 的記憶體、500 GB 的硬碟空間、Ubuntu 10.04 LTS 的作業系統以及 Java Development Kit 6。

表 三.2 為第一次帳號開啟時，所有的檔案都還沒有算過雜湊值，所以必需要生成所有檔案雜湊值並計算出 root hash 的時間。若是使用者一次上傳許多檔案，服務商更可以先個別計算該檔案的雜湊值，再一起計算 root hash，這樣即可大幅降低計算的時間。Hadoop 系統則因為需要備份檔案到多台電腦，所以時間才會比較慢。

而**表 三.3** 則為之後資料夾裡已經有了各個檔案的雜湊值後，直接從檔案的雜湊質產生 root hash 的時間，可看到計算 root hash 的時間大幅降低，例如在帳戶 C 這麼多個檔案之中，節省了約四分之一的時間。

接著我們測量產生完整 Merkle tree XML 檔案[1] 所需的時間，一樣是在 PC 以及 Hadoop 系統上面實作，時間如**表 三.4** 所示，因為每個檔案的雜湊值已經有產生了，所以可以看到在這麼多檔案的資料夾中產生 Merkle tree 的 XML 檔案格式的時間並不長。

接著**表 三.5** 為產生出完整Merkle tree的容量大小，皆只有MB單位的容量，以現在每個人的手持裝置來說，這樣的負擔其實並不大，更何況我們提出 pMT 再減輕用戶的負擔。

表 三**.6** 是從 Merkle tree 計算出 root hash 的時間，皆在一秒之內完成，這對用戶的負擔也是很少。

表 三**.7** 則是根據帳戶 C 隨著隨機更改檔案的數量，所生成的 partial Merkle tree 的容量大小，由這成長曲線顯示節省的空間在經常更動的檔案不超過 100 個的話是非常可觀的。最後**表 三.8** 則以表格方式呈現**表 三.7**，也就是說如果更動 10 個檔案，則最多會有 10 個 pMT 所組合而成的 pMT。

---

[1] Merkle Tree 實作 XML 格式請參考附錄裡的圖 A.1

表 三.4：產生 Merkle tree XML 檔案的時間(單位:秒)

帳戶 A	0.039
帳戶 B	2.037
帳戶 C	4.414

(A) PC

帳戶 A	0.798
帳戶 B	289.415
帳戶 C	312.549

(B)Hadoop cluster

表 三.5：Merkle tree 產生的容量大小

帳戶 A	3.2 kB
帳戶 B	2.96 MB
帳戶 C	3.4 MB

表 三.6：由 Merkle tree 產生 root hash 的時間(單位：秒)

帳戶 A	0.071
帳戶 B	0.841
帳戶 C	0.985

表 三.7：以帳戶 C 為例根據經常更改的檔案數量，所生成的 Merkle tree 的容量

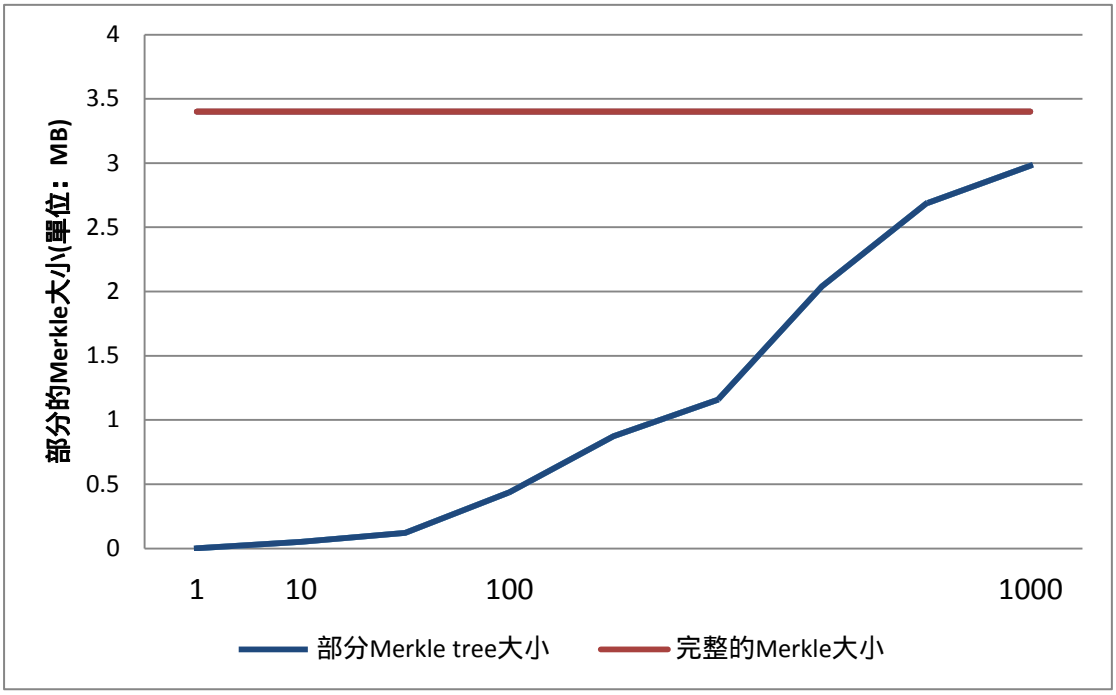


表 三.8：根據表 三.7，以表格方式呈現

隨機選取檔案數量	1	10	100	200	300	400	500	1000
Partial Merkle tree 平均大小 (單位 MB)	0.002	0.05	0.438	0.871	1.398	1.676	2.04	2.98

接下來為”部分 Merkle tree”相關實驗數據，分為四個部分討論，(1)讀取檔案有 hit(2)寫入檔案有 hit (3)沒有 hit 時，稽核伺服器傳過來的 pMT 檔案大小 (4)根據稽核伺服器傳過來的 pMT，更新本地 pMT，並重新計算出 root hash 的時間。

首先第一部分，在讀取檔案過程中，檔案有在本地 Merkle tree 中，則向服務商提出讀取檔案請求，服務商收到後，將檔案傳回給用戶，用戶只要將傳回來的檔案雜湊後的值，比對儲存的 Merkle tree 裡的該檔案是否為相同，即可知道此檔案是否為最新。其實驗數據為**表 三.9** 所示，隨機找尋 1000 次以存成 JAVA 物件的 Merkle tree 裡面所記錄的檔案的雜湊值，並分為最差、平均以及最好的時間呈現，可以看到時間都可在一秒之內即可完成。

表 三.9：讀取檔案如果 hit，則由本地 pMT 找到該檔案的 hash 時間(單位：秒)

worst	average	best
0.05	0.04	0.03

(A) 帳戶 A

worst	average	best
0.61	0.28	0.09

(B) 帳戶 B

worst	average	best
0.75	0.42	0.15

(C) 帳戶 C

第二、討論儲存檔案時，如果檔案有在本地 pMT 中，則向服務商提出儲存請求，服務商將檔案寫入後，重新計算 root hash 並傳回給用戶，用戶也將該檔案的雜湊值更新本地 pMT，並重新計算 root hash，如果與服務商傳回一致，則驗證程序結束。其數據如**表 三.10** 所示，欲寫入的檔案數目為 1 個、10 個、100 個到 1000 個檔案時，分別跑了 100 次數據，最多需花多少時間，最少需花多少時間，並且在這 100 次平均為多少時間。

表 三.10：儲存如果 hit，找到檔案位置後更新 Merkle tree，接著重新計算 root hash 的時間(單位:秒)

更改檔案數量	worst	average	best
1 file	0.058	0.034	0.029
10 files	0.287	0.223	0.187

(A) 帳戶 A

更改檔案數量	worst	average	best
1 file	0.874	0.584	0.358
10 files	5.784	3.158	1.181
100 files	9.184	5.97	3.539
1000 files	31.598	28.514	26.158

(B) 帳戶 B

更改檔案數量	worst	average	best
1 file	0.977	0.758	0.498
10 files	7.954	4.811	1.797
100 files	11.399	7.121	4.872
1000 files	33.587	32.897	31.154

(C) 帳戶 C

第三，討論如果讀取檔案，而此檔案沒有記錄在用戶本地的 pMT 之中，稽核伺服器傳過來的給用戶更新的部份的 Merkle tree 的大小，**表 三.11** 呈現實作的數據，為三個帳戶分別以更改的檔案數目 1 個、10 個、100 個到 1000 個檔案時，分別跑了 100 次數據，最大以及最少會到多少容量，並且在這 100 次平均為多少容量。

表 三.11：讀取如果沒有 hit，則從稽核伺服器傳過來更新的 pMT 大小

更改檔案數量	worst	average	best
1 file	0.64	0.192	0.32
10 file	3.2	1.87	1.1

(A) 用戶 A(單位: KB)

更改檔案數量	worst	average	best
1 file	1.93	0.161	0.064
10 files	2.01	0.87	0.131
100 files	2.12	1.32	0.99
1000 files	2.96	2.57	2.34

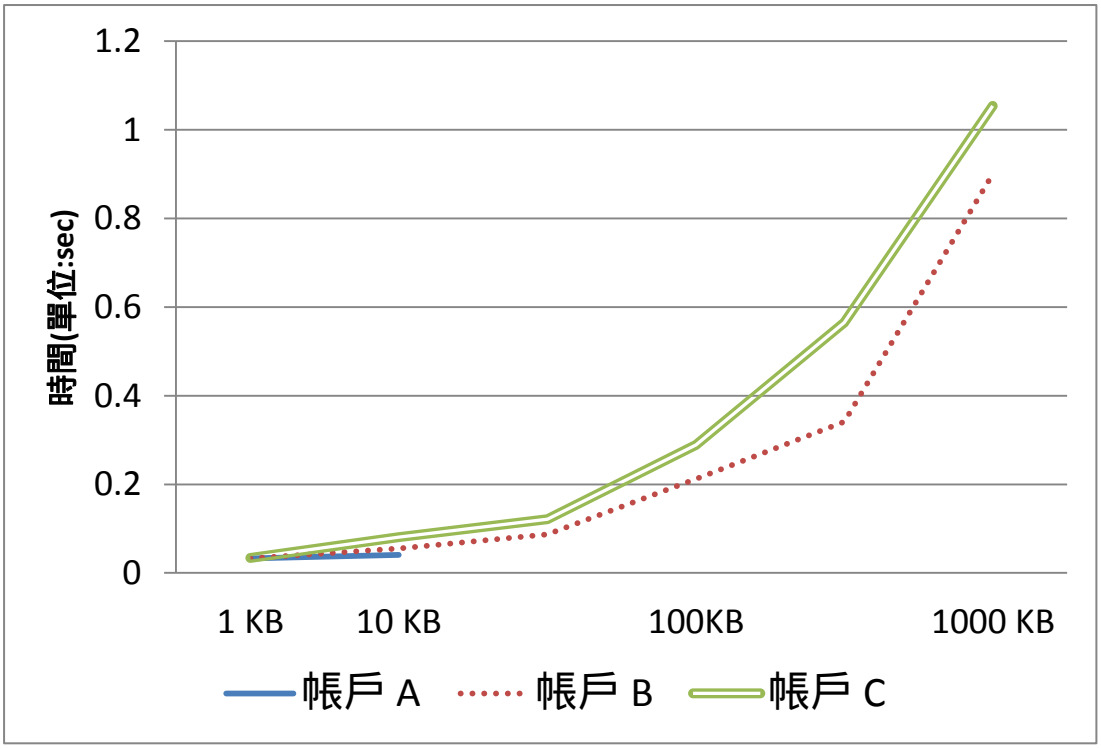
(B) 用戶 B(單位:MB)

更改檔案數量	worst	average	best
1 file	1.38	0.501	0.299
10 files	2.32	0.987	0.482
100 files	2.67	1.98	1.5
1000 files	3.1	2.98	2.78

(C) 用戶 C(單位:MB)

第四、在知道傳過來的部分 Merkle tree 大小後，並更新 pMT 後，就要根據 Merkle tree 重新計算 root hash，而**表 三.12** 即根據 pMT 的大小，重新計算 root hash 的成長曲線的數據。

表 三.12：寫入如果沒有 hit，更新 pMT 後，重新計算 root hash 所需花的時間



接下來我們設計了一系列完整的實驗以及比較。第一、最一般的檔案傳輸，也就是並未加入任何稽核的架構，讓用戶隨機存取帳戶 C1000 次檔案以做為比較。實驗數據結果如**表 三.13**。

表 三.13：以帳戶 C 為例，在沒有任何稽核機制下的檔案傳輸時間(單位：秒)

檔案大小	讀取	寫入
1~100K	0.012	0.016
100K~1MB	0.047	0.049
1MB~5MB	0.133	0.154
5MB~10MB	0.216	0.223

第二、根據我們的即時稽核架構，也就是圖 二.1，實作同步伺服器、用戶以及服務商程式，三者之間使用 socket 網路協定溝通，並用 semaphore 鎖住用戶，讓同一時間只會有一個用戶到同步伺服器拿取證據以及 root hash，並隨機讀取以及寫入檔案，根據架構流程更新狀態以及串起證據，驗證檔案是否正確，最後將 root hash 以及證據放回同步伺服器，並解開 semaphore 後，讓下一個用戶可以繼續操作。

我們分成兩個部分，單一用戶以及三個用戶個別隨機存取帳戶 C 1000 次檔

案並分別根據檔案的大小來列出完成整個流程所需的時間，如果是單一用戶，則不需要同步伺服器做驗證，因為用戶手上握有的永遠都是最後的證據以及 root hash，並跟多個用戶的加上同步伺服器的機制下做比較。其實驗數據分別為**表 三.14** 以及**表 三.15**。我們可以看到在**表 三.14** 單一用戶數據中，在讀取以及 Merkle tree 有 hit 的情況下，在加了即時稽核機制後大約只增加 0.4~0.6 秒的延遲，即使 miss 後需向稽核伺服器下載部分的 Merkle tree 與用戶持有的 Merkle tree 合併，也只再多花約 0.3~0.4 秒合併。在寫入方面，因為需要更改 Merkle tree 以及重新 root hash 的關係，與無任何稽核機制的寫入時間，大約增加了 0.9~1 秒的時間。接著在**表 三.15** 三個用戶的實驗與**表 三.14** 不同為加了同步伺服器取得最後的證據以及 root hash，可以看到與單一用戶的數據也只增加了約 0.3 秒的延遲時間。

表 三.14：即時稽核完整流程時間(單一用戶)

操作	Merkle tree hit or miss	檔案大小	時間
讀取	Hit	0~100K	0.417
		100K~1MB	0.518
		1MB~5MB	0.651
		5MB~10MB	0.712

	miss	0~100K	0.751
		100K~1MB	0.922
		1MB~5MB	1.098
		5MB 以上	1.156
寫入	Hit	0~100K	0.921
		100K~1MB	1.024
		1MB~5MB	1.239
		5MB~10MB	1.367
	Miss	0~100K	1.239
		100K~1MB	1.359
		1MB~5MB	1.601
		5MB 以上	1.721

表 三.15：即時稽核完整流程時間(三個用戶)

操作	Merkle tree hit or miss	檔案大小	時間
讀取	Hit	0~100K	0.791
		100K~1MB	0.883
		1MB~5MB	1.098
		5MB~10MB	1.138
	miss	0~100K	1.023
		100K~1MB	1.268
		1MB~5MB	1.471
		5MB~10MB	1.591
寫入	Hit	0~100K	1.32
		100K~1MB	1.462
		1MB~5MB	1.618
		5MB~10MB	1.76
	Miss	0~100K	1.691
		100K~1MB	1.719

		1MB~5MB	2.067
		5MB~10MB	2.209

接下來展示用戶儲存 partial Merkle tree 的優勢，根據我們即時稽核的步驟，如果用戶是儲存完整的 Merkle tree，那麼在一開始時，用戶必須要下載完整的 Merkle tree 才能開始運作，而使用 Partial Merkle tree 的話，只需要下載部分 Merkle tree，以下為實驗比較，其中用戶 D 約 85200 個檔案，5130 個目錄，以及總共 21GB。相關實驗結果為表 三.16。我們可以看到在 Partial Merkle tree 的時候，檔案資料夾越大，因為樹高並不會明顯變高，所以平均時間並不會增加，但是若使用完整的 Merkle tree，隨著檔案數目越大，用戶下載的 Merkle tree 會越來越大，啟動時間也會越來越長，由這樣的比較可以顯示 Partial Merkle tree 的優勢。

表 三.16：用戶開始檔案操作後的啟動時間(單位：秒)

	Scheme	最少時間	最大時間	平均時間
帳戶 C	Partial Merkle tree	0.034	0.053	0.045
	Complete Merkle tree	0.681	0.752	0.701
帳戶 D	Partial Merkle tree	0.034	0.055	0.046
	Complete Merkle tree	3.428	3.617	3.478

第四章 相關研究

近年來，由於虛擬機器技術以及分散式結構的電腦運算技術成熟，促使了雲端運算的流行，使用者資料不再隨身攜帶，全部都放置在網路世界的另一頭，常見的雲端儲存系統包含了 Google Drive, Dropbox, SugarSync , SkyDrive, 和 Box ，但這些系統都不支援相互的不可否認性服務層級協議，當使用者資料出問題的時候，就只能束手無策。一個有效且簡單解決問題的辦法是將已經儲存的固定資料完整的複製幾份到不同的伺服器上，當伺服器出問題時，可以利用這些備份還原，Plutus[14]是一個加密的儲存系統，即使不依靠伺服器上部屬的資訊安全環境，也能讓使用者安全的分享資料。其所有資料被加密過後儲存，而且加密的金鑰採用分散式的配置管理。使用者可以透過備份的伺服器讀取與寫入資料，當某一台複製的伺服器損毀，使用者可以透過其他備份的伺服器進行修復，但是這個方法的缺點是，他必須在不同伺服器管理多份完整的備分資料。即使在許多伺服器都遭受攻擊而損毀的情況下，利用多台伺服器上的備份也可以確保資料的完整性。此外，有些系統利用這個備份的方法達到資料的安全性，像是[18][19][20][21]。有一些系統考慮雲端儲存伺服器是不可信任的，這些系統不解決資料遺失之後該如何還原，而是著重於偵測系統錯誤，像是偵測資料的完整性及一致性，以及做運算時，是否符合循序寫入以及讀取最新的原則。SiRiUS[13]是一個具安全性考

量的檔案系統,他的設計理念是在不安全的網路環境下利用階層的概念實作系統，檔案儲存在檔案伺服器中包含兩個部分,第一個部分是檔案的元資料(meta data)，第二個部分是加密過的檔案資料，利用雜湊樹保證元資料是否為最新的。客戶端為使用者生成最新的元資料雜湊樹,藉由這個雜湊樹保證使用者在讀取元資料時能讀到最新的，然而這個系統就只能保證元資料的最新鮮性質(freshness)。由於回復性攻擊[9]，攻擊者可能拿舊的檔案資料假裝新的檔案資料，因此並不能保證使用者讀取檔案資料一定可以讀取到最後修改的。Venus[9]這個檔案系統藉由假設運算都能符合規定的完成，消除使用者間額外的通訊負擔，最後在檢驗資料是否滿足一致性。做完運算時，當伺服器回復” optimistically”，這裡稱為紅色的運算代表這個運算滿足了資料的完整性但不保證資料的一致性,假使過了一段時間，並驗證這些紅色的運算是正確的，Venus 的應用程式會將紅色的運算改為綠色的運算,代表運算通過一致性的驗證,透過這樣的概念我們可以瞭解到 Venus 這個檔案系統在最後會達成檔案的一致性,他保證所有的綠色運算都滿足一致性。假如某個紅色運算驗證錯誤，那他絕對不會被標記成綠色的運算，在最後會通知所有系統上的使用者發生錯誤，但使用者必須經常的跟系統管理核心溝通(系統管理核心由部分常駐的使用者們所組成)，且核心內部成員不可以發生錯誤。總括上述幾個系統，他們都只能偵測是否違反資料的一致性及完整性，但不能證明以及釐清導致資料發生錯誤的原因，說服第三方的人員。

接下來為可達到即時稽核的相關研究：SUNDR[23]是一個網路檔案系統，設計的核心是假設伺服器端是不可信任的，使用者的資料可會被竄改或是損毀。SUNDR 讓使用者偵測所有伺服器端未經過授權的檔案更動，SUNDR 的協定當達成循序的寫入以及讀取時一定是最新的資料兩種性質時，稱為滿足 fork consistency，並且能保證客戶端能偵測檔案的完整性或一致性是否正確。客戶端必須要在檔案系統上維護一份檔案系統的快照清單，當使用者想要對檔案系統上的資料做更動，使用者必須要下載最新的檔案系統的快照清單，然後才開始做運算。這個快照清單可以讓使用者偵測檔案系統是否有異常，檔案系統若是沒有異常，當運算結束後，使用者必須產生一個最新的檔案系統快照，並且加入至檔案系統快照清單中，持續維持系統的完整性及一致性。SUNDR 處理違規檢測的問題，並利用”forking”的語意，在[24][25][26]中被採用。這些解決的方案保證資料的完整性並利用使用者間額外的通訊頻寬來實現相關的資料一致性概念。Iris 是以一個商業導向的架構，在公司內部建置了一個 Portal，處理所有的公司員工的檔案傳輸以及稽核的運作，多個用戶如果操作的路徑相同，則可以減少計算 root hash 的時間，讓即時稽核所需的時間降低，以及因為 Portal 與用戶是同一個網路環境，所以會減少用戶傳輸檔案的時間。但是這樣的架構一般使用者並無法直接使用，在稽核上也只有確認檔案的完整性，用戶與服務商雙方也沒有交換證據來達成 CIWF 特性。

最後我們將以上相關研究作了以下的分析與比較，包含是否支援即時稽核、不可否認性問題以及支援多個用戶。

表 四.1：與其他研究之比較

	即時稽核	不可否認性	支援多個用戶
SiRiUS	X	X	O
SUNDR	O	X	O
Venus	X	X	O
CloudProof,	X	O	O
Iris	O	X	O
我們的研究	O	O	O

第五章 結論

在雲端儲存用戶和服務提供商之間，用戶讀到的檔案是否為最新，或者是有沒有被惡意更改的問題，這是非常重要的。又若是用戶某一天發現某個資料夾或者檔案不見了，就會向服務提供商賠償，雙方如何提出證據證明自己是對的，例如可能是使用者自己忘記而刪掉或故意詐騙，服務提供商如何提出證據保護自己避免敲詐。

本文提出一個機制，讓多個用戶可以跟服務商存取檔案時，每個用戶都能確保讀到檔案的正確性以及新鮮性，一般作法使用 Merkle tree 解決，而我們不僅降低使用者需儲存 Merkle tree 的負擔，並以鏈結雜湊解決不可否認性的問題。並以一系列的實驗證明我們在每個步驟上花的時間，是可以達到即時稽核的。服務商以及用戶都遵守這樣的機制，我們可以在一整個目錄，或是只有特定重要檔案的目錄簽訂合約，用戶即可不用擔心檔案是否會被修改，或是一個用戶有多個設備時，都能同時保證正確性。

參考著作

[1] *"Google Drive," https://drive.google.com/start#home*

[2] *"Dropbox," https://www.dropbox.com/home*

[3] *"SugarSync," https://www.sugarsync.com/*

[4] *"Microsoft SkyDrive," http://skydrive.live.com/.*

[5] *"Box," http://www.box.net*

[6] *AMAZON. "Amazon S3 Service Level Agreement, ". http://aws.amazon.com/s3-sla/.*

[7] *MICROSOFT CORPORATION. "Windows Azure Pricing and Service Agreement," http://www.microsoft.com/windowsazure/pricng/.*

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

[9] *Alexander Shraer, Idit Keidar, Christian Cachin, Yan Michalevsky, Asaf Cidon, and Dani Shaket, "Venus: Verification for untrusted cloud storage," ACM CCSW 2010.*

[10] *Seny Kamara and Kristin Lauter. "Cryptographic Cloud Storage," in Financial Cryptography Workshops, pp. 136-149, January 2010.*

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

[12] *Gwan-Hwan Hwang, Jenn-Zjone Peng, and Wei-Sian Huang. A Mutual Nonrepudiation Protocol for Cloud Storage with Interchangeable Accesses of a Single Account from Multiple Devices*

[13] *E. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. In Proc. Network and Distributed Systems Security Symposium (NDSS 2003), pages 131–145, 2003.*

[14] *M. Kallahalla, E. Riedel, R. Swaminathan, Q.Wang, and K. Fu. Plutus: Scalable secure file sharing on*

untrusted storage. In Proc. 2nd USENIX Conference on File and Storage Technologies (FAST), 2003.

[15] *M. T. Goodrich, C. Papamanthou, R. Tamassia, and N. Triandopoulos. Athos: Efficient authentication of outsourced file systems. In Proc. Information Security Conference 2008, 2008.*

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

[17] *The Apache Software Foundation, "Welcome to Apache Hadoop!" http://hadoop.apache.org/.*

[18] *A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, Available, and Eliable Storage for an Incompletely Trusted Environment," In OSDI, pages 1–14, December 2002.*

[19] *J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. "Oceanstore: An Architecture for Global-scale Persistent Storage," In ASPLOS, December 2000.*

[20] *G. Ganger, P. Khosla, M. Bakkaloglu, M. Bigrigg, G. Goodson, S. Oguz, V. Pandurangan, C. Soules, J. Strunk, and J. Wylie. Survivable storage systems. In DARPA Information Survivability Conference and Exposition, IEEE, volume 2, pages 184–195, June 2001.*

[21] *P. Druschel and A. Rowstron. Storage management and caching in PAST, a large-scale, persistent peerto-peer storage utility. In SOSP, 2001.*

[22] *J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger, "Self-securing storage: protecting data in compromised systems," In OSDI, October 2000.*

[23] *Jinyuan Li, Maxwell Krohn, David Mazie`res, and Dennis Shasha, "SUNDR: Secure untrusted data repository," In OSDI (2004).*

[24] *C. Cachin, A. Shelat, and A. Shraer. Efficient fork-linearizable access to untrusted shared memory. In Proc. 26th ACM Symposium on Principles of Distributed Computing (PODC), pages 129–138, 2007.*

[25] *M. Majuntke, D. Dobre, M. Serafini, and N. Suri. Abortable fork-linearizable storage. In T. F. Abdelzaher, M. Raynal, and N. Santoro, editors, Proc. 13th Conference on Principles of Distributed Systems (OPODIS), volume 5923 of Lecture Notes in Computer Science, pages 255–269, 2009.*

[26] *C. Cachin and M. Geisler. Integrity protection for revision control. In M. Abdalla and D. Pointcheval, editors, Proc. Applied Cryptography and Network Security (ACNS), volume 5536 of Lecture Notes in Computer Science, pages 382–399, 2009.*

[27] *Jun Feng, Yu Chen, Douglas H. Summerville: "A fair multi-party non-repudiation scheme for storage clouds," in Collaboration Technologies and Systems (CTS), pp. 457-465, May 2011*

附錄A

```
<?xml version="1.0" encoding="UTF-8"?>
<FileSkeleton>
   <fileObject type="dir">
      <name>d1</name>
      <hash>B13ACBDD6F5247145696279C777EF2463B97F3C7</hash>
      <containedFileObject>
         <fileObject type="dir">
            <name>d2</name>
            <hash>5F8D1754BFAC13502BB9A32C73C3D0CF76918B91</hash>
            <containedFileObject>
               <fileObject type="dir">
                  <name>d3</name>
                  <hash>5247D8B00331AC2CCEE0812CA988D989F96B8206</hash>
                  <containedFileObject>
                     <fileObject type="file">
                        <name>f5.txt</name>
                        <hash>547CD2BA3A17B483651496C5BA8C78F1789B5CBD</hash>
                     </fileObject>
                     <fileObject type="file">
                        <name>f6.txt</name>
                        <hash>FE91C0394869857C0E93272302F4D04FDE05A402</hash>
                     </fileObject>
                  </containedFileObject>
               </fileObject>
               <fileObject type="file">
                  <name>f3.txt</name>
                  <hash>619AAE029DDA528253A6AF0BA619B45BAA1DF115</hash>
               </fileObject>
               <fileObject type="file">
                  <name>f4.txt</name>
                  <hash>ADFEC5772AE8932AA10896037B0779BEC915015B</hash>
               </fileObject>
            </containedFileObject>
         </fileObject>
         <fileObject type="file">
            <name>f1.txt</name>
            <hash>C09BB890B096F7306F688CC6D1DAD34E7E52A223</hash>
         </fileObject>
         <fileObject type="file">
            <name>f2.txt</name>
            <hash>CF1126F67238BF3E85FCC8C8737B72E80DDCFDDB</hash>
         </fileObject>
      </containedFileObject>
   </fileObject>

</FileSkeleton>
```

圖 A.1：Merkle tree 範例

(A)

$Q_i=(OP_i,[OP_i]_{pri(U)})$

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
   <Operation>
      <Type>WRITE</Type>
      <Data_Location>\Attestation\Data\data-0.txt</Data_Location>
      <Data>c3VtbWVy…</Data>
   </Operation>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo><CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
         <Transforms>
           <Transform
              Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>0psNpD5xJ9A6BQp54eku6ESgdUM=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>INxwW92YApqpEiOX0e46EJ9XVd+3…cR2VQ0FrPQ==
      </SignatureValue>
   </Signature>
</Request>
```

(B)

$R_i=(Q_i, [Q_i, CH_{i-1}]_{pri(P)})$

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
  <Request>
   <Operation>
      <Type>WRITE</Type>
      <Data_Location>\Attestation\Data\data-0.txt</Data_Location>
      <Data>c3VtbWVy…</Data>
   </Operation>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo><CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
         <Transforms>
           <Transform
              Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>0psNpD5xJ9A6BQp54eku6ESgdUM=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>INxwW92YApqpEiOX0e46EJ9XVd+3…cR2VQ0FrPQ==
      </SignatureValue>
   </Signature>
  </Request>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo><CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
         <Transforms>
           <Transform
              Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>O0/eYWiIFcYn6E3Zpc3kQjba1fQ=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>yFM3YyIJZtwb1ZsQNeOiUNmpJ/4kOLVJCCp…ayiZjuNduK5uGLGfwCg==
      </SignatureValue>
   </Signature>
</Response>
```

(C)

$RR_i=(R_i, [R_i]_{pri(U)})$

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseOfResponse>
  <Request>
   <Operation>
      <Type>WRITE</Type>
      <Data_Location>\Attestation\Data\data-0.txt</Data_Location>
      <Data>c3VtbWVy…</Data>
   </Operation>
```

```
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo><CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
         <Transforms>
           <Transform
             Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>0psNpD5xJ9A6BQp54eku6ESgdUM=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>INxwW92YApqpEiOX0e46EJ9XVd+3…cR2VQ0FrPQ==
      </SignatureValue>
   </Signature>
  </Request>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo><CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
         <Transforms>
           <Transform
             Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>TLQEkl6MSTmEwVzdMdF2Yvv0JiY=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>Vp/CGWlUZsLXybOYWVxjIdZpNCKzU9nI6VVo1cCDi1qTnX…+HrMKKMrlzoL3EZ
A==
      </SignatureValue>
   </Signature>
</ResponseOfResponse>
```

圖 A.2：鏈結雜湊中的訊息範例