具安全隱私的雲端資料去重複技術

謝谷松 [1]、游家牧 [2*]
[1, 2] 中興大學資訊科學與工程學系
[1]oudoollo@yahoo.com.tw、[2]chiamuyu@nchu.edu.tw

摘要

因為其易於實作的特性，雲端儲存 (cloud storage) 大量使用資料去重複技術 (data deduplication) 來節省其營運成本，但是資料去重複技術本身帶來許多全新的安全與隱私威脅。本文將首先介紹伴隨資料去重複技術的諸多威脅，接著敘述具安全隱私的雲端資料去重複技術，希冀能喚起研究人員對此議題的注意。

關鍵詞：雲端儲存、安全、隱私

Cloud Data Deduplication with Security and Privacy Preservation

Ku-Sung Hsieh[1*], Chia-Mu Yu [2]
[1, 2] Department of Computer Science and Engineering, NCHU
[1] oudoollo@yahoo.com.tw, [2]chiamuyu@nchu.edu.tw

Abstract

Data deduplication has been widely adopted by cloud storage provides to reduce the operation cost, including the storage, bandwidth, and management. Unfortunately, data deduplication itself incurs new security and privacy threats. This article first describes the security and privacy threats, and then has an overview of the state-of-the-art cloud data deduplication techniques with security and privacy preservation.

Keywords: Cloud storage, security, privacy

壹、背景

目前最被大眾所接受的雲端服務就屬雲端儲存 (cloud storage)莫屬了。Cloud storage 所提供的服務就是提供使用者 (user) 儲存空間讓 user 能夠備份上傳他們的資料，並且進行雲端與 user 所持有的不同裝置做資料同步。目前著名的 cloud storage 包含 Bitcasa

* 通訊作者 (Corresponding author.)

(https://www.bitcasa.com/) ， Dropbox (https://www.dropbox.com/) ， Google Drive (https://www.google.com/drive/)，Microsoft One Drive (https://onedrive.live.com/)，SpiderOak (https://spideroak.com/)，Wuala (https://www.wuala.com/)等。這些 cloud storage 的一個共同特徵就是都會給予未付費使用者 (free user) 一定程度的免費儲存空間 (free space)。譬如 Dropbox 會給予 2GB 的 free space，Google Drive 會給予 15GB 的 free space。Free user 即可隨意使用這些 free space 來儲存備份資料。但是 cloud storage 要如何精簡且有效率地儲存 user 所上傳的資料則是個大問題。這是因為除了付費使用者的資料當然得妥善保存之外，現在的 free user 往往都會註冊多個 cloud storage 的帳號來多獲取一些 free space，而雖為 free user 的資料，但是 cloud storage 仍不能讓其漏失，以上種種都加重了雲端儲存營運者 (cloud storage administrator) 的儲存成本 (storage cost)。

目前最常被使用來降低 cloud storage administrator 的 storage cost 的方法即為資料去重複技術 (data deduplication)。這 data deduplication 技術的中心思想是「不要將相同的資料重複儲存」。雖然是這樣簡單的概念，但是譬如全球網路儲存工業協會 (Storage Networking Industry Association，SNIA)的報告 [11] 即指出，使用了 data deduplication 的儲存裝置將可以省下 70~95%的儲存空間 (storage space)，而這也直接就代表著省下 70~95%的 storage cost。因此，data deduplication 對 cloud storage 來說可以說是一個非常重要且必不可缺的技術。

1.1 資料去重複技術的各種分類

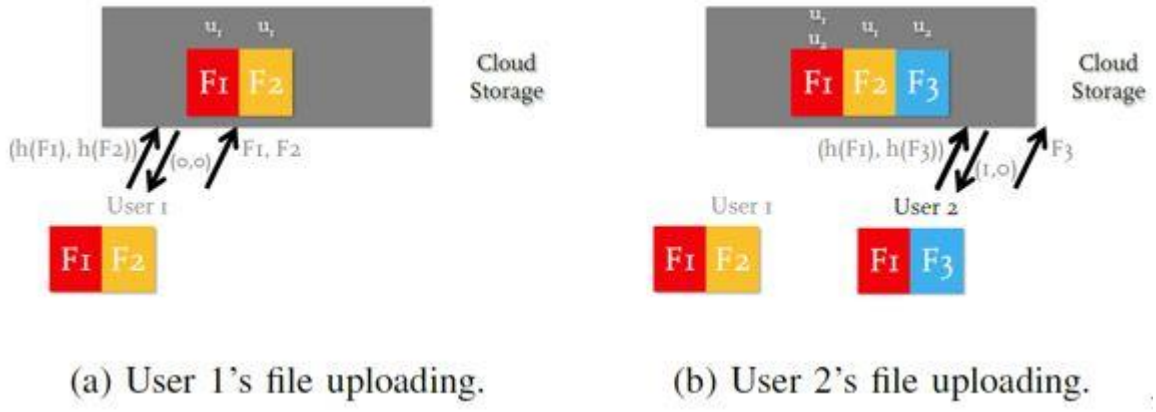
就以 data deduplication 來說，雖然其背後的原理只是「不要將相同的資料重複儲存」，但是就以實際上各種應用的 data deduplication 實作來說，則會有不同的分類方式。在此，我們考慮一個可能會是多人使用的 cloud storage 如 Dropbox，在這樣的 cloud storage 上面，首先，可以針對「是否對於不同 user 的資料來進行 data deduplication」來做分類。如果只能對個別 user account 內的資料進行 data deduplication 的話，即稱之為 single user data deduplication，而若是可以把所有 user account 內的資料一併考慮進去，統一進行 data deduplication 的話，即稱之為 crosse user data deduplication (或又稱 global data deduplication)。

另外，根據「進行 data deduplication 的最小資料單位」來區分的話，也可以區分成 file-based 與 chunk-based (或又稱 block-based) data deduplication。顧名思義，前者是以檔案為單位來進行 data deduplication; 如果任兩個檔案一模一樣的話，則只儲存第一份檔案即可。而 chunk-based data deduplication 則是會先將每個檔案切分成多個 chunk，然後以 chunk 為最小單位來進行 data deduplication。很顯然地，如果單論因為 data deduplication 所省下的 storage space saving 來說的話，chunk-based data deduplication 比 file-based data deduplication 獲得更多的 storage space saving。譬如考慮兩個 1GB 的檔案，且兩個檔案幾乎相同，若使用 file-based data deduplication 的話則無法獲得 storage space saving，而

用 chunk-based data deduplication 的話就可以獲得大部分的 storage space saving。而 chunk-based data deduplication 仍可以利用如 Rabin fingerprint [34] 之類的技術再改良成 variable-size chunk-based data deduplication 以獲得更多的 storage space saving。關於 data deduplication 的分類細項可以參照 [31]。

最後，根據「資料上傳者是否會得知上傳的資料已經有重複」來區分的話，將可以區分成 server-side 與 client-side data deduplication 兩種。就以前者來說，user 想上傳資料的話就直接進行上傳，而由 cloud storage 來決定是否儲存剛獲得上傳的資料。而以後者來說，user 在實際上傳資料之前，會先計算對應於資料的 deduplication tag，並且先將 deduplication tag 上傳至 cloud storage。Cloud storage 在經由 deduplication tag 的比對來確認是否已經存有對應的資料之後，會通知 user 是否需要再上傳資料。其中，deduplication tag 在 file-based data deduplication 實是採用完整檔案下去做如 SHA256 的 hash，而在 chunk-based data deduplication 時則是用各 chunk 下去做 hash。可以知道，因為如果 cloud storage 已經存有對應資料了，則甚至資料不用再被重複傳輸，因此，client-side data deduplication 是一個除了可以獲得 storage saving 之外，還可以獲得 bandwidth saving 的作法。

目前許多 cloud storage 為了節省 storage cost，也為了 bandwidth saving 與提昇使用者滿意度，均採用 cross-user chunk-based client-side data deduplication。譬如 Wuala 與 Dropbox 就是使用 chunk size 固定為 4MB，且 hash function 是 SHA256 的 cross-user chunk-based client-side data deduplication [10][28]。因為 chunk-based 這個特性不好被畫出來且和將討論的安全議題無關，我們先以下圖來表示 cross-user client-side data deduplication 的運作方式。首先，在圖一(a)時，user 1 想上傳 F1 與 F2 兩個檔案，首先，user 1 先算出並上傳 h(F1)與 h(F2)兩個 deduplication tag，這裡假設 cloud storage 還沒有 F1 與 F2 的複本，所以 cloud storage 傳回兩個 negative ACK (圖中以 0 表示) 通知 user 1 繼續完整上傳 F1 與 F2。接著，在圖一(b)當中，user 2 想上傳 F1 與 F3 兩個檔案，user 2 算出並上傳 h(F1)與 h(F3)兩個 deduplication tag，因 cloud storage 已有 F1，所以 cloud storage 傳回一個 negative ACK 與一個 positive ACK (圖中以 1 表示) 通知 user 2 只需完整上傳 F3 即可。這裡可以看出，雖然從 user 的觀點來看，deduplication tag 的傳輸都是多餘的，但是卻可以藉此得知資料是否已經存在 cloud storage 並獲得更多的 storage saving。



圖一: Cross-user client-side data deduplication 的運作範例

1.2 安全與隱私顧慮

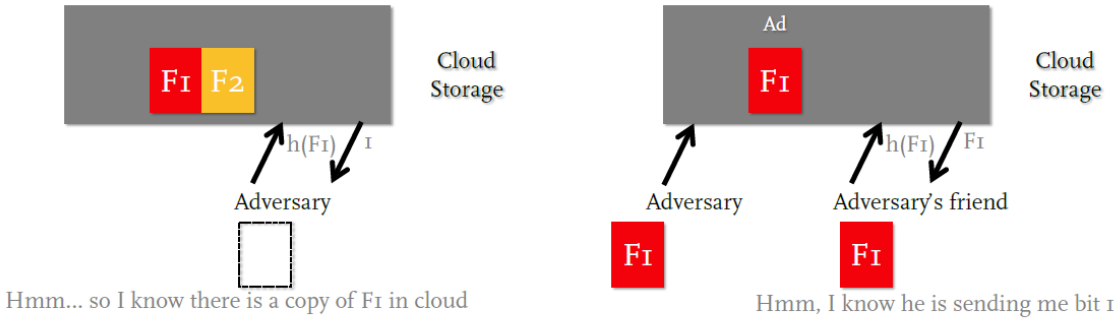
從 2010 年開始有研究指出 [18]，具有 data deuplication，尤其是 cross-user client-side data deduplication 的 cloud storage (以下簡稱 deduplicated cloud storage) 雖然可以獲得 storage 與 bandwidth saving 但是也多了安全與隱私上的顧慮。譬如從圖二(a)可以看出，攻擊者可以藉由上傳可能的 hash 以及觀察 deduplicated cloud storage 所做的回應來獲知對應的檔案是否已經存在於 deduplicated cloud storage 內，導致檔案的隱私性被破壞。而從圖二(b)可以看出 deduplicated cloud storage 會被濫用而成為一種 covert channel。更仔細一點地來說，譬如兩個攻擊者之間的通訊若是被監聽，而假設他們事前已經同意一個共同的檔案，則他們藉由上傳或是刪除那個共同檔案，換言之，經由 deduplicated cloud storage 內那個共同檔案的存在與否，則可以建立起一個 low-bandwidth 的 covert channel 來進行秘密傳輸而不被發現傳輸正在進行當中。

而從圖二(c)可以看出 deduplicated cloud storage 會被濫用而成為一種 CDN (Content Delivery Network)來協助檔案的分配傳送。這樣的情況最常出現在不特定的多人想要共同分享同一份大檔案。因為是大檔案，所以無法經由一般的 email 附件來傳送。而假設攻擊者又不想付費取得相對應的 storage 與 bandwidth 資源，所以原始的資料擁有者可以先上傳想分享的資料至 deduplicated cloud storage，然後將相對應的 deduplication tag 傳遞給其他人。此時，收到 deduplication tag 的人假裝他仍要上傳同一份資料，並且也給予 deduplicated cloud storage 那些 deduplication tag。在看到相同的 deduplication tag 之後，deduplicated cloud storage 會認為此時上傳 deduplication tag 的人也只是恰好擁有相同資料，而中斷資料的傳輸並讓此人獲得資料的擁有權。至此，此人即可順利下載資料，且以 deduplicated cloud storage 的角度來看，他只是下載自己的資料，並不算在需要付費的頻寬限制之內。這樣的攻擊並不僅只是研究人員的想像，已經有一套 open source 的

software 稱為 Dropship [22] 被開發出來，可以利用 Dropbox 的 deduplication 技術來進行這樣的 cloud storage 濫用。

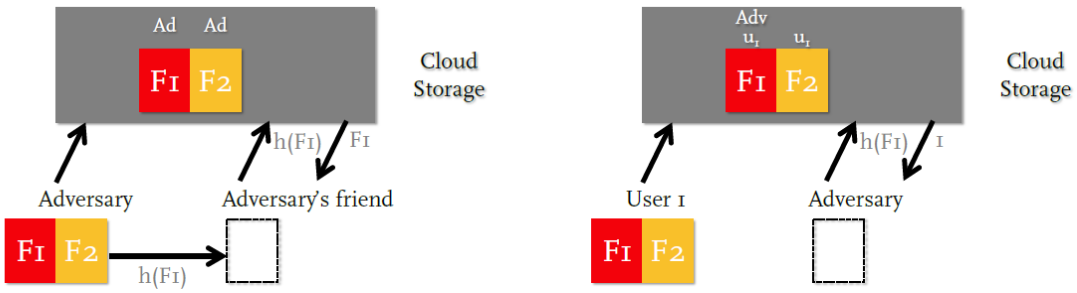
還有，從圖二(d)可以看出 deduplicated cloud storage 上的資料也有可能被攻擊者所惡意下載。這裡是說，如果攻擊者想下載某份檔案，但恰巧只擁有檔案的 deduplication tag 的話，即可利用類似上述方法，攻擊者先假裝自己要上傳檔案，藉此來獲取檔案的擁有權，之後再逕行下載即可獲得本不屬於攻擊者的檔案。

由於上述的非法下載檔案的這件威脅影響太大，另外，也怕 cloud storage administrator 會想探知上傳至 cloud storage 的資料內容，所以，一個合理的防禦手段即是在真正上傳資料之前，user 即先將資料進行加密 (encryption) 之後再上傳加密過後的資料。如此一來，即便攻擊者經由任何手段從 cloud storage 獲得資料也無法解密 (decryption) 獲得真正內容。



(a) Snooping the file existence.

(b) Deduplicated cloud storage abused as covert channel.



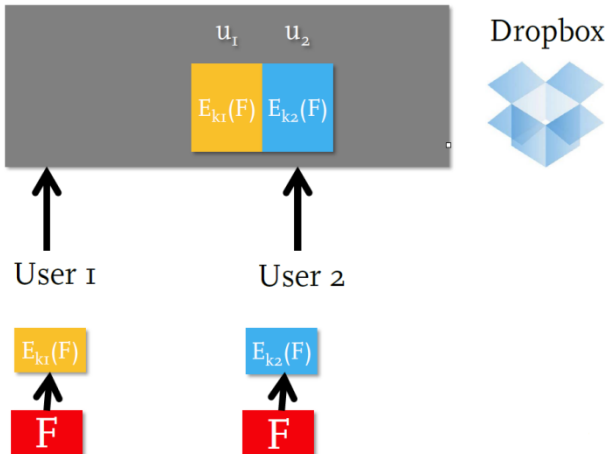
(c) Deduplicated cloud storage abused as CDN.

(d) Unauthorized file downloading.

圖二: 各種 deduplicated cloud storage 不一樣的安全與隱私攻擊

但是 user 端的加解密將會使得 cross-user client-side data deduplication 無效。可以考慮一個場景如圖三。其中，user 1 與 user 2 明明擁有同一份檔案 F，但是因為選擇的 key 幾乎不可能會相同，所以加密出來的結果不會一樣，這導致了即便使用了 cross-user

client-side data deduplication 也無法省下任何 storage space 與 bandwidth，甚至還因此多浪費了 deduplication tag 計算的時間與傳輸 deduplication tag 的 bandwidth。



圖三: user 端的加密對 cross-user client-side data deduplication 的影響

另外，除了圖二內關於 cloud storage 內檔案安全與隱私的問題之外，deduplicated cloud storage 也可能遭遇一個稱之為 poison attack 的攻擊 [4][41]。這個攻擊並非攻擊者主動地想獲得檔案的資訊，而是退一步想讓原本的檔案擁有者屆時取得錯誤的檔案，造成錯誤的判斷即可。這樣的攻擊可以用以下手法簡單完成; 首先攻擊者 A 獲知資料擁有者 B 的某檔案 F，並想讓 B 在上傳 F 並刪除 F 之後，日後獲得一個假的檔案 G。假設 deduplicated cloud storage 原本並未有 F，則 A 可以先計算並上傳 h(A)，而在 deduplicated cloud storage 要求 A 上傳整份 F 時，A 改上傳 G。此時 poison attack 就已經算完成。因為接下來的 B 也想上傳 F。但是 deduplicated cloud storage 會通知 B 不用上傳，而會直接設定是 F 的擁有者。但是因為 F 已經被 A 惡意取代成 G 了，所以日後若是 B 需要 F 而從 deduplicated cloud storage 下載 F 的話，會下載到 G，但是 B 仍渾然不知，進而可能使 B 做出錯誤的判斷。

貳、文獻評述

回顧我們在圖二所闡述的關於 deduplicated cloud storage 所遇到各種安全與隱私問題，從 2010 年開始，即開始陸續有研究成果被提出來反制安全與隱私的漏洞。譬如在 [18] 當中即提出 randomized threshold 的方法來一次解決圖二(a)與圖二(b)的問題。其 randomized threshold scheme 的基礎原理是觀察到一般使用的 data deduplication 其實都是隱含假設一個 deduplication threshold td 為 1 (或是一個其他的固定值)。也就是說，如果正要上傳的資料已經存有 td 份於 deduplicated cloud storage 內的話，則 user 無需再進行

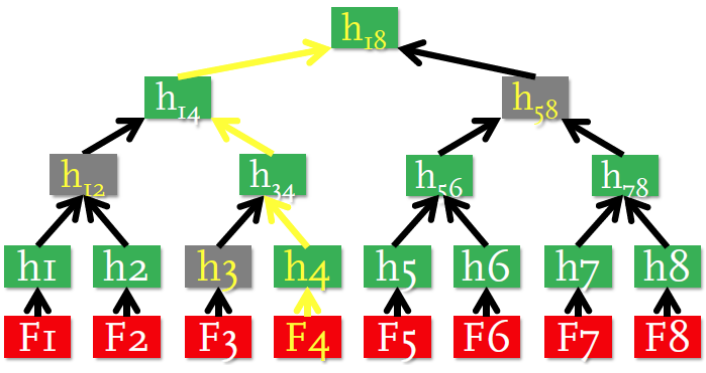
上傳。但是這項「user 無需再進行上傳」的資訊即洩漏了檔案存在與否的資訊給攻擊者，使得攻擊者可以進行圖二(a)與圖二(b)般的隱私竊取與 covert channel。但是如果每個檔案都各自有各自的 deduplication threshold 的話，則攻擊者相對來說會較難以獲取檔案存在與否的資訊。而在實務上，Wuala 有個特別的策略來防堵圖二(a)與圖二(b)的問題; 他們對數百 KB 以下的資料完全不做 data deduplication，只對大檔案做 data deduplication。Wuala 認為只對大檔案做的話可以增進效率，取得 data deduplication 的好處，但是他們認為機密內容必定都存在於小檔案內，所以小檔案完全不做 data deduplication，來避開所有 data deduplication 帶來的安全漏洞。

[28] 更特別以 Dropbox 為目標，挖掘出更多關於 data deduplication 的安全漏洞。譬如 Dropbox 對於 deduplication tag 的製作其實是向外呼叫 library 來完成，但是如果攻擊者惡意修改了 library 則會造成如圖二(a)與圖二(d)的情況。也更暴露出一些 Dropbox client software 實作上的安全漏洞，讓 user 可以不付費而獲得無限量的 Dropbox stprage space 以及非法下載其他 user 的資料。

另外，在 [17] 與 [32] 當中也提出 Proof of Ownership (POW)的概念來解決圖二(c)與圖二(d)的問題。且在 [32] 裡，幾個經過改良的 s-POW scheme 陸續被提出。觀察到圖二(c)與圖二(d)的問題主要是攻擊者都利用了「上傳同樣的 hash 之後，deduplicated cloud storage 即會不加以檢查而直接認為上傳 hash 的 user 是擁有對應的資料」的這樣弱點，s-POW 就針對這樣關於 data deduplication 的弱點加以防護; s-POW scheme 都有相同的特徵，更確切來說，在 s-POW-1 scheme 當中，一旦有 user 上傳了 deduplicated cloud storage 內已經有的 deduplication tag 之後，deduplicated cloud storage 將會在連結對應檔案與 user 之前，先隨機要求 user 回傳檔案內的幾個 bit。如果 user 能順利回答那幾個 bit 是 0 或是 1，則 deduplicated cloud storage 相信 user 真的是擁有對應資料。反之，若是 user 無法順利回答那幾個 bit 是 0 或是 1，則 deduplicated cloud storage 認為方才上傳 deduplication tag 的 user 是攻擊者，只想騙取檔案的所有權。而在 s-POW-2 與 s-POW-3 這兩套方法當中，則是觀察到 deduplication tag 的計算需要過多 I/O，但是用於 client-side data deduplication 判別是否有無資料存在於 deduplicated cloud storage 中的 deduplication tag 其實並不需要一般如 MD5 或是 SHA1 般的繁複計算量。因此，s-POW-2 與 s-POW-3 各自提出一個並非 cryptographic use 而有 collision 的 hash function，能大幅降低計算 hash 所需的時間，藉此來提升 s-POW scheme 的效能。

[32] 的 s-POW scheme 其實有個共同的缺點，也就是每次 deduplicated cloud storage 收到 deduplication tag 時,都需要藉由 disk I/O 動態地去已經儲存在 secondary storage 的檔案抓出 random bit 來詢問 user。因此，雖然 s-POW scheme 在安全度上已有大幅提昇，可是卻犧牲了 deduplicated cloud storage 端的效率。這裡得附加說一點，雖然 deduplicated cloud storage 可以藉由一次提取多個檔案的多個 random bits 來盡量減少 s-POW 所造成的 I/O delay，但是仍受限於 deduplicated cloud storage 中 memory 的大小限制，無法儲存太多 random bits，而改進不了多少效能。相比於此，在 [17] 所提出的 POW scheme 架

構則可以根本地解決這個 cloud side 的效能瓶頸。由於 [17] 的 POW scheme 的主要 idea 是根基於一個稱之為 Merkle tree 的資料結構，因此，在敘述 POW scheme 之前，我們先簡述何為 Merkle tree。Merkle tree 主要用於 authentication，且其實就是一個 binary tree，其中，所有的 leaf nodes 都是欲做 authentication 的資料的 hash，而所有 internal nodes 都是由相對應 children nodes 的 hash 串接起來再算出的一個 hash。在 Merkle tree 的架構下，只需要儲存一個 tree root 即可對所有對應於 lead nodes 的資料做 authentication。一個 Merkle tree 的範例可以如圖四所示。在圖四當中，hi 這樣的 leaf node 是由對應的資料 Fi 來算得的 hash。根據 Merkle tree 的規則，tree root h18 就等於 h(h14||h58)。如果想驗證 F4 的正確性，則驗證者除了手上握有 tree root h18 之外，若又獲得對應於 F4 的 sibling path 上的所有 hash，則可以驗證 F4 的正確性。其中，所謂的 Fi 的 sibling path 即是 Fi 與 tree root 相連的 path 上的所有 sibling nodes 的集合，譬如 F4 的 sibling path (見圖四的黃色路徑) 就包含 h3，h13，h58 (見圖四的灰色點)。可以看出，若驗證者除了手上握有 tree root h18 之外，若又獲得 h3，h13，h58 則可以藉由根據這些資訊自行計算出 tree root h18，而若計算出的 tree root h18 與本來握有的 tree root h18 不相同，則代表 F4 不正確，反之則代表 authentication 成功。在 Merkle tree 這樣的資料結構下，POW-1 scheme 即是將 user 上傳的檔案切割成 chunk，並視這些 chunk 為 Merkle tree 內的資料，並進行 Merkle tree 運算。接著，對每份檔案來說，將檔案放入 secondary storage 當中且只留 tree root 在 memory 當中。而當又有 user 上傳 deduplication tag 欲獲得檔案的擁有權，則 deduplicated cloud storage 則隨機挑多個對應於該檔案的 Merkle tree 的 leaf node 位置，並請 user 回傳給 deduplicated cloud storage 對應的 sibling path 讓 deduplicated cloud storage 可以重新自行算得 tree root。觀察可知，不同於上述的 s-POW schemes 那樣，因為 deduplicated cloud storage 從頭到尾只要保留 tree root 在 memory 即可，deduplicated cloud storage 不需要任何 I/O 即可順利檢查 user 是否真的握有宣稱的檔案。在 [17] 中，除了 POW-1 scheme 之外，也針對如何改進 POW-1 scheme 的效能來提出 POW-2 與 POW-3 schemes，不過因為基本原理都相同，故不贅述。POW scheme 有被試著實現在 OpenStack Swift 當中來檢視其效率 [21]。

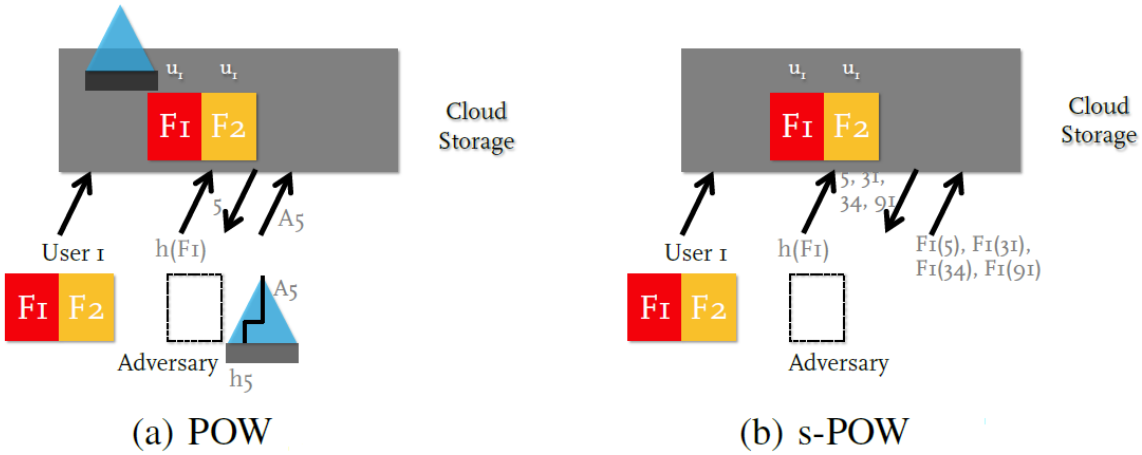


圖四: 一個 Merkle tree 的圖示範例

POW 與 s-POW schemes 的運作可以從圖五當中來看。譬如 POW scheme (見圖五 a)，就如上述般，假設有 user 1 已上傳 F1 與 F2。接著又有一個攻擊者欲藉由上傳 h(F1) 來取得 F1 的擁有權，但是 deduplicated cloud storage 選擇詢問對應於 F1 的第 5 條 sibling path A5，因攻擊者沒有 F1 無法得到 A5，所以 deduplicated cloud storage 無法得出與存於 memory 內一樣的 tree root，代表此時上傳 h(F1)的 user 是攻擊者，不可授與 F1 的擁有權。又譬如 s-POW scheme (見圖五 b)，也假設有 user 1 已上傳 F1 與 F2。接著又有一個攻擊者欲藉由上傳 h(F1)來取得 F1 的擁有權，但是 deduplicated cloud storage 則選擇詢問對應於 F1 的第 5，31，34，91 個 bits 是 0 或 1。因攻擊者沒有 F1 無法得知第 5，31，34，91 個 bits 是 0 或 1，所以只能亂回答，導致很容易讓 deduplicated cloud storage 發現有答案不一致之處，因而發現此時上傳 h(F1)的 user 是攻擊者，不可授與 F1 的擁有權。

上述的 POW scheme [17]其實可以被改良成 privacy-preserving POW。在 [42] 當中，privacy-preserving POW 即被設計出來，而所使用的方法是利用現有的 randomness extractor [30][40] 加上 Proof of Retrievability (POR) [20] 來達成的。

在之前的圖三所遇到的問題，也就是 user 先行對要上傳至 deduplicated cloud storage 的資料加密會導致 cross-user data deduplication 失效的這個問題上，也在近幾年陸續獲得關注並且提出可能的解決方案。其中，雖然考慮的並非是現行 cloud storage 的問題，但是最被常被拿出來視為一種可能的解決方案是在 [9] 當中被提出的 convergent encryption (CE)。所謂的 CE 並非是一種新形態的加解密演算法，而只是在說明加密時是用欲加密的明文來進行 hash 並用此 hash 來當做加密用的 key。實際上的加解密演算法可以是 AES 或 DES 都無所謂。當 CE 被 deduplicated cloud storage 使用時，加解密與 data deduplication 將可以並存而互不妨礙。



圖五: POW scheme 與 s-POW scheme [32] 的圖示範例

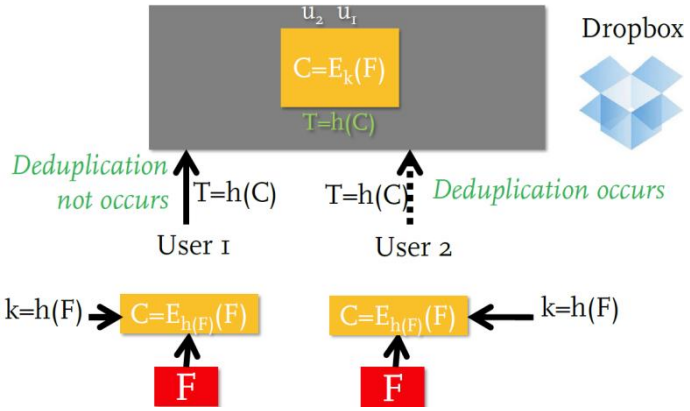
圖六以一個簡單的例子來說明 CE 如何同時達成加解密與 cross-user data

deduplication。在圖六當中，user 1 與 user 2 是互不認識且無法互相溝通，但是恰好都要上傳 F。若是按照一般的加解密方法，且讓 user 1 與 user 2 自由選擇 key 的話，就如圖三所表示，兩人的 F 會被加密成不同的密文導致無法被 deduplicate。但是利用 CE 的話，user 1 會使用計算 h(F)並利用 h(F)當做加密 F 的 key。最後，假設一開始 deduplicated cloud storage 內沒有 F 的話，user 1 就根據一般 deduplicated cloud storage 的規則來將 C=Eh(F)(F)的 hash 與 C 本身上傳。接著，user 2 雖然沒有與 user 1 有任何互動，但是因為欲上傳的資料一模一樣都是 F，所以加密用的 key 也都會相同是 h(F)並得到一模一樣且可以讓 deduplicated cloud storage 進行 deduplicate 的密文。在這樣的過程當中，首先可以觀察到，data deduplication 的能力不會因為加解密而被破壞。另外，也可以知道除了真正擁有 F 的人之外，也無人能獲得 h(F)並解密得到 F，這確保了資料的機密性。一些整合了 CE 的 backup system 的例子可以參考 [2][39]。

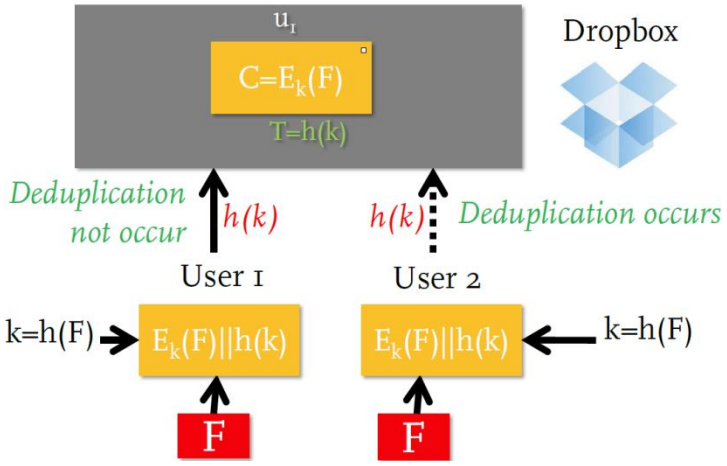
CE 可以與 file popularity 做搭配來改善效率與安全性 [38]; 譬如設定了一個 popularity threshold，有超過的檔案才需要特別做 encryption，否則則不需要。但是仔細觀察 CE 的運作過程之後，可以發現兩個 CE 共通缺點 [4]; 首先是他的效率通常不好，需要 3 個 pass 才能完成運算。這裡要注意一下，通常我們說「一個 pass」的意思是指「需要將 F 或是約略等量的資料從 secondary storage 載入到 memory 當中進行運算」。在這種定義之下，CE 的 3 個 pass 來自於 (1) 計算要加密的 key h(F)時，因為 hash 計算需要用到 F 的每個 bit，所以算是一個 pass，(2) 計算 deduplication tag Eh(F)(F)時，因為加密計算需要用到 F 的每個 bit，所以算是一個 pass，(3) 計算 h(Eh(F)(F))時，因為 hash 計算需要用到 Eh(F)(F)的每個 bit，且 Eh(F)(F)的長度與 F 大多數情況下一樣或是差距不大，所以算是一個 pass。由於 3 個 pass 都是要大量的 I/O 與計算，因此需要 3-pass 的 CE 的效率仍有改善空間。CE 的第二個缺點則是會有如前所述 poison attack 的問題。在 [4] 當中發現，因為利用 CE 所執行的 data deduplication 是直接用 C=Eh(F)(F)的 deduplication tag h(C)來讓 deduplicated cloud storage 可以知道檔案是否已經存在，所以只要 deduplicated cloud storage 在收到 C 之後，可以再自己檢查是否收到的 C 可以算出與之前收到的 h(C)一模一樣的值，即可完全避免 poison attack。在[4] 當中，這種可以完全避免 poison attack 的性質稱之為 strong tag consistency (STC)。

而在 [4] 也陸續提出多個藉由減少 pass 數來增進效率的 CE。首先是 Hash-and-CE-1 (HCE1) 被提出。HCE1 的運作可以從圖七看出; 主要是 user 除了模仿 CE 繼續用 k=h(F)當做 key 來加密 F 之外，還用 deduplication tag h(k)=h(h(F))來讓 deduplicated cloud storage 可以知道檔案是否已經存在。HCE1 的效率已經獲得很大改進; 可以看出除了 h(F)與 $C=E_{h(F)}(F)$ 的運算可能導致 I/O 之外，因為 k=h(F)是一個很短的 bit string，所以 deduplication tag h(k)的計算非常簡單。簡言之，HCE1 降到只有 2-pass。但是 HCE1 卻有完全不能抵抗 poison attack 的缺點。主要是因為 h(k)已經與 $E_{h(F)}(F)$ 的內容脫勾，所以其實攻擊者仍是可以做譬如 $E_{h(F)}(F')||h(k)$ 這樣的上傳，導致資料 F 被置換成另一份完全不相干的資料 F'。因此，HCE1 並非是一個適用於 deduplicated cloud storage 的加解密方

式。

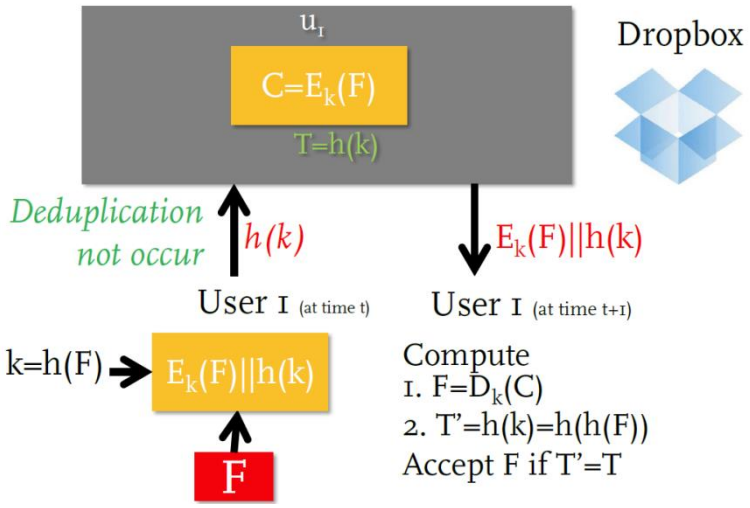


圖六: Convergent encryption (CE) 的圖示範例



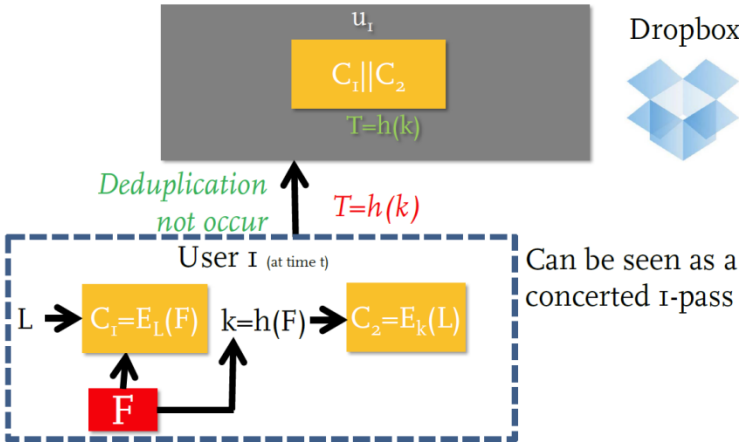
圖七: Hash-and-CE-1 (HCE1) 的圖示範例

接著，HCE1 被改良成 Hash-and-CE-2 (HCE2)。其實 HCE2 與原來的 HCE1 沒有任何改變，但是 HCE2 規定下載資料的 user 會多進行一些檢查，藉此來避免遭受 poison attack。但是照 HCE1 的規則來說，poison attack 是一定可以被攻擊者所啟動的，因此，HCE2 的檢查只能被動地保證「下載資料的 user 不會被遭受 poisoned data 所矇騙」但是無法保證「poison attack 不會發生」。我們可以藉由圖八來看看 HCE2 如何運作。在資料上傳那部分，因為與 HCE1 完全一樣，所以不再贅述。在資料下載那部分，則可以看出當 user 下載資料之後，將會先對資料做解密取回 F。而照著這樣過程所獲取的 F 理當要可以算出一個 h(h(F))，且這個計算出來的 h(h(F))理當要與下載回來的 h(h(F))一模一樣才是。HCE2 就是利用這樣方式來保證「下載資料的 user 絕對不會被遭受 poisoned data 所矇騙」。但是相比於 strong tag consistency 所保證的「poison attack 不會發生」，[4] 稱這樣的弱化保證為 tag consistency。



圖八: Hash-and-CE-2 (HCE2) 的圖示範例

但是 [4] 又繼續設計了 randomized CE (RCE)，在維持具有 tag consistency 的前提之下又再繼續增進效率。RCE 的運行過程可看圖九。其中，user 先憑空產生一個 random string 稱之為 L。利用 L 當做 key 來加密 F 得到 $C_1=E_L(F)$。同時間，user 也算出 k=h(F) 然後利用 k 對剛剛的 L 進行加密，獲得 $C2=E_k(L)$。之後，對於如何讓 deduplicated cloud storage 知道檔案是否已經存在仍是如同 HCE1 與 HCE2 一般採用上傳 deduplication tag h(k)來達成。而若 data deduplication 沒有發生的話，則將 C_1 與 C_2 一起上傳，deduplicated cloud storage 則一次保留 h(k)，C_1，C_2。而利用這種迂迴的方式來進行加密與 data deduplication 的好處是可以得到 concerted 1-pass。可以從圖九看出，雖然 $E_L(F)$的計算與 h(F)的計算都需要大量的 I/O，但是因為彼此沒有先後順序 (可看圖八中，k 與 $E_k(F)$的計算有明顯的先後順序)，如果 user 可以進行平行計算的話，其實可以同時完成。如此一來，雖然並非真正的 1-pass，而是所謂的 concerted 1-pass (若有平行計算能力的話才可 1-pass)，但是仍在特殊情況下獲得效率改進。



圖九: Randomized convergent encryption (RCE) 的圖示範例

CE，HCE1，HCE2，RCE 在 [4] 中統稱為 message-locked encryption (MLE)。針對 MLE 抵擋 poison attack 的能力以及他們的效率來比較的話，我們可以得到如圖十那樣的表格。可以看出，雖然 RCE 的設計較 CE，HCE1，HCE2 複雜，但是取得了對抵擋 poison attack 的能力以及效率的相對平衡。

	Poison attack	Performance
CE	STC	3-pass
HCE1	X	2-pass
HCE2	TC	2-pass
RCE	TC	Concerted 1-pass

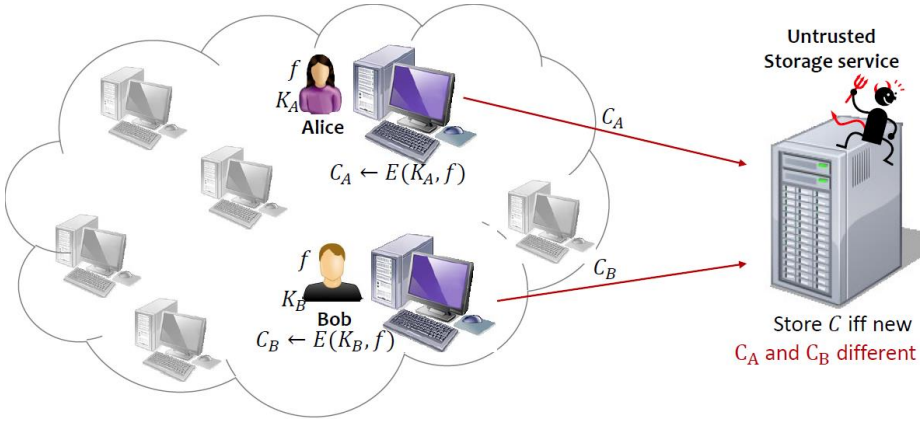
圖十: CE，HCE1，HCE2，RCE 之間的比較表

但是無論是哪種 MLE，都算是一個特殊的針對保留 data deduplication 能力的加解密方法而已，但是現實的應用需要一整套系統來防禦攻擊者。為何這麼說的原因是因為不論是哪種 MLE 都會遭受 brute force attack。也就是說在部分欲加密的資料已經洩漏，而只具有 low min-entropy 的時候[17][5] (min-entropy 是評估資料中資訊多寡的一種方法，在此可被視作攻擊者仍未知的部份)，brute force attack 讓明文與 key 掛勾的所有 CE 類型的加解密 run 過所有可能的明文 pattern 即可獲知真正的欲保護的明文資料為何。這裡需要註明的是在正常加密中，明文即便椅被知道大部分 (low min-entropy)，密文仍可依靠即多可能性的 key 來保持安全性，但是 CE 則因幾無此特性。因此，我們需要的是一個真正可以拿來討論的現實環境以及真正可以 implementable 的系統來評估 deduplicated cloud storage 與加解密的關係。在 MLE 被 [4] 發表之後，[1] 隨即將 MLE 加以改良; 在 [1] 中，明文的 distribution 也被考慮進去，並且最重要的是相比於在 [4] 的 MLE 的 key 是 deterministically 被決定，[1] 內 MLE 的 key 可以是 fully randomized，這明顯地加強了安全性，但是因為其設計，[1] 內的 MLE 明顯效率較差。

另外，無論是哪種 MLE，user 在上傳資料之後，對於每一份檔案來說，都必須保留對應的 hash (也就是加解密用的 key)在 user 端的機器上。否則，之後再下載資料回來時，將會無法對密文做解密。雖然 key 的長度不長，但是一旦備份的檔案一多，管理這些 key 也造成一種負擔，因此在 [23] 當中則提出一套針對 CE 的 key management 的方法。

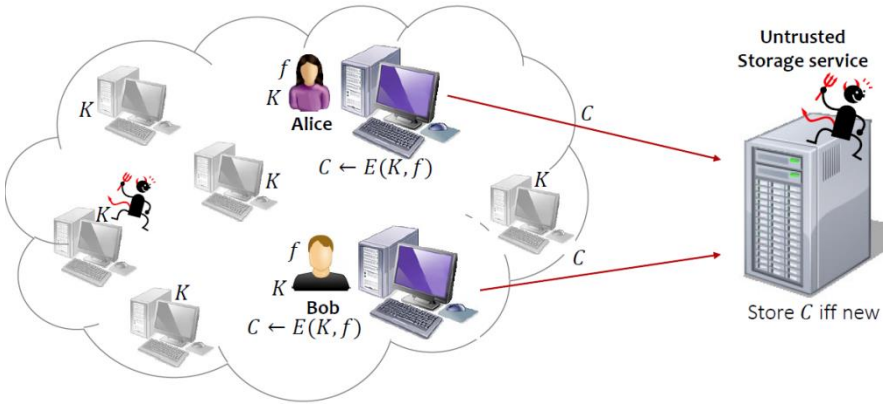
在 [5] 當中，一套這樣的系統被提出來稱之為 DupLESS。DupLESS 考量的環境是在一個類似大型辦公環境當中，且每台電腦都有需求將資料上傳至 deduplicated cloud storage 或是從 deduplicated cloud storage 下載資料。其中，所上載的每筆資料都希望經由加密而不暴露給任何人。從 deduplicated cloud storage 的角度來看，也不能因為 user 使用了加密而導致 data deduplication 失效。另外，在這個環境當中，有些電腦會被攻擊者實體或是 malware 入侵，使得裡面的安全機制不再有效而能取得該台電腦的任何資料，DupLESS 也會考慮這樣的情況來設計對應的安全機制。

圖十一就如同圖三所要表示的一樣，DupLESS 面對的環境與安全的要求非常嚴苛，所以如果只是單純地讓各個 user 獨立地產生 key 來對要上傳的資料加密的話，則明明相同的 F，因為加密的緣故，將無法被 deduplicate。



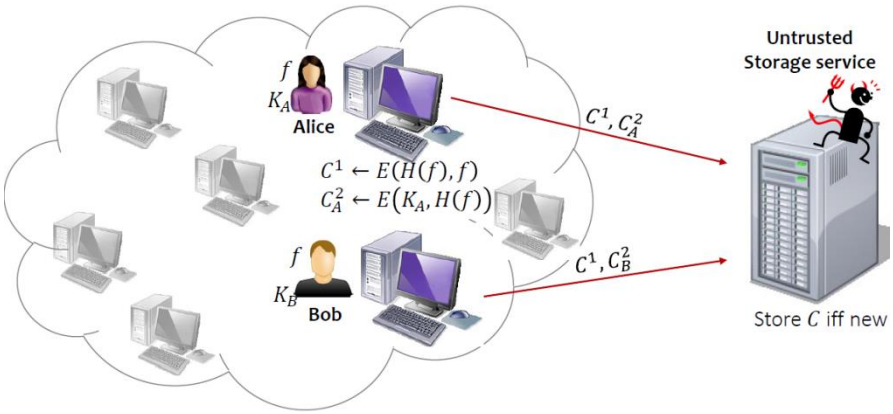
圖十一: user 端的加密雖然保護了資料免於被惡意的 cloud storage 所得知，但是也使得 data deduplication 無法順利進行，其中 $E_A(B)$與 E(A,B)皆代表用 A 當 key 加密 B

在現實世界的大型辦公環境當中，也許可以藉由該環境的網路管理員強制設定每台電腦所要使用的 key 都是相同的，這就避開了 user 端的加密而導致 data deduplication 失效的問題。但是就如圖十二所示，這樣的缺點是如果有任何一台 user 的電腦被 compromise 的話，所有電腦共同使用的那把 key 將被暴露給攻擊者知道，也就完全失去了資料的隱私性。雖然這樣的安全性不高，但是搭配著名的 TTTD algorithm [13] 與假設 deduplication information 都儲存在 user 端 [35]，卻能獲得良好的 deduplication gain。



圖十二: 所有 user 共用同一把 key，這將無法達到 compromise resilience 的要求，其中 $E_A(B)$與 E(A,B)皆代表用 A 當 key 加密 B

唯一可能的解決方法是讓每個 user 都獨立選擇 key 來對資料做加密。DupLESS 選擇 key 來做加密的方式與 [5] 的 RCE 有點類似。可以看圖十三來得知 DupLESS 的想法。DupLESS 的 user 首先利用 CE 來對資料做加密得到 c^1。之後 user 又隨意挑選一個 random string 來為 CE 所使用的 key 做加密，得到 c^2_A。最後 user 將 c^1 與 c^2_A 都上傳至 deduplicated cloud storage。這樣做的好處是可以讓 data deduplication 與資料加密並存，但是又不像單純的 CE 一般，需要為每個上傳過的資料的 hash 都保留下來解密。對於圖 13 這樣的機制來說，每個 user 都只需要保留單獨一把 key 即可解密任何一個加密過的資料，但是又因為解密的過程並非對加密的資料直接解密，而是迂迴地透過先解密出 CE 使用的 key，再行利用 CE 的方式解出所需的資料，所以安全度得以維持。可看出這簡化了上述使用 CE 還得搭配特殊 key management [23] 的複雜管理機制。可以注意到，c^2_A 的這份密文其實是無法被 deduplicate 的，但是因為對一把 key 做加密之後的密文多數也只有數百 bit，所以雖然無法被 deduplicate，但是對 deduplicated cloud storage 來講其實影響不大。而圖十三的這個方法卻也仍承襲了 CE 的多數缺點。譬如最重要，c^1 其實就是 CE 的密文，但是如上所述，在明文對攻擊者只有 low min-entropy 的情況下，即便攻擊者只攔截到密文也會讓攻擊者可以利用很短時間內可完成的 brute force attack 來取得明文。



圖十三: DupLESS 的基礎 idea，其中 $E_A(B)$與 E(A,B)皆代表用 A 當 key 加密 B

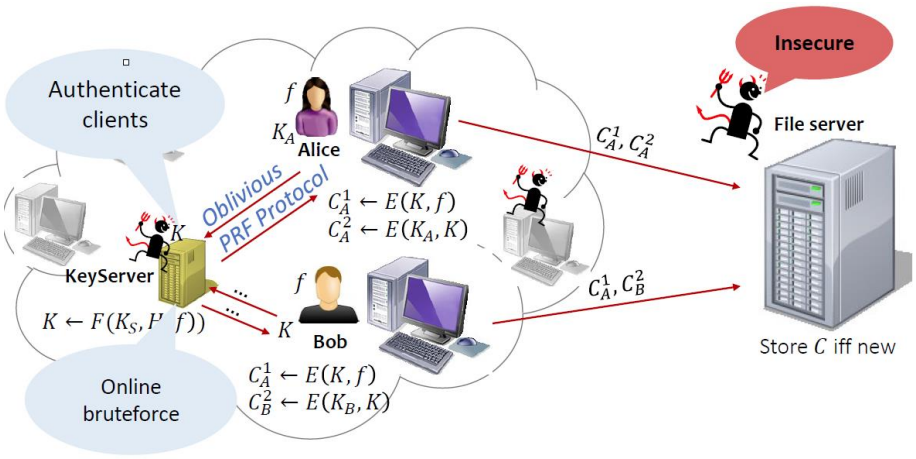
有鑑於上述問題，DupLESS [5]的架構被提出來。在 DupLESS 中，一個額外增加的，執行固定演算法 (deterministic algorithm) 的 key server (KS)被加入環境當中。這個 KS 的用途即在於若是有 user 向其利用明文的 file hash 來詢問，則 KS 回答一個固定的 key 給詢問的 user。因為 KS 是執行 deterministic algorithm，所以相同的 file hash 將導致相同的 key，因此，即便不同的 user 對 KS 做查詢也會拿到相同的 key，也因此，利用所獲得的 key 來加密的話，會獲得相同的可以被 deduplicate 的密文。在假設 KS 端的演算法設計得當的情況下，也因為從 KS 所獲得的 key 與想加密的資料其實是不相關的，所以完全地避免掉 CE 的密文會在 low min-entropy plaintext 時是不安全的問題。但是仔細檢驗這

樣的架構之後可以發現多個問題: (1) 若是對 KS 一直不斷地查詢 key，則也會讓 KS 負載過重或甚至是查詢到對應的 key，(2) 在向 KS 查詢時，明文的 file hash 被送出，而其實 file hash 在 low min-entropy plaintext 時也是不安全。針對以上兩個問題，首先，前者的問題易於解決，在 [5] 中的解決方法是利用對 KS 限制流量與詢問速率來解決。其背後的原理是正常 user 並不會短期內發送極大量要求 key 的查詢，而若收到如此多的查詢則代表是攻擊者，可以安心與其斷開連線。而後者的問題 [5] 則採用了 Oblivious PseudoRandom Function (OPRF) [29] 來解決。OPRF 主要是讓兩個互不信任的 user 能合力在不暴露出各自的秘密之下計算出一個 random value。而 OPRF 這項密碼學工具恰好可以解決 DupLESS 所面臨的問題。因為譬如在圖十四裡，user 與 KS 可以視為互不信任; user 不想把 file hash 讓 KS 得知，但是 KS 又得根據 file hash 來產生對應的 key。因此，OPRF 可以為上述問題解套。我們再從圖十四來看一下 DupLESS 的詳細過程; 相比於圖十三裡 user 仍是用 file hash 對資料做加密，這裡 user 先利用 OPRF 向 KS 索取一把 key 並對資料加密，然後利用 user 自己獨一無二的 individual key 對 KS 給予的 key 做加密後，一次將兩份密文上傳至 deduplicated cloud storage。

可以從圖十四的下半部來看看 DupLESS 對各項攻擊的容忍程度。首先，若是外部攻擊，也就是攻擊者在只能獲取密文的情況下，因為 DupLESS 使用的並非 CE，無法如 CE 般有 low min-entropy 的 case 可以有效率地突破，所以攻擊者只能硬破如 DES/AES 之類的加密法，近乎不可能。而若是有些 user 的機器被 compromise，則攻擊者可以一直利用這台被 compromised 的機器一直對 KS 做查詢，但是因為限速的關係，攻擊者只能問為數不多的問題，幾乎無法正確猜中 key。最後，又若是 KS 被 compromise 呢? 因為 DupLESS 執行的是 OPRF，所以 KS 也無法獲得 file hash 或其他資訊，所以 KS 無法對攻擊者帶來任何新資訊。在 [5] 中，DupLESS 的安全性其實並未被嚴格證明，但是在 [12] 中，DupLESS 的安全性已經被嚴格證明，且提出一個利用如 P2P 之類的分散式架構來取代單一台的 KS，希望能避免單一 KS 的 single point of failure 問題。

其他也有研究來同時結合 encryption，compression 與 data deduplication [3]。一個被稱之為 Pangolin 的系統被開發出來 [15]，號稱他的 data reduction 技術除了 data deduplication 之外，可以同時考量欲上傳的檔案格式以及不同 compression 與 data deduplication 的先後以及搭配關係。[19] 有嘗試要將 POW，encryption，data deduplication 三者結合，但是用的方法是 bilinear map 相當沒效率之外，也對環境設定加上諸多假設，因此對合併使用幫助不大。針對不同使用者有不一樣的 data deduplication 需求，也有相對應的 secure data deduplication 被開發出來 [24]。在 [25] 中則是利用 proxy re-encryption 的技巧來獲得 policy-based data deduplication。在 [27] 中，一個稱之為 deduplication proxy 的角色被引入，user 與 deduplication proxy 之間的溝通只會有 intra-user 的 data deduplication 而 deduplication proxy 與 deduplicated cloud storage 之間則會有 inter-user (也就是 cross-user) 的 data deduplication。[26] 特別考慮 secure data deduplication 用在 mobile device 上的情境。[37][43][44] 分別對 privacy-preserving data deduplication 與 POW 在一

些限制的假設下，做了嚴格的安全性證明。[48] 則是嘗試將 proof of storage 與 data deduplication 相結合。而相比於利用 MLE 與 KS 來調和 data deduplication 與加解密，在 [45] 中，equality predicate encryption 則被提出，用來當做另一個調和 data deduplication 與加解密的想法。最後，在 [36] 中，proof of retrievability (PoR) 與 proof of ownership (POW) 被放在同一個架構下來共同設計。



Attack type	Reason for security	Best attack
External attacks	Authenticating clients	Break encryption **(very hard)**
Client compromise	KS interaction overhead	Online brute-force **(slow)**
KeyServer compromise	Obliviously evaluating F	Brute-force attacks

圖十四: DupLESS 運行詳細過程

參、現有問題

關於 deduplicated cloud storage 上安全議題的研究其實是於 2012 年 [18] 開始，這幾年已經陸續有更多 deduplicated cloud storage 上的安全問題與解決方案被提出，但是我們仍觀察到有以下問題待解決：

1. POW 的效率不彰。就以目前的方法，也就是 POW scheme [17] 與 s-POW scheme [32]來說，可以被歸類為 POW scheme 的 communication overhead 嚴重大增 (因為每次對 user 的資料上傳都需 user 上傳至少一條 Merkle tree 的 sibling path 來做驗證) 而 s-POW 的 cloud-side overhead 很糟糕 (因為 deduplicated cloud storage 的 memory 承載太多欲查詢的 bits)。因此，亟需設計一個能夠同時對 user 端的 overhead 以及 deduplicated cloud storage 端的 overhead 做 best balance 的方法。
2. Privacy-preserving data deduplication 與 client-side data deduplication 的方法與目標其

實基本上是相違背的。譬如以 CE [9] 與 DupLESS [5] 的例子來說，直接使用 CE 或是暴露待上傳資料的 file hash 給攻擊者的話都會讓攻擊者在 low min-entropy 的 case 時有機會進行有效率的 brute force attack。但是因為要進行 client-side data deduplication，user 勢必要上傳 file hash 至 deduplicated cloud storage，這也直接導致了 brute force attack。一個直覺的解決方法是結合 DupLESS，使 user 是在與 KS 溝通並取得 key 與加密之後，才針對加密後的資料來做 client-side data deduplication。但是因為這樣就讓 user 得先花費不少的 communication overhead (與 KS 通訊) 與 computation overhead (計算 OPRF 與 AES) 就會有可能讓攻擊者對 user 進行 DoS attack。因此，亟需設計一個具有容忍 DoS attack 能力的 privacy-preserving client-side data deduplication 機制。

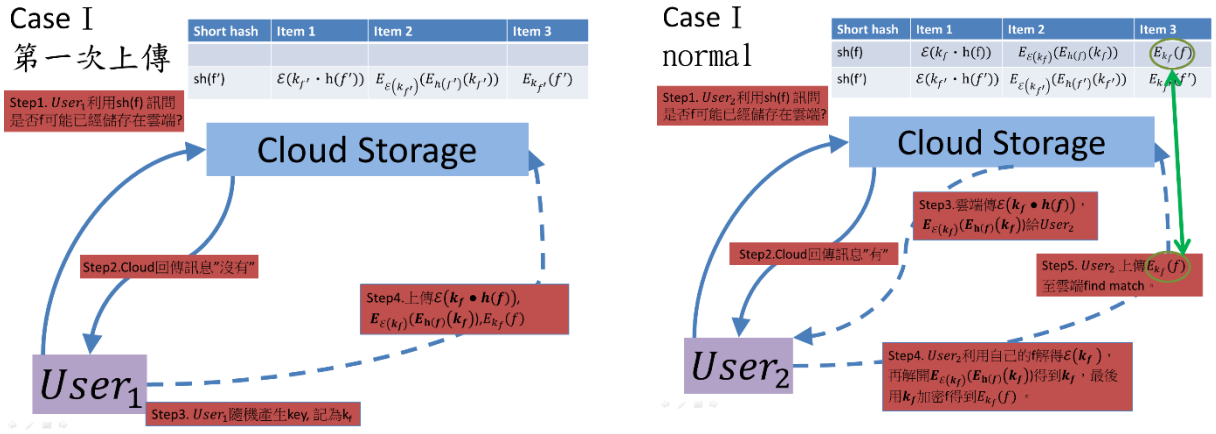
3. 雖然 DupLESS [5] 引入了 KS 來讓不同 user 之間可以對共同的資料獲得同樣的 key，但是 KS 的運作仍是個問題。由於 DupLESS 限制了使用環境是在大型辦公環境下，在這類環境下，通常會有網路管理員來為各 user 做服務。因此，KS 可以由環境的網路管理員來架設，供 user 使用。但是考慮一般大眾使用的 Dropbox 或是 Google Drive 則並沒有此類網路管理員。而 KS 的用途特殊，也不會有如 Verisign 之類的公司來做商業服務，因此唯一可能成立 KS 的單位即為 deduplicated cloud storage 本身。但是 KS 與 deduplicated cloud storage 為同一個單位來維持的話，即便 KS 是採取 OPRF，也可能因為 deduplicated cloud storage 能額外獲得極大量 KS 所不能獲得的 file hash 而可能破壞 OPRF 對於安全的保證。因此，重新探討 KS 是否能由 deduplicated cloud storage 來營運是個重要的議題。

肆、解決方案

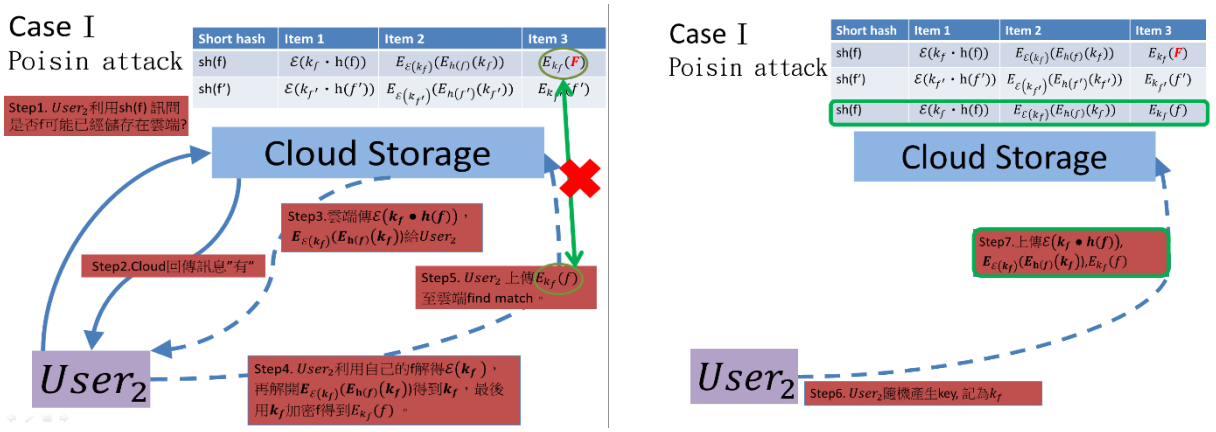
我們利用 homomorphic encryption 加上 AES 來實現能夠抵擋 poison attack 的雲端儲存 protocol，利用的是 homomorphic encryption 擁有在密文做運算等於在明文做相同運算再加密的結果之特性，我們所使用的是 ElGamal，在 cyclic group G of order q with generator g, 若 public key 為(G,q,g,h), 其中 h=g^x, x 是 serect key, 則加密 m 我們記為$\mathcal{E}$(m)，且 $\mathcal{E}(\mathrm{m}) = \left(g^r, \mathrm{m} \cdot h^r\right)$，random r ∈{0,1,……q-1}，ElGamal 的特性是$\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = \left(g^{r_1}, m_1 \cdot h^{r_1}\right)\left(g^{r_2}, m_2 \cdot h^{r_2}\right) = (g^{r_1+r_2}, (m_1 \cdot m_1)h^{r_1+r_2}) = \mathcal{E}(m_1 \cdot m_2)$。使用 k_f 進行 AES 加密 f 我們記為$E_{k_f}(f)$。

CASE Ⅰ.在第一次上傳檔案 f 的時候，利用 shorthash 可以簡單達成混淆的動作，因為 shorthash 的重複率較高，若 attacker 得到 sh(f)的話無法輕易得知是哪一個檔案的 shorthash，若單純使用 hash 的話，因為 hash 的碰撞很少，所以 attacker 可以利用 brute force attack 來得知 f。第一位使用者先利用 short hash 來做初步判斷 f 是否可能已經儲存在雲端上。如果沒有的話，表示雲端沒有儲存 f，雲端會傳訊息“沒有”給第一位使

用者，接著第一位使用者隨機產生 key，記為k_f，然後上傳$\mathcal{E}(k_f$ • h(f)), $E_{\mathcal{E}(k_f)}(E_{h(f)}(k_f))$, $E_{k_f}(f)$至雲端。當第二位使用者也想要上傳相同檔案 f 的時候，一樣會先利用 shorthash 來做初步判斷，此時雲端就會回傳訊息“有”給第二位使用者，接著雲端傳$\mathcal{E}(k_f$ • h(f)), $E_{\mathcal{E}(k_f)}(E_{h(f)}(k_f))$給第二位使用者，第二位使用者可以利用自己擁有的 f 來計算 $\mathcal{E}$(h(f))，再利用$\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = \mathcal{E}(m_1 \cdot m_2)$的特性得到$\mathcal{E}(k_f)$，得到$\mathcal{E}(k_f)$後解密$E_{\mathcal{E}(k_f)}(E_{h(f)}(k_f))$得到$E_{h(f)}(k_f)$，再用 h(f)解密得到$k_f$，之後第二位使用者利用解密所得到的$k_f$加密自己的 f 上傳到雲端進行 find match，若 find match 則表示雲端確定有 f 且沒有被 poison attack，若無法 find match 則可能雲端上的 f 已經被改動過了，此時則回到第一次上傳的後半步驟繼續進行，當成雲端沒有儲存 f，第二位使用者隨機產生k_f，然後上傳$\mathcal{E}(k_f$ • h(f)), $E_{\mathcal{E}(k_f)}(E_{h(f)}(k_f))$, $E_{k_f}(f)$至雲端。



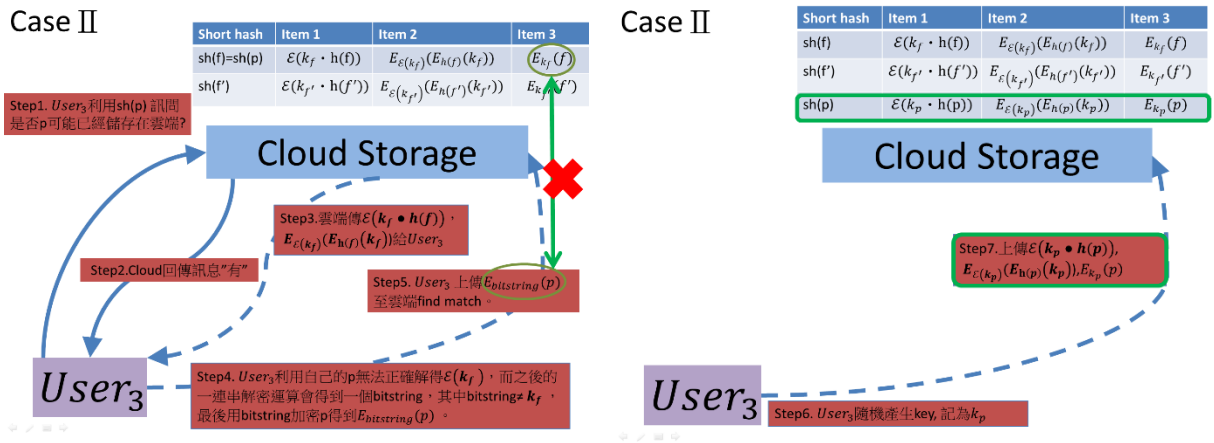
圖十五: Case Ⅰ:normal 運作圖示



圖十六: Case Ⅰ:exists poison attack 運作圖示

CASE Ⅱ.第三位使用者想上傳檔案 p，且恰巧 sh(p)=sh(f)，此時第三位使用者利用 sh(p)詢問雲端時會得到訊息“有”，但是其實雲端上並沒有儲存檔案 p 的，接下來雲端還是

會傳$\mathcal{E}(k_f \cdot \mathrm{h(f)}), E_{\mathcal{E}(k_f)}(E_{h(f)}(k_f))$給第三位使用者，但此時因為第三位使用者擁有的檔案為 p，所以無法得到正確的$\mathcal{E}(k_f)$，所以之後一連串的解密運算結果會是一團亂的 bitstring，其中 bitstring$\neq k_f$，接著第三位使用者使用 bitstring 加密 p 後上傳至雲端 find match，雲端無法 find match，表示雲端沒有儲存檔案 p，所以也是回到第一次上傳的後半步驟繼續進行。



圖十七: Case Ⅱ : 運作圖示

伍、結論

本文描述了 cloud storage 的 data deduplication 所帶來的各項 overhead saving 以及跟隨而來的各項 security 與 privacy threat。本文主要論述的是 POW 機制與 secure data deduplication 機制的設計；我們進行了大量的文獻回顧，並且最後提出一套新的 secure data deduplication 機制。

參考文獻

[1] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan and G. Segev, "Message-locked encryption for lock-dependent messages," *International Cryptology Conference (CRYPTO)*, 2013.

[2] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," *USENIX Large Installation System Administration Conference (LISA)*, 2010.

[3] N. Baracaldo, Androulaki, J. Glider and A. Sorniotti, "Reconciling end-to-end confidentiality and data reduction in cloud storage," *ACM Workshop on Cloud*

Computing Security (CCSW), 2014.

[4] M. Bellare, S. Keelveedhi and T. Ristenpart, "Message-locked encryption and secure deduplication," *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[5] M. Bellare, S. Keelveedhi and T. Ristenpart, "DupLESS: server-aided encryption for deduplicated storage," *USENIX Security Symposium*, 2013.

[6] J. Blasco, A. Orfila, R. D. I. Pietro and A. Sorniotti, "A Tunable Proof of Ownership Scheme for Deduplication Using Bloom Filters," *IEEE Conference on Communications and Network Security (CNS)*, 2014.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422 - 426, 1970.

[8] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and its Applications," *Journal of Algorithms*, vol. 55, pp. 29–38, 2004.

[9] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," *International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[10] I. Drago, M. Mellia, M. M. Munafo, A Sperotto, R. Sadre and A. Pras, "Inside dropbox: understanding personal cloud storage services," *ACM conference on Internet Measurement Conference (IMC)*, 2012.

[11] M. Dutch and L. Freeman, "Understanding data de-duplication ratios," SNIA, 2009. http://www.snia.org

[12] Y. Duan, "Distributed key generation for encrypted deduplication: achieving the strongest privacy," *ACM Workshop on Cloud Computing Security (CCSW)*, 2014.

[13] K. Eshghi and H. K. Tang, "A Framework for Analyzing and Improving Content-Based Chunking Algorithms," *Hewlett-Packard (HP) Technical Report*, TR 2005-30, 2005.

[14] L. Fan, P. Cao, J. Almeida and A. Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3 pp. 281-293, 2000.

[15] Y. J. Fu, N. Xiao, X. K. Liao and F. Liu, "Application-aware client-side data reduction and encryption of personal data in cloud backup services," *Journal of Computer Science and Technology*, 2013.

[16] D. Guo, J. Wu, H. Chen, Y. Yuan and X. Luo, "The dynamic bloom filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 120-133, 2010.

[17] S. Halevi, D. Harnik, B. Pinkas and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," *ACM Conference on Computer and Communications (CCS)*, 2011.

[18] D. Harnik, B. Pinkas and A. Shulman-Peleg, "Side channels in cloud services, the case

of deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40-47, 2010.

[19] X Jin, L Wei, M Yu, N Yu and J Sun, "Anonymous deduplication of encrypted data with proof of ownership in cloud storage," *IEEE/CIC International Conference on Communications in China (ICCC)*, 2013.

[20] A. Juels and B. Kaliski Jr, "PORS: proofs of retrievability for large files," *ACM conference on Computer and Communications Security (CCS)*, 2007.

[21] N. Kaaniche and M. Laurent, "A secure client side deduplication scheme in cloud storage environments," *International Conference on New Technologies, Mobility and Security (NTMS)*, 2014.

[22] W. V. der Laan, Dropship, https://github.com/driverdan/dropship

[23] J. Li, X. Chen, M. Li, P. Lee and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615 – 1625, 2014.

[24] J. Li, Y. K. Li, X. Chen, P. P. C. Lee and W Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems* (to appear).

[25] C. Liu, X. Liu and L Wan, "Policy-based de-duplication in secure cloud storage," *Trustworthy Computing and Services*, pp. 250 - 262, 2013.

[26] L. Marques and C. J. Costa, "Secure deduplication on mobile devices," *Workshop on Open Source and Design of Communication (OSDOC)*, 2011.

[27] P. Meye, P. Raipin, F. Tronel and E. Anceaume, "A secure two-phase data deduplication scheme," *International Symposium on Cyberspace Safety and Security (CSS)*, 2014.

[28] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber and E. Weippl, "Dark clouds on the horizon: using cloud storage as attack vector and onlineslack space," *USENIX Security Symposium*, 2011.

[29] M. Naor and O. Reingold, "Number-Theoretic Constructions of Efficient Pseudo-Random Functions," *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.

[30] N. Nisan and D. Zuckerman, "Randomness is Linear in Space," *Journal of Computer and System Sciences*, vol. 52, pp. 43-52, 1996.

[31] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Computing Surveys (CSUR)*, vol. 47. No. 1, 2014.

[32] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012.

[33] P. Puzio, R. Molva, M. Ö nen and S. Loureiro, "Block-level de-duplication with encrypted data," *Open Journal of Cloud Computing (OJCC)*, vol. 1, no. 1, pp. 10 - 18, 2014.

[34] M. O. Rabin, "Fingerprinting by Random Polynomials," Center for Research in Computing Technology, Harvard University, *Tech Report TR-CSE-03-01*, 1981.

[35] F. Rashid, A. Miri and I. Woungang, "A Secure Data Deduplication Framework for Cloud Environments," *International Conference on Privacy, Security and Trust*, 2012.

[36] F. Rashid, A. Miri and I. Woungang, "Proof of retrieval and ownership protocols for enterprise-level data deduplication," *Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, 2013.

[37] Y. Shin and K. Kim, "Differentially private client-side data deduplication protocol for cloud storage services," *Security and Communication Networks* (to appear).

[38] J. Stanek, A. Sorniotti, E. Androulaki and L. Kencl, "A Secure Data Deduplication Scheme for Cloud Storage," Research Report IBM RZ 3852, 2013.

[39] M. Storer, K. Greenan, D. Long and E. Miller, "Secure data deduplication," *International Workshop on Storage Security and Survivability*, 2008.

[40] S. Vadhan, "Constructing Locally Computable Extractors and Cryptosystems in the Bounded-Storage Model," *Journal of Cryptology*, vol. 17, no. 1, pp. 43-77, 2004.

[41] J. Xu, E. C. Chang and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," *ACM SIGSAC symposium on Information, computer and communications security (ASIACCS)*, 2013.

[42] J. Xu and J. Zhou, "Leakage Resilient Proofs of Ownership in Cloud Storage, Revisited," *International Conference on Applied Cryptography and Network Security (ACNS)*, 2014.

[43] C. Yang and J. Ren, "Provable ownership of encrypted files in de-duplication cloud storage," *Ad Hoc & Sensor Wireless Networks* (to appear).

[44] C. Yang, J. Ren and J. Ma, "Provable ownership of files in de-duplication cloud storage," *IEEE Global Telecommunications Conference (GLOBECOM)*, 2013.

[45] S. Youngjoo and K. Kwangjo, "Efficient and Secure File Deduplication in Cloud Storage," *IEICE Transactions on Information and Systems*, Vol. E97-D, No.2, pp.184 – 197, 2014.

[46] C.-M. Yu, "HTTP Botnet Resilient to Takedown," *IEEE Symposium on Security and Privacy (S&P)*, San Jose, California, USA, 2014.

[47] C.-M. Yu, C.-Y. Chen, and H.-C. Chao, "Proof of Ownership in Deduplicated Cloud Storage with Mobile Device Efficiency," IEEE Network (to appear).

[48] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," *ACM conference on Data and Application Security and Privacy (CODASPY)*, 2012.