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

多個參與者利用鏈結雜湊抵禦回復式攻擊
Resistance of Replay Attack for Chain Hashing in Multiple Participants

研究生：　彭震宗　　撰

中華民國　　102　　年　7　月

摘 要

多個參與者利用鏈結雜湊抵禦回復式攻擊

彭震宗

在使用者與服務提供者之間保持雙向的不可否認性在雲端儲存服務上是非常重要的。根據這個問題，其中一種解決方法是紀錄雙方的行為當作證據，並為這些證據加上電子簽章，以便未來有爭議時，可以相互檢視。對於每個服務的請求，使用者和服務提供者必須交換儲存的證據，這些記錄下來的證據將會在未來稽核使用者儲存在雲端儲存空間的資料之時，驗證之前所有存取的行為是否正確。

利用鏈結雜湊的資料結構將存取資料的行為記錄成證據並將他們儲存在服務提供者端，可以有效的保證資料寫入的循序性及資料讀取的最新性。但是，單一使用者在不同的裝置上交錯的讀取或是寫入資料的情況下，原始的鏈結雜湊是不足夠的。在本篇論文中，我們首先證明，單一使用者在不同的裝置上交錯的讀取或是寫入資料的情況下，原始的鏈結雜湊的資料結構沒有辦法抵禦服務提供者所發動的回復式攻擊，除非客戶端必須儲存所有的證據或是存在一種方法，可以將每次存取資料的運算證據廣播給所有的客戶端的裝置。

為了解決這個問題，我們提出了一個架構，即使客戶端的裝置不用交換任何的證據，也可以保證使用者和服務提供者的雙向不可否認性，在這個架構下，每

個客戶端裝置只需要儲存他們最後一個存取資料的運算所產生的證據，伺服器端則需要儲存所有運算的證據。然而，一段時間後，雲端儲存服務的伺服器會累積大量的證據，造成伺服器的負擔。因此，我們還利用雜湊樹(hash tree)，或稱為馬可樹(Merkle tree)建構資料系統的骨架，以便消除累積的證據。

再者，我們利用 java 程式語言實作了上述提出的概念，以證明這個方法是可行的，雲端儲存服務的提供者可以在他們的服務層級協議中，利用這個架構來保證與使用者的不可否認性。

關鍵字：雲端儲存、雲端安全、服務階層協定

ABSTRACT

Resistance of Replay Attack for Chain Hashing in Multiple Participants

by

Jen-Zjone Peng

Obtaining mutual nonrepudiation between the user and service provider is crucial in cloud storage. One of the solutions for mutual nonrepudiation is based on logging attestations, which are signed messages. For every request, clients and service provider exchange attestations. These attestations will be used in an auditing protocol to verify their behavior. The chain-hashing scheme chains attestations and stores them in service provider for supporting write-serializability and read freshness of files. However, the chain-hashing scheme is inefficient when files in an account can be accessed by multiple client devices interchangeably.

In this paper we first show that the chain-hashing scheme cannot resist roll-back attack from service provider unless client devices keep all the attestations or there exists a way to broadcast the last attestation to all the client devices.

We propose a scheme that can guarantee mutual nonrepudiation between the user and service provider without requiring the client devices to exchange any messages, and each client device only has to store the last attestation it received. We also propose how to apply the hash tree to remove accumulated attestations. The results from related experiments demonstrate the feasibility of the proposed scheme. A service provider of cloud storage can use the proposed scheme to provide a mutual nonrepudiation guarantee in their service-level agreement.

Key words: Cloud Storage, Cloud Security, SLA

誌　謝

感謝黃冠寰老師在這兩年的指導下，讓我感覺受益良多，每當研究遇到瓶頸時，都會慢慢地引導我找出適當的方向，更讓我在解決問題的同時，領悟到研究科學的精神，而不僅僅只是一般書本上隨處可見的知識，當然最後還有不錯的成果，可以讓我出國看世界，見識國際規模的會議，以及國外的風情文化。此外，我也感謝我的父母親盡心盡力栽培我，並全力支持我取得碩士學位，使我能專心於課業以及研究，不必擔心學業以外的事情。另外，要感謝這兩年遇到的學長與同學，感謝哲生學長、李祺學長以及恆毅學長，傳授了很多寶貴的科學知識與工作上的實務經驗，讓人受益良多。感謝我的同學們，翊展、偉賢、思鋒，因為你們的陪伴，我的碩士生涯過得很精采，懷念我們一起寫的作業，一起討論的問題、一起為了中華電信計畫奮鬥，我們會是永遠的朋友，感謝我的學弟，裕偉，加入了新血，實驗室變得生氣盎然，合作與討論更加熱烈，感謝這些日子在我身旁的所有朋友，因為你們的支持，我今天才能有這樣的成就。

彭震宗　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

表格目錄

圖目錄

第一章緒論

雲端儲存空間(Cloud storage)是一種透過網際網路的線上儲存服務，使用者的資料都儲存在服務提供者利用虛擬機器所創建的虛擬儲存環境中，服務提供者創建大型的資料中心，讓使用者可以在這些大型的資料中心存放資料，使用者可以依照各自的需求向服務提供者購買或是租賃儲存空間，雲端儲存服務可以藉由應用程式介面或是網頁基底的使用者介面來使用服務，一些常見的雲端儲存系統包含Google Drive [1]、Dropbox[2]、SugarSync [3]、SkyDrive [4]、and Box[5]。

第一節　雲端儲存的安全性議題

然而儲存重要資料在雲端儲存空間中可能會有許多嚴重的安全性問題，服務提供者可能會洩漏具保密性的資料、擅自修改使用者的資料，回傳不正確的資料給使用者。導致出現這幾個問題的原因，有可能是伺服器端的程式有錯誤、伺服器損毀、運算出錯，或是系統不正常的設定。除此之外，來自競爭對手惡意的安全性攻擊可能更難抵禦而且比上述這些意外的因子還要危險，外部的攻擊者可能非法的存取服務提供者儲存的使用者資料，或是服務提供者內部的員工不小心允許了來自外部的攻擊，即使服務提供者建置了非常完善的安全環境，這些跳板還是有可能導致安全性問題。

現行沒有任何雲端儲存服務在他們的服務層級協定(service level agreement)中保證安全性無虞，就如同Amazon s3[6]和Microsoft Azure[7]只保證可用性，他

們的服務層級協定訂定了，可用性(availability)低於99.9%，服務提供者必須向使用者賠償一些金錢。基本的安全功能包含身分驗證(authentication)、保密性(confidentiality)、資料的完整性(data integrity)以及不可否認性(non-repudiation)，然而在加密的雲端儲存空間(cryptographic cloud storage) [8]中，身分驗證、保密性、資料的完整性這三個功能可以很容易地達成。

在加密的雲端儲存服務文中所示，服務提供者是完全不可信任的，而且服務的核心性質是(1)使用者自己掌控並維護自己的資料(2)安全性質來自加密演算法而不是系統的規範、實體的安全部屬(防火牆)或是存取控制。加密的雲端儲存服務利用加密演算法提供了資料的保密性，每一筆儲存在雲端上的資料，都被使用者的私密金鑰簽署過，以保證資料的完整性。然而，使用者仍然無法確保他們的資料是否會遺漏，又或是當從雲端上檢索資料時，是否是最後修改過的資料。考慮一種情況，由於某些內部的系統錯誤或是惡意的安全性攻擊，某些使用者儲存在雲端儲存系統上的資料會毀壞或是被整個銷毀，服務提供者有可能利用以前備份的資料回復，包含使用者所簽屬的電子簽章，而且可以否認使用者最後一個版本的資料遺失，這樣的攻擊稱為回復式攻擊(roll-back attack)[9]，或是重送攻擊(replay attack)[10]。

第二節　商業上的安全性議題－雙向的不可否認性

一個常見的問題就是使用者跟服務提供者之間的不可否認性是沒有辦法保證的。我們需要一個架構能讓服務提供者保證，當使用者資料有問題時能證明服

務提供者是無辜的，是使用者自己管理不當，又或者是當雲端儲存的伺服器發生系統錯誤，導致使用者資料損毀或遺失，根據這樣的情況，讓使用者證明是服務提供者出問題，這樣的證明稱之稽核(Auditing)[11]。一旦建立了雙向的不可否認性，使用者和服務提供者便可以根據使用者需要的安全性以及預算，來簽訂適當的商業合約，當證明是服務提供者發生了資料安全性問題，或是惡意竄改使用者資料，服務提供者必須保證支付補償金。

這篇文章指出了設計雲端儲存系統的問題，商業上的雲端儲存服務必須能夠偵測儲存的資料是否會違反安全性性質，Popa提出了一個系統稱為CloudProof，這個系統包含了偵測以及驗證四個安全性性質的協定，包含保密性(confidentiality)、完整性(integrity)、寫入的循序性(write serializability)以及讀取的最新性(read freshness)，這裡簡稱為CIWF[11]。使用者可以偵測雲端儲存空間上的資料是否有違反上述四個性質,若是違反則可以向服務提供者索取賠償。反之，也可以讓服務提供者確認使用者是否故意假造系統錯誤並向提供者敲詐，因此我們可以說CloudProof這個系統的架構是能夠確保雙方的不可否認性，這樣的保證源自於儲存的證據，透過簽章過的證據可以讓使用者了解儲存在雲端上資料的狀態。使用者和服務提供者在每一次的請求都交換證據，這些證據都利用鏈結雜湊的方式記錄下來，所以每一個使用者的裝置都必須儲存最後一個擁有鏈結雜湊的證據。此外，服務提供者必須儲存所有請求的證據以便未來使用者稽核時驗證。當使用雲端儲存服務時，這個協定在CloudProof中能夠維持雙方的不可否認性。

然而，這樣的協定在多個裝置利用單一個帳戶時，裝置交互的使用會沒有辦法保證，某個裝置可以取得最新的證據，除非有一種方法可以確保所有客戶端的裝置都可以得到最新的證據，否則將有可能被服務提供者發動回復式攻擊。

第三節　情境說明與應用

透過網際網路，我們可以很便利的使用一些服務，像是雲端儲存服務、網路銀行，或是其他的網路服務，但是在這便利之下衍生了一些問題，如圖 一-1 所示，把 Resource Pool 當作一個服務提供者，好比說使用雲端儲存服務，三個使用者可以共享一份文件，假設使用者 A 寫入一份文件，而且使用者 B 想要讀取使用者 A 做的變更，接者使用者 C 也對文件內容作更變，然而網路延遲或是服務提供者伺服器發生錯誤造成使用者 C 更變文件後，使用者 B 才讀取到，會導致使用者 B 讀取到錯的文件，因此寫入的順序沒有有效的管理，使用者們最後得到的資料會不一致。

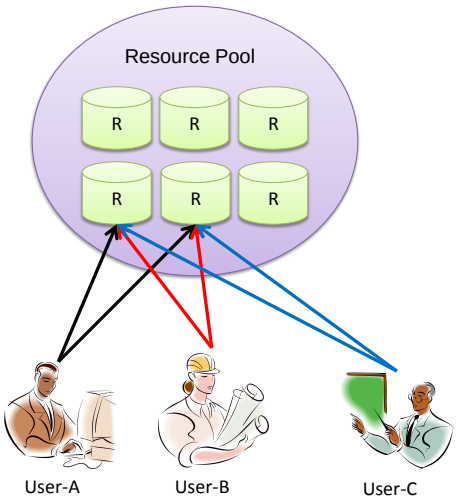

圖 一-1 單一個資源被多個參與者使用

第一項情境

根據上一個小節的情境讓我們了解到多個參與者並行的使用相同資源的問題，會造成結果不一致，導致使用者取得不正確的結果。在解決此問題之前，我們先透過幾個真實的例子來更深刻的了解。

情境一

Dropbox 雲端儲存服務供應商可以讓使用者透過行動裝置上傳文件，而且還可以將上傳文件分享給其他使用者。假設現在使用者 A 是一個律師，他和公司主管去國外簽商業合約並利用 Dropbox 的服務，將他負責的公司商業合約上傳並分享給公司老闆。此時 dropbox 內部發生損毀或是遭受攻擊導致使用者 A 的資料毀壞，Dropbox 為了避免賠償問題，可能以錯誤的資料造假，讓使用者渾然不覺。

情境二

假設某個使用者向銀行註冊了帳戶,透過雙方共同協議,允許帳戶可以透支，即變成信用借貸，因此每一個交易都必須要有明確的先後關係，否則雙方會存在爭議點。用一個例子如圖 一-2，假設現在使用者 A 開了帳號並存了 b 元進去，接者有合作關係的銀行匯了 X 元近來，此時帳戶的擁有人想提 Y 元，且 Y>b，由於是兩個獨立的交易，雙方都不知道他們是否都完成交易，因此在 S1 的情況下，銀行想敲詐使用者賺取利息，故意將交易的順序更改，使的使用者必須付利息給銀行，因此造成使用者的損失。當匯款或是轉帳的機構大於兩個，各個機構的交易明細都是機密，雙方不可能交換他們的紀錄，因此得知交易順序的只有銀行，若是銀行真的有意從新排序交易順序，使用者將會無法得知錯誤。

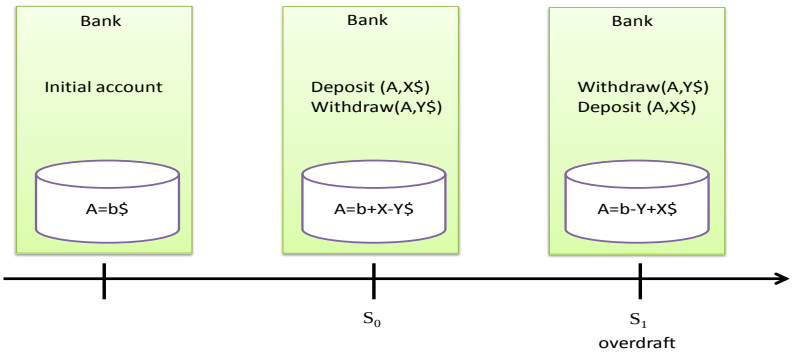

圖 一-2 銀行服務的情境

情境三

免費軟體專案通常都是由來自世界各地的開發者開發，並將開發結果上傳到第三方原始碼管理中心。以 sourceforge.net 來說，他們管理超過兩萬個軟體包(source code package)，而且這些原始碼撰寫的環境可以來自各種不同的系統，因此沒有一種方法可以有效的驗證這些原始碼的正確性，當存放原始碼的伺服器被惡意攻擊或是伺服器內部毀壞，開發人員的原始碼可能會被竄改或是消失，最嚴重的情形是被惡意竄改,加上了後門程式的原始碼,下一個下載這份原始碼的人，很有可能就被惡意攻擊，根據這個問題，最大的癥結點在於沒有方法可以有效的管理檔案寫入的順序性以及完整性，讓開發人員能偵測出原始碼更動過。

第四節 論文組織與結構

在本篇論文中，我們首先論證，在單一帳戶下用不同的裝置使用雲端儲存服務時,除非存在一個方法可以將最後一個請求的證據,廣播給其他的客戶端裝置，否則基本的鏈結雜湊是不足以抵擋，服務提供者利用已儲存的證據發動的回復式

攻擊。我們提出了一個新的架構能夠維持雙向的不可否認性，在這個架構下，每一個客戶端的裝置進行資料請求時，都不需要交換證據，而且每個裝置只需要保有他們最後一個請求的證據。當請求不斷增加時，累積過多的證據會造成伺服器的負擔，因此我們提出利用雜湊樹來清除累積的證據，並且滿足CIWF性質，在最後我們也做了許多的實驗，並且分析在我們提出的架構下，會有那些額外的負擔，實驗的結果證明了我們的架構是可行的。

本篇論文的組織如下：第二章說明了為什麼原始的鏈結雜湊是不足夠的，第三章提出了在單一帳戶下不同的裝置交錯的使用之解決方案，第四章描述如何利用雜湊樹清除累積的證據，第五章提出了實作的細節以及實驗的結果，第六章描述相關的研究，在第七章對我們研究做了總結。

第二章 鏈結雜湊在平行運算時的問題

在這個章節，我們要論證原始的鏈結雜湊架構在單一帳戶下，使用不同的裝置來交互存取雲端上的資料會衍生的問題。首先，我們先說明原始的鏈結雜湊的運作模式，為了更正式的描述鏈結雜湊，我們先定義幾個符號，$[O]_{Pri(x)}$表示電子簽章簽屬資料物件(Data object)O，並且利用使用者 X 的私有金鑰簽屬，此外在中括號裡面以逗號隔開，表示多個資料物件被使用者 X 簽屬，如例子，$[O_1, O_2, O_3]_{Pri(x)}$表示資料物件 O1,O2,O3 被使用者 X 的私有金鑰簽屬。以下步驟說明鏈結雜湊架構如何運作:

Step 1: 當某個使用者 U 想要在他的用戶端存取他私人的文件，他會先發送一個請求訊息(Request Message) Q_i, 此外 $Q_i=(OP_i, [OP_i]_{pri(U)})$給服務提供者(Service Provider)，OP_i 表示請求的運算，可以是開新文件、更新文件、或是讀取文件。

Step 2: 當服務提供者收到從使用者送來的請求 Q_i，首先，利用 U 的公開金鑰檢查電子簽章$[OP_i]_{pri(U)}$是否正確，若是正確則回傳回應訊息(Response Message) R_i,此外 $R_i=(Q_i, [Q_i, CH_{i-1}]_{pri(Provider)})$給使用者 U。其中 CH_{i-1} 是 $(i-1)^{th}$ chain hash 而且 $CH_{i-1}=[Q_{i-1}, CH_{i-2}]_{pri(Provider)}$。

Step 3: 步驟三:當使用者 U 收到了 R_i，首先，利用步驟一使用者發送的 Q_i 以及 $CH_{i-1}=[Q_{i-1}, CH_{i-2}]_{pri(Provider)}$進行驗證，若驗證成功，則使用者發送回覆回應訊息(reply-response message)給服務提供者，此外 $RR_i=(R_i, [R_i]_{pri(U)})$。服務提供

Step 4: 者要儲存所使用者發送的回覆回應訊息已備未來驗證之用途。

Step 5: 服務提供者執行剛才使用者提出的請求，假設執行結果為 L_i，服務提供者回傳服務承認訊息(acknowledgement message)給使用者 U，此外，$ACK_i=(L_i, RR_i, [L_i, RR_i]_{pri(U)})$。使用者 U 必須持有最新的服務承認訊息，這表示使用者端的證據，未來雙方產生爭議時，使用者必須要拿出此證據來證明對錯。

圖 二-1 k 個運算交換訊息

表示利用鏈結雜湊架構執行 k 個運算，表格內呈現所有的運算訊息。CH_0 是事先定義好的，當使用者和服務提供者協議要用鏈結雜湊架構時，雙方必須事先定義一個基準。在執行 k 個運算後，服務提供者必須要儲存所有的證據 (i.e., RR_1, RR_2, …, RR_k)而且使用者只需要存最新的證據(i.e., RR_K).客戶端回傳的證據 RRj+1 被鏈結在 RRj 之後，且 $1 \le j \le k-1$。這表示加密演算法可以被用來驗證 RR_j 是 RR_{j+1} 之前的回覆回應訊息。圖 二-2a 假設現在使用者的使用狀態為α，並且已經執行了 k 個運算(i.e. OP_1, OP_2, …, OP_k)，我們可以很容易的利用這 k 個運算來解析狀態 α′。所以在雙方要使用這個協定時，服務層級協定必須規定服務提供者保存所有的伺服器端證據(server-side attestations) (i.e., RR_1, RR_2, …, and RR_k)，這可以讓我們用來驗證寫入的循序性 和讀取的最新性: (1) 使用者 U 提供他最新的 ACK_k, 和(2) 服務提供者必須提供 RR_1, RR_2, …, 和 RR_k. 就像上述說明的，我們可以知道所有運算的鏈結關係。使用者和服務提供者之間的不可否認性問題是不存在的，在後面章節，我們將探討如何消除所有累積的證據。

$Q_1=(OP_1,[OP_1]_{pri(U)})$ $R_1=(Q_1, [Q_1, CH_0]_{pri(Provider)})$ $RR_1=(R_1,[R_1]_{pri(U)})$ $ACK_1=(L_1, RR_1, [L_1, RR_1]_{pri(U)})$
$Q_2=(OP_2,[OP_2]_{pri(U)})$ $R_2=(Q_2, [Q_2, CH_1]_{pri(Provider)})$ $RR_2=(R_2,[R_2]_{pri(U)})$ $ACK_2=(L_2, RR_2, [L_2, RR_2]_{pri(U)})$
. . .
$Q_{k-1}=(OP_{k-1},[OP_{k-1}]_{pri(U)})$ $R_{k-1}=(Q_{k-1}, [Q_{k-1}, CH_{k-2}]_{pri(Provider)})$ $RR_{k-1}=(R_{k-1},[R_{k-1}]_{pri(U)})$ $ACK_{k-1}=(L_{k-1}, RR_{k-1}, [L_{k-1}, RR_{k-1}]_{pri(U)})$
$Q_k=(OP_k,[OP_k]_{pri(U)})$ $R_k=(Q_k, [Q_k, CH_{k-1}]_{pri(Provider)})$ $RR_k=(R_k,[R_k]_{pri(U)})$ $ACK_k=(L_k, RR_k, [L_k, RR_k]_{pri(U)})$

圖 二-1 k 個運算交換訊息

接下來，我們將解釋為什麼鏈結雜湊架構不足以應付一種常見的情形，那就是在單一帳戶下被多個客戶端裝置交互的使用。以一個例子來說，Dropbox 雲端儲存服務提供者可以讓智慧型手機利用某個帳戶自動的上傳照片，而且他還可以開啟資料分享的功能，將他分享給朋友。

我們假設每個客戶端的裝置在存取資料時，都利用同一個帳戶並且不交換使用者端的證據，而且為了節省使用者端的資源利用率，每個客戶端的裝置都只儲存最後一個從服務提供者送來的使用端證據，以下有幾個原因是為什麼使用者端的裝置很難交換最後一個使用者端的證據，以確保每個使用者端的裝置都有最新

的證據：(1)顯而易見的，並不是每個使用者端的裝置都一定同一時間上線，(2)使用者可能在他的帳號加入新的裝置以便使用，(3)因為使用者的裝置有可能頻繁的交互的使用，所以在使用者的裝置間交換最後一個使用者端的證據有可能需要極大的成本，(4)由於使用者的裝置有可能是記憶體很小的嵌入式裝置，所以在使用者端的裝置上儲存所有的使用者端證據不太可行。

我們現在要描述為什麼服務提供者可以發動回復式攻擊，就像是圖 二-2A。假設現在有 x 個客戶端裝置都具有權限存取使用者 U 的資料，而且這些裝置已經成功做了 N 次的運算，分別為 $OP_1, OP_2, \ldots, OP_N$。此外，我們假設現在儲存在雲端上的資料有部分已經損毀(可能伺服器損毀或是遭惡意攻擊)，此時服務提供者企圖還原這些資料。假設使用者 U 在做這 N 次運算之前資料的原始狀態為α，而且系統狀態只能回復到α′，表示 OP_k 之後的運算全部遺失了，此時服務提供者試圖用不正當的手法將資料回復到原始狀態如以下步驟。首先，服務提供者試圖隱藏伺服器端儲存的證據，從 RR_{k+1} 到 RR_N，由於這些證據都是存在伺服器上，因此所有客戶端的裝置都不會發現這個伺服器違規的動作，並且持續地發送要求，而且每個客戶端都沒有最新的證據，他們沒有辦法確認伺服器端是否就像步驟三一樣，正確地將他們的要求鏈結起來，因此客戶端在當下為了能讓要求順利執行，只能不檢查證據的正確性，持續的更新做過的運算，當 X 個客戶端裝置都做過運算並且持續在 OP_k 之後更新，最後所有的客戶端都成功更新證據，而且 Server 將這些 OP 都鏈結進 OP_k 之後，如圖 二-2 B 所示，服務提供者已經將伺服器端的證

據都鏈結起來了，如 $RR_1, \ldots, RR_k, RR_q, \ldots, RR_M$。因為所有的客戶端裝置都成功做了新運算 $RR_q, \ldots, RR_M$，在這個例子底下，使用者將無法確認之前遺失的資料是否有誤，造成使用者端的損失。根據這個例子，可能的作法是使用者必須要有一些技術來分享各個裝置的使用者端證據，以確保所有的裝置都擁有最新的證據，但是這樣的做法並不適當，行動裝置有可能會沒電或是只想要做個簡短的服務，為此必須要分享證據，負擔很高。

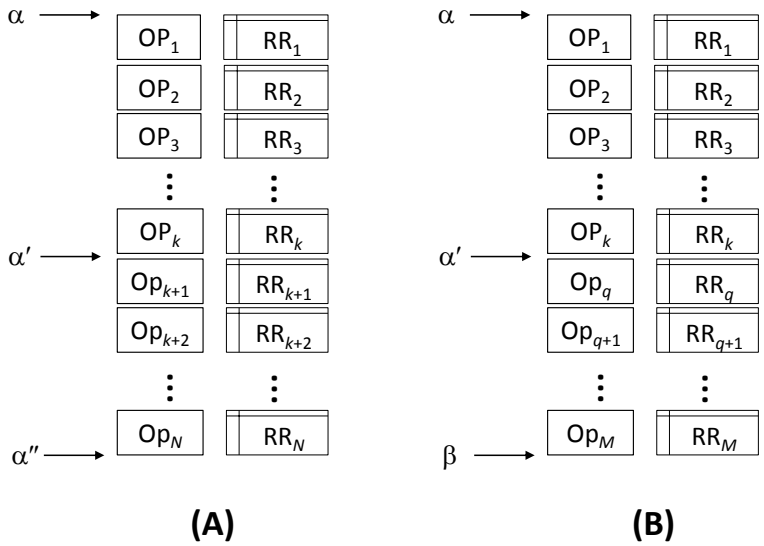

圖 二-2 回復式攻擊的例證

第三章一個實現不可否認性的協定

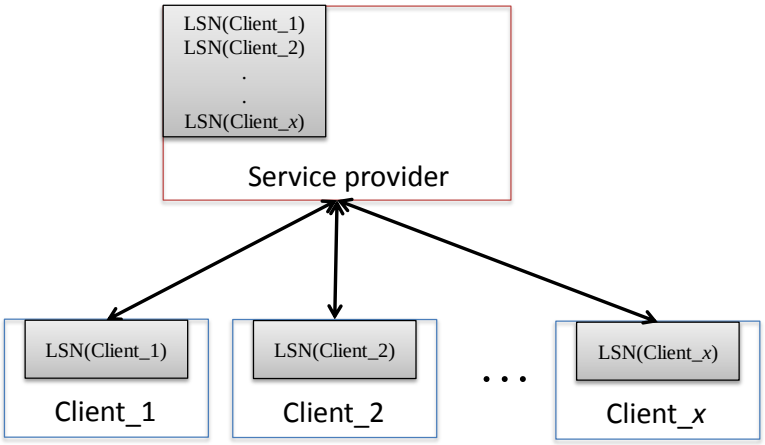

圖 三-1 在 C&L 架構下使用者帳戶維護的 LSN

在這個章節，我們提出了一個新的不可否認協定，考慮我們剛剛所提到的情形，多個客戶端裝置利用相同的帳戶交錯的對資料做存取，客戶端的裝置都不交換訊息，而且不需要額外的空間將所有使用者端的證據都儲存下來。

我們假設每個客戶端的裝置都有使用者的私鑰可以存取使用者 U 的帳戶。當某個新的客戶端裝置準備開始利用 U 的帳戶存取資料時，他首先要送 *client-ID-request* 訊息 Z=(Timestamp, RequestID, $[\text{Timestamp, RequestID}]_{pri(U)}$)給服務提供者申請唯一的客戶端識別名稱，時間戳記(Timestamp)表示現在的時間，請求身分識別(RequestID)這個訊息是指請求唯一的身分識別名稱，這個請求訊息還加上了電子簽章，所以服務提供者可以藉此認證這個訊息是從哪個使用者所發過來的，之後服務提供者回送訊息(Timestamp, EpochID, ClientID, [Timestamp,

EpochID, ClientID]$_{pri(Provider)}$)給客戶端的裝置，這裡的時間戳記(Timestamp)指的是產生此訊息的時間，時代識別名稱是指某一段交易的時間，不同的使用者(或帳戶)有不同的時代識別名稱。當所有的證據被銷毀時，我們將更變時代。每個客戶端的裝置都有唯一的客戶端識別名稱，如果將它遺失了，必須向伺服器重新申請。每個客戶端識別名稱都會連接一個正整數，稱為 local sequence number (LSN)，標記為 LSN(ClientID)。LSN 是從 1 開始連續不斷的成長。

參考到圖 三-1，服務提供者必須保留所有客戶端裝置的 LSN，以及每個使用者的時代識別名稱。客戶端的裝置只需要保留他們所申請的客戶識別名稱以及 LSN 和時代識別名稱。這個架構採用了鏈結雜湊以及 LSN，將他們嵌入在溝通訊息之中以確保雙方的不可否認性。

C&L 架構包含以下步驟：

Step 1: 當一個參與者想要存取某個共用資源時，這個參與者必須傳送一個要求訊息 $Q_{i,}$ 到服務提供者，此外 Q_i=(OP$_i$, ClientID, LSN(ClientID), [OP$_i$, ClientID, LSN(ClientID)]$_{pri(U)}$)。這裡的 OP$_i$ 表示一個請求運算，如果這個服務是雲端儲存伺服器，這個運算可以是寫入文件、讀取文件。若是網路銀行服務，這個運算可以是提錢、存錢。

Step 2: 當服務提供者收到 Q_i，他必須先驗證電子簽章([OP$_i$, ClientID, LSN(ClientID)]$_{pri(U)}$)是否正確，再者必須驗證 LSN(ClientID)的值是否有效(每個使用者的 LSN 必須連號)，最後，服務提供者回傳回應訊息 R_i, R_i=(Q_i,

EpochID, $[Q_i, \text{EpochID}, CH_{i-1}]_{pri(Provider)}$)給使用者。$CH_{i-1}$ 是 是第 i–1 次的鏈結雜湊而且 $CH_{i-1}=[Q_{i-1}, \text{EpochID}, CH_{i-2}]_{pri(Provider)}$。

Step 3: 當這個參與者收到 R_i，他必須先驗證 R_i 的電子簽章是否有效而且要確保時代識別名稱以及客戶端識別名稱是否正確。在這裡，這個參與者只能確認電子簽章以及上述這些因子，他沒有辦法確定 R_i 是否真的鏈結到 R_{i-1} 之後，原因是每一個參與者都只保有他們從服務提供者得到最新的承認訊息 ACK_j，$j<i$ 而且 j 不一定等於 i–1。當這個簡略的驗證成功後，這個參與者送回覆回應訊息 RR_i 給服務提供者，此時 $RR_i=(R_i, [R_i]_{pri(U)})$，在最後把 LSN(ClientID) 加一。這裡要注意的是服務提供者要保留所有的回覆回應訊息以供未來稽核之用。

Step 4: 服務提供者執行此參與者的要求，當執行結束後回傳承認訊息 ACK_i, where $ACK_i=(L_i, RR_i, [L_i, RR_i]_{pri(U)})$給使用者，$L_i$ 表示執行結果。

Step 5: 服務提供者將 LSN(ClientID)加一表示此運算完成

以下是 C&L 架構的例子，假設有兩個參與者分別為 ClientA 以及 ClientB 想要存取伺服器上的資源，並且這個時間點的時代識別名稱為 EPH000001

$Q_1=\{(OP_1, ClientA,1), [OP_1, ClientA,1]_{Pri(U)}\}$ $R_1=\{(Q_1,EPH000001), [Q_1,EPH000001, CH_0]_{Pri(Server)}\}$ $RR_1=\{R_1 ,[R_1]_{pri(U)}\}$ $ACK_1=(L_1, RR_1, [L_1, RR_1]_{pri(U)})$
$Q_2=\{(OP_2, ClientA,2), [OP_2,ClientA.2]_{Pri(U)}\}$ $R_2=\{(Q_2,EPH000001), [Q_1,EPH000001, CH_1]_{Pri(Server)}\}$ $RR_2=\{R_2 ,[R_2]_{pri(U)}\}$ $ACK_2=(L_2, RR_2, [L_2, RR_2]_{pri(U)})$
$Q_3=\{(OP_3, ClientB,1), [OP_3, ClientB,1]_{Pri(U)}\}$ $R_3=\{(Q_3, EPH000001), [Q_3, EPH000001, CH_2]_{Pri(Server)}\}$ $RR_3=\{R_3 ,[R_3]_{pri(U)}\}$ $ACK_3=(L_3, RR_3, [L_3, RR_3]_{pri(U)})$
$Q_4=\{(OP_4, ClientA,3), [OP_4, ClientA,3]_{Pri(U)}\}$ $R_4=\{(Q_4, EPH000001), [Q_4, EPH000001, CH_3]_{Pri(Server)}\}$ $RR_4=\{R_4 ,[R_4]_{pri(Client)}\}$ $ACK_4=(L_4, RR_4, [L_4, RR_4]_{pri(U)})$
$Q_5=\{(OP_5, ClientB,2), [OP_5, ClientB,2]_{Pri(U)}\}$ $R_5=\{(Q_5, EPH000001), [Q_5, EPH000001, CH_4]_{Pri(Server)}\}$ $RR_5=\{R_5 ,[R_5]_{pri(Client)}\}$ $ACK_5=(L_5, RR_5, [L_5, RR_5]_{pri(U)})$

圖 三-2 在 C&L 架構下五個運算的訊息

圖 三-2 對於客戶端 A 來說有三個證據，他的 LSN 是一、二、三。在 C&L 架構中，客戶端的裝置只須要維護他們自己的 LSN，若成功的完成對服務的請求，則每個客戶端裝置的 LSN 必須連續不可中斷，如果客戶端的裝置有問題，在跟服務提供者交換證據時，其 LSN 不連續，則服務提供者必須拒絕此次的存取。某個客戶端的裝置有可能遺失他們的 LSN，因此在這個情況下，每個客戶端的裝置必須重新像伺服器註冊新的客戶識別名稱，服務提供者的負擔包含了維護所有已註冊的客戶識別名稱的 LSN。每一個客戶端的裝置都必須儲存最後一個從服務提供者傳來的承認訊息。

定理(一):當我們使用 C&L 架構，服務提供者不可能啟動回復式攻擊

證明:

在 C&L 架構中，服務提供者必須儲存所有合法的伺服器端證據，並且利用這些證據做以下的四點的確認：(R1)所有的電子簽章是正確的、(R2)所有的證據裡面儲存的時代識別名稱必須相同而且正確、(R3)每個客戶識別名稱所連結的 LSN 都必須從一開始，並且要連續且唯一、(R4)在伺服器端的證據所包含的回應訊息是正確的鏈結，而且初始的雜湊值是雙方一開始就定義好的。若是依照上述 R1、R2、R3、R4 這四個需求做檢測，服務提供者不可能遺失或是在正確的存取順序中，插入任何合法的伺服器端證據造成使用者資料的錯誤，因為每一個客戶端的裝置都存有最後一個從服務提供者送來的承認訊息,在 C&L 架構下的第三步驟，客戶端的裝置沒有確認當下的回應訊息是否有正確的鏈結到前一個回應訊息中。然而，根據第四個需求，因為初始的雜湊值是雙方事先定義好的，服務提供者必須要正確的鏈結所有回應訊息，除非沒有在稽核的時候偵測錯誤，否則服務提供者不可能發動回復式攻擊。

回到 Figure2,假設服務提供者試圖要遺失或是隱藏伺服器端的證據(RR_{k+1} to RR_N)，由於 LSN 是由客戶端裝置所產生並加在請求訊息中的,這樣會造成客戶端裝置的 LSN 不連續。客戶端裝置知道現在正確的 LSN，而且最後一個承認訊息也可以驗證 LSN 的正確性。除了不可否認性之外，C&L 架構也可以保證寫入的順序性，以及讀取的最新性，因為在同一個帳戶下，每一個運算的回應訊息都被正確的鏈結在一起。

第四章 C&L 架構和雜湊樹

在一段長時間後，客戶端裝置理所當然的會有數以百計的資料運算，然而這樣的情況會造成服務提供者必須儲存大量的伺服器端證據造成系統負擔，因此我們必須要有方法*消除*這些證據，然而利用雜湊樹或是馬可樹(Merkle tree)[12]可以有效的解決這個問題。雜湊樹是由雜湊值所產生的，樹葉是將資料區雜湊過所產生的值，而且雜湊樹的最頂端是雜湊樹的樹根(root hash)。圖 四-1A 描述了雜湊樹。每一個在資料夾的資料都連結了各自的雜湊值。常用的加密雜湊演算法為SHA-1、Whirlpool、Tiger。資料雜湊的符號在這裡表示 *h*(資料名稱)。如資料 f6 的雜湊值為 h(f6)。此外，資料夾的雜湊值是資料夾內所有資料雜湊值的串接，也就是h(d2)=h(h(f3),h(d3),h(f4))。雜湊樹的樹根就是所有資料夾根的雜湊值串接在雜湊，如 h(d1)=h(h(f1),h(d2),h(f2))。

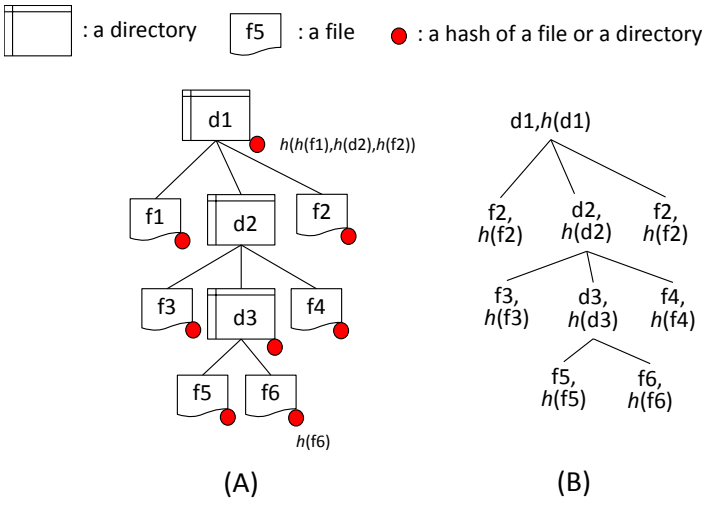

圖 四-1 資料夾的雜湊樹

在帳戶新增之後，使用者更新一些資料和包含資料的資料夾。首先，服務提供者和客戶端的裝置先各自算出他們所擁有資料的雜湊樹的樹根，然後比較他們得到的雜湊樹的樹根，同意這個雜湊樹的樹根為第一個時代所產生的雜湊樹的樹根。接者，服務提供者送出一個訊息為{(Root hash, EpochID), [Root hash, EpochID]$_{\text{Pri (Server)}}$}，確認並同意它為雜湊樹的樹根。之後，使用者回送簽章過的承認訊息。最後，服務提供者利用所有使用者的資料產生一個資料的骨架(*file skeleton* (FSK))並將它儲存起來。這裡要注意的是使用者也需要儲存協議過的雜湊樹的樹根。資料的骨架包含所有資料的結構、名稱、以及所有資料及資料夾的雜湊值，如圖 四-1B。後續的資料運算(包含讀取與寫入)，都要遵守第三章提及的協定。

我們現在要描述當稽核過後，或是累積過多的伺服器端證據，該如何消除證據。事實上我們是在同一時間進行稽核和驗證證據。在消除證據之後，我們轉換現在第 i 個時代到第 i+1 個時代。以下是消除證據的步驟(這裡要注意的是，使用者必須選一個客戶端的裝置來實施溝通的需求以及運算，我們假設協議好的第 i 個時代的雜湊樹的樹根已經存在裝置中):

Step 1: 服務提供者送第 i 個時代的 FSK F 給客戶端的裝置。

Step 2: 如果驗證 F 是正確的，客戶端的裝置使用 F 並且同意某個雜湊值是第 i 個時代的雜湊樹的樹根。

Step 3: 服務提供者回送伺服器端的證據給客戶端的裝置，這些證據就是先前提到的從客戶端的裝置收到的回覆回應訊息，我們假設它們為{RR_1, …, RR_z}。

Step 4: 客戶端的裝置驗證{RR_1, …, RR_z} 是否正確。

Step 5: 根據現在使用者帳戶的資料夾結構，服務提供者計算出最新的雜湊樹的樹根(這裡稱之γ)。

Step 6: 客戶端的裝置利用 F 以{RR_1, …, RR_z}推導出修正過 FSK F'的，並且根據 F'計算出雜湊樹的樹根(這裡稱為μ)。

Step 7: 服務提供者和客戶端的裝置比較它們剛才推算出的雜湊樹的樹根是否相同，如果γ等於μ，代表雙方同意此雜湊樹的根為 i+1 的時代的雜湊樹的樹根。服務提供者送出訊息{[μ, $(i+1)^{th}$ Epoch ID], [μ, $(i+1)^{th}$ Epoch ID]$_{Pri(Server)}$}給客戶端的裝置。

Step 8: 服務提供者產生現在資料夾狀態的 FSK，並將它儲存起來。這就是第 i+1 的時代的 FSK。

{RR_1, …, RR_z}這些證據可以在步驟八之後刪除，顯而易見的，並不是所有在相同帳戶底下的使用者裝置都會被通知轉換新時代，在開始一個新的時代後，連結每個客戶識別名稱的 LSN 都會重新從 1 開始，因此當某個客戶端裝置送請求訊息時，會發現 LSN 不正確，此時服務提供者會通知此裝置已經進入新的時代，客戶端的裝置必須重置 LSN，並且發送正確的請求訊息。

第五章 實作與實驗成果

我們進行了一系列的實驗來佐證我們的架構是可行的，在第一階段，我們量測了請求訊息、回應訊息、以及請求回應訊息的大小，再者是量測鏈結雜湊架構和 C&L 架構，認證回覆回應訊息正確性所需的時間。

首先我們定義了請求訊息、回應訊息，回覆回應訊息的格式。在實作上，我們利用可延伸標記式語言(XML)來表現這些訊息。在附錄，我們提供了實作鏈結雜湊以及 C&L 架構的請求訊息、回應訊息，回覆回應訊息的範例。

第一節 驗證回覆回應訊息所需的時間

由於在轉換到下一個時代之前，服務提供者必須儲存所有的回覆回應訊息，我們在鏈結雜湊以及 C&L 架構下，考慮回覆回應訊息的大小，表格 五-1 指出在鏈結雜湊以及 C&L 架構下，訊息的大小都線性的成長，而且都非常的小。這是為了在消除證據的步驟三、四、六展示{RR1, …, RRz}的大小。

表格 五-1 鏈結後的回覆回應訊息大小 (|RR|是訊息的個數)

Applied Scheme / |RR|	Chain hashing	C&L
10	23.6 kB	23.1 kB
100	240 kB	234 kB
1000	2340 kB	2290 kB
10000	23500 kB	22900 kB
100000	235000 kB	229000 kB

表格 五-2 表示在鏈結雜湊架構和 C&L 架構下，確認回覆回應訊息正確性所需的時間，我們可以發現鏈結雜湊架構稍稍的比 C&L 架構快一些，因為 C&L 還必須多驗證每個客戶端身分識別名稱所連結的 LSN 是否連續。這是在消除證據階段的步驟四。

表格 五-2 驗證一連串的回覆回應訊息正確性的時間

Applied scheme / |RR|	Chain hashing	C&L
1	0.113	0.124
10	0.187	0.219
100	0.655	0.702
1000	2.902	3.510
10000	122.133	124.301
100000	746.150	844.445

第二節 C&L 與 鏈結雜湊的執行的時間

在我們實驗的第二階段，我們量測在鏈結雜湊架構以及 C&L 架構下，進行資料存取時，所需的時間。一個資料的存取包含以下步驟：(1)客戶端的裝置發送一個請求訊息、(2)服務提供者發送回應訊息、(3)客戶端裝置回送回覆回應訊息、(4)服務提供者執行了對於資料存取的請求，接者回送確認訊息。需要注意的是，這些執行步驟在第四章節有詳細描述。

第一項 在相同網段執行的時間

首先，我們呈現客戶端裝置和服務提供者在**相同**網段下運作。兩個不同的運算包含讀取及寫入是在客戶端裝置隨機的選擇，每一次運算的資料大小為 1k、10k、100k、1M、10M 以及隨機選擇上述五種檔案進行運算，分別記錄在表格 五-3、

表格 五-4、表格 五-5、表格 五-6、表格 五-7、表格 五-8。表格中的數據都是以秒計。不論是 C&L 架構，或是鏈結雜湊架構，每一個運算所需的時間都非常接近。在表格 五-3 對於小資料來說，在兩個架構下跑單一個運算都小於 0.05 秒，然而用比較大的資料來作運算，將會花費多一些的時間處理資料的雜湊運算以及在服務提供者端進行資料讀取和資料寫入，如表格 五-7 所示，這樣每一次處理資料運算的平均時間大約 0.5 秒。

根據實驗結果，我們可以發現交換請求訊息、回應訊息、以及請求回應訊息，以及確認訊息，會產生一些負擔。這些負擔是建立完成安全的資料運算以及達成不可否認性所無法避免的代價。

表格 五-3 做資料運算所需的時間 (資料大小為 **1kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.590	0.059	0.622	0.062
100	4.527	0.045	4.620	0.046
1000	35.896	0.035	36.77	0.036
10000	325.994	0.032	379.057	0.037

表格 五-4 做資料運算所需的時間 (資料大小為 **10kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.340	0.034	0.156	0.035
100	3.435	0.034	3.872	0.038
1000	37.628	0.037	35.413	0.035
10000	359.269	0.035	364.73	0.036

表格 五-5 做資料運算所需的時間 (資料大小為 **100kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.387	0.038	0.388	0.038
100	3.981	0.039	4.137	0.041
1000	39.563	0.039	40.826	0.040
10000	403.900	0.040	434.041	0.043

表格 五-6 做資料運算所需的時間 (資料大小為 **1000kB**)

	CH Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.887	0.088	0.887	0.088
100	9.425	0.094	9.768	0.097
1000	98.047	0.098	91.698	0.091
10000	947.177	0.094	977.628	0.097

表格 五-7 做資料運算所需的時間 (資料大小為 **10000kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	5.600	0.560	6.224	0.622
100	57.596	0.575	51.746	0.517
1000	542.242	0.542	553.724	0.553
10000	5568.196	0.556	5572.788	0.557

表格 五-8 做資料運算所需的時間 (資料大小是隨機選擇 **(1kB,10kB,100kB,1000kB,10000kB)**的檔案)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.140	0.014	0.562	0.056
100	1.701	0.017	3.354	0.033
1000	14.32	0.014	16.473	0.016
10000	184.731	0.018	177.623	0.017

為了瞭解 C&L scheme 所產生的負擔，我們比較在相同網段，資料存取的時間以及 C&L 執行的時間，如表格 五-9、表格 五-10 所示，對於小資料，額外負擔的比例略高，對於大資料由於傳送使得額外負擔的比例略低。

表格 五-9 執行單一資料存取以及 C&L 架構之時間比較圖(一千個運算)

	Pure file access		**C&L Scheme**	
1000 個運算	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
1K	1.018	0.001	36.770	0.036
10K	1.357	0.001	35.413	0.035
100K	4.23	0.004	40.826	0.040
1M	33.648	0.033	91.698	0.091
10M	331.63	0.331	542.242	0.542

表格 五-10 執行單一資料存取以及 C&L 架構之時間比較圖(一萬個運算)

	Pure file access		**C&L Scheme**	
10000 個運算	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
1K	58.226	0.005	379.057	0.037
10K	51.357	0.005	364.730	0.036
100K	91.685	0.009	434.041	0.043
1M	395.102	0.039	977.628	0.097
10M	3431.726	0.343	5572.788	0.557

第二項 在不同網段執行的時間

再來我們呈現客戶端裝置和服務提供者在**不同網段**下運作。兩個不同的運算包含讀取及寫入是在客戶端裝置隨機的選擇，每一次運算的資料大小為 1k、10k、100k、1M、10M 以及隨機選擇上述五種檔案進行運算，表格內單位以秒計算，分別記錄在表格 五-11、表格 五-12、表格 五-13、表格 五-14、表格 五-15、表格 五-16。

表格 五-11 做資料運算所需的時間 (資料大小為 **1kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.779	0.077	0.927	0.092
100	5.758	0.057	6.161	0.061
1000	42.191	0.042	49.505	0.029
10000	425.994	0.042	479.057	0.047

表格 五-12 做資料運算所需的時間 (資料大小為 **10kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.408	0.040	0.425	0.042
100	4.003	0.040	4.154	0.041
1000	40.445	0.040	41.385	0.041
10000	459.249	0.045	464.334	0.046

表格 五-13 做資料運算所需的時間 (資料大小為 **100kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.496	0.049	0.498	0.049
100	5.295	0.052	5.165	0.051
1000	50.777	0.050	53.589	0.053
10000	533.873	0.053	574.041	0.057

表格 五-14 做資料運算所需的時間 (資料大小為 **1000kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	1.451	0.145	1.455	0.145
100	14.327	0.143	14.32	0.143
1000	148.936	0.148	142.823	0.142
10000	1547.177	0.154	1477.628	0.147

表格 五-15 做資料運算所需的時間 (資料大小為 **10000kB**)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	12.283	1.228	13.317	1.331
100	123.573	1.235	125.637	1.257
1000	1243.185	1.243	1248.548	1.248
10000	13568.196	1.356	14572.788	1.457

表格 五-16 做資料運算所需的時間 (資料大小是隨機選擇 **(1kB,10kB,100kB,1000kB,10000kB)**的檔案)

	Chain hashing Scheme		**C&L Scheme**	
運算的個數	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
10	0.212	0.021	1.393	0.013
100	2.055	0.020	3.897	0.038
1000	19.968	0.019	22.636	0.022
10000	264.731	0.026	247.623	0.024

為了瞭解 C&L scheme 所產生的負擔，我們比較在不同網段，資料存取的時間以及 C&L 執行的時間，如表格 五-17、表格 五-18 所示，對於小資料，額外負擔的比例略高，對於大資料由於傳送使得額外負擔的比例略低。

表格 五-17 執行單一資料存取以及 C&L 架構之時間比較圖(一千個運算)

	Pure file access		**C&L Scheme**	
1000 個運算	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
1K	1.863	0.001	29.505	0.029
10K	2.654	0.002	21.385	0.021
100K	10.366	0.010	33.589	0.033
1M	98.347	0.098	1477.628	0.147
10M	920.539	0.920	1248.548	1.248

表格 五-18 執行單一資料存取以及 C&L 架構之時間比較圖(一萬個運算)

	Pure file access		**C&L Scheme**	
10000 個運算	總共執行時間	每一個運算的平均時間	總共執行時間	每一個運算的平均時間
1K	91.156	0.009	479.057	0.047
10K	88.427	0.005	464.334	0.046
100K	227.634	0.022	574.041	0.057
1M	1043.346	0.104	977.628	0.097
10M	10295.178	1.029	14572.788	1.457

第六章　相關研究探討

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，導致了雲端運算的流行，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 [1]、Dropbox[2]、SugarSync [3]、SkyDrive [4]、and Box[5]，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[13]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[14][15][16][17]。

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[19]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謊充新的檔案資料，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。SUNDR[20]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[21][22][23]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。Venus[10]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，當伺服器回覆”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。

在 Microsoft Cloud Proof[11]中，使用者不僅僅能資料的完整性、寫入的循序性、以及讀取資料時，還能對第三方的人證明當違反這些性質時，責任歸屬在何方。本文作者 Popa et al.在文中提出一個實作富有安全性的雲端儲存系統架構，伴隨者簽章以及鏈結雜湊，他們的架構可以滿足不可否認性以及寫入循序性，而讀取資料的最新性則在定期的稽核資料就可以滿足其性質。然而就像是緒論所提出的問題,在遇到單一資源需要給多個使用者使用時,每個使用者都必須相互溝通，拿到所有的證據，對使用者而言會多出很多溝通上的負擔。Feng et al[24]考慮不可抵賴性協定的公平性問題，他們假設使用者和服務提供者雙方都有可能在收到對方的證據後，拒絕回覆證據，因此當發生這類問題時，必須仰賴第三方的人員來協調，然而他們所提出的架構，使用者們也需要交換訊息，增加額外的溝通。

第七章　結論

使用者以及服務提供者的雙向的不可否認性在商業的雲端儲存服務是非常關鍵的一環。雖然鏈結雜湊架構可以被應用於維持雙向的不可否認性，但是在單一帳戶下可以被許多個客戶端裝置交互的使用是不夠的。我們提出 C&L 架構，包含兩個新的參數嵌入在使用者的客戶端裝置與服務提供者交換的訊息之中:客戶端身分識別碼以及 local sequence number (LSN)。我們證明了在 C&L 架構之中不需要多個客戶端裝置交換任何訊息，並且能保證雙向的不可否認性，以及維持 CIWF 四個性質的需求。我們的實驗包含了，在鏈結雜湊架構以及 C&L 架構下，所產生的額外負擔，包含運算資源及儲存空間都幾乎相同。除了 C&L 架構，我們還提出如何建構雜湊樹以及消除累積的證據，並且進行了相關的實驗。雖然鏈結雜湊架構以及 C&L 架構都能保證雙向的不可否認性，但是這個性質的正確性只有在清除累積的證據時，對累積的證據進行驗證之時，才滿足此特性。

在第五章節提供的實驗結果證實了我們提出的架構之可行性。

雲端儲存的服務提供者可以利用在本篇文章所提出的架構，實現雙向的不可否認性，並定義在他們的服務層級協議中。這個架構在客戶端裝置以及服務提供者之間，有交換回覆訊息以及交換回覆回應訊息的負擔，一個折衷的辦法是，事先定義好某個資料夾，然後只讓以雜湊樹為基礎的 C&L 架構使用，使用者可以將重要的資料放在這個資料夾裡面，保證雙向的不可否認性。

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] Seny Kamara and Kristin Lauter. “Cryptographic Cloud Storage,” in Financial Cryptography Workshops, pp. 136-149, January 2010.

[9] 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.

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

[11] Raluca Ada Popa, Jacob R. Lorch, David Molnar, Helen J. Wang, and Li Zhuang "Enabling Security in Cloud Storage SLAs with CloudProof," in USENIX Annual Technical Conference, June 2011

[12] 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

[13] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable Secure File Sharing on Untrusted Storage," In USENIX FAST (2003).

[14] 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.

[15] 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.

[16] 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.

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

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

[19] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh, "Sirius: securing remote unstrusted storage," In NDSS (2003).

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

[21] 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.

[22] 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.

[23] 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.

[24] 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

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

附錄

(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/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/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>
  <Chain_Hash>OIxl1ZNHlKJ4dtiSgoblGH5smr8=</Chain_Hash>_
  <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/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>
   <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/20/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>
    <Chain__Hash>OIxl1ZNHlKJ4dtiSgoblGH5smr8=</Chain__Hash>
    <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/200/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>F2EoKYSzFgyHudRr95f+wBU+D7U=</DigestValue>
        <Reference>
      </SignedInfo>
    </Signature>
  </Response>
  <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…+HrMKKMr
lzoL3EZA==
     </SignatureValue>
   </Signature>
</ResponseOfResponse>
```

圖 0-1 請求、回應，回覆回應訊息在鏈結雜湊架構下執行後的範例

(A)

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

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Operation>
    <Type>WRITE</Type>
    <DataLocation>\summer\Data\data-0.txt</DataLocation>
    <Data>c3VtbWVy…</Data>
  </Operation>
  <ClientID>summer</ClientID>
  <LSN>0</LSN>
  <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/xmldsig#enveloped-signature"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

<table>
<tr><td></td><td><pre>
        <DigestValue>0psNpD5xJ9A6BQp54eku6ESgdUM=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>INxwW92YApqpEiOX0e46EJ9XVd+3…cR2VQ0FrPQ==
      </SignatureValue>
  </Signature>
</Request>
</pre></td></tr>
<tr><td></td><td>$R_i=(Q_i, \text{EpochID}, \text{GSN}, [Q_i, \text{EpochID}, \text{GSN}]_{pri(P)})$</td></tr>
<tr><td>(B)</td><td><pre>
<?xml version="1.0" encoding="UTF-8"?>
<Response>
 <Request>
  <Operation>
    <Type>WRITE</Type>
    <DataLocation>\summer\Data\data-0.txt</DataLocation>
    <Data>c3VtbWVy…</Data>
  </Operation>
  <ClientID>summer</ClientID>
  <LSN>0</LSN>
  <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/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>
  <Chain__Hash>x9B0ZlG4OnZ4EnfI1HyTgeyWRLU=</Chain__Hash>
  <Epoch>Epoch001</Epoch>
  <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/xmldsig#enveloped-signature"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>kKDOsQZXqX/ULnTDGSUmDVp2r7s=</DigestValue>
     </Reference>
     </SignedInfo>
     <SignatureValue>Fkc8HTA6Y/WNhJyCJJ1iAylHVkHfivF+3…Uq6g4FOSpg==
     </SignatureValue>
  </Signature>
</Response>
</pre></td></tr>
<tr><td>(C)</td><td>$RR_i=(R_i, [R_i]_{pri(U)})$<pre>
<?xml version="1.0" encoding="UTF-8"?>
<ResponseOfResponse>
 <Response>
    <Request>
       <Operation>
          <Type>WRITE</Type>
          <DataLocation>\summer\Data\data-0.txt</DataLocation>
          <Data>c3VtbWVy…</Data>
       </Operation>
       <ClientID>summer</ClientID>
       <LSN>0</LSN>
       <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://wwww.../xmldsig#enveloped-signature"/>
              </Transforms>
              <DigestMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <DigestValue>0psNpD5xJ9A6BQp54eku6ESgdUM=</DigestValue>
           </Reference>
           </SignedInfo>
           <SignatureValue>INxwW92YApqpEiOX0e46EJ9XVd+3…cR2VQ0FrPQ==
           </SignatureValue>
</pre></td></tr>
</table>

```
        </Signature>
      </Request>
      <Chain__Hash>x9B0ZlG4OnZ4EnfI1HyTgeyWRLU=</Chain__Hash>
      <Epoch>Epoch001</Epoch>
      <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/20/xmldsig#enveloped-signature"/>
           </Transforms>
           <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <DigestValue>kKDOsQZXqX/ULnTDGSUmDVp2r7s=</DigestValue>
         </Reference>
         </SignedInfo>
         <SignatureValue>Fkc8HTA6Y/WNhJyCJJ1iAylHVkHfivF+3…Uq6g4FOSpg==
         </SignatureValue>
      </Signature>
    </Response>
    <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/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>1T2sKeCGIBRoKY9Q405MPCAA8V8=</DigestValue>
       </Reference>
       </SignedInfo>
       <SignatureValue>dyd5yhq26hC0Ex8brTXHm16…ZA4+BX Awn34dm8dw==
       </SignatureValue>
    </Signature>
</ResponseOfResponse>
```

圖 0-2　請求、回應，回覆回應訊息在 C&L 架構下執行後的範例