電子病歷系統的動態且有效率之完整性檢測

暨實作可搜尋的對稱式加密系統

學生：劉正偉　　指導教授：曾文貴

國立交通大學資訊科學與工程研究所碩士班

摘　要

隨著雲端儲存的興起，病人為了方便管理個人的病歷資料會選擇將病歷儲存在雲端在伺服器上。這就存在一個問題：假設這些提供儲存的伺服器是不可信任的情況下，我們如何確保放在伺服器上的電子病歷未被竄改呢? Ateniese 等人提出 Provable Data Possession (PDP)此協定來幫助使用者驗證儲存在不可信任的伺服器中的資料。在他們的協定中使用者會事先根據資料(通常代表一個檔案)計算出驗證使用的 metadata 後，將資料以及 metadata 送至伺服器上，接著使用者可以刪除使用者端的資料。使用者只要保留一些機密資訊就可以有很高的機率可以驗證放在伺服器上的資料正確性。然而 PDP 驗證只能針對靜態的資料。此外 PDP 是採用公開金鑰的方式，驗證以及證明的計算時間比較長。因此我們提出的方法考量到計算時間而採用 HMAC 的方式做驗證，除此之外，我們的方法還能支援修改病歷的操作。同時我們也實作了一套能讓使用者檢查病歷完整性的電子病歷系統。

使用者倘若將資料儲存在不可信任的伺服器中，為了保護其機密性，儲存在伺服器中的資料皆以加密方式保護。使用者上傳資料後，日後如果想搜尋在雲端上它儲存了哪些資料，由於資料已經加密過，並不能直接對密文搜尋。因此我們參考 Curtmola 等人提出的 Searchable Symmetric Encryption(SSE)的架構，實作了一個能在對稱式加密檔案中作關鍵字搜尋的系統，並且提供使用者能下載與搜尋結果相關的檔案。同時也對我們系統作效能的分析。

Dynamic and Efficient Provable Data Possession for Electronic Health Record System & Symmetric Searchable Encryption Implementation

Student: Cheng-Wei Liu Advisor: Wen-Guey Tzeng

Department of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

We propose a data integrity check scheme for electronic health record system. In our scheme, the patient can check the integrity of his health records when he finished uploading his health records. Obviously, our scheme is based on HMAC, so the computing time of the proof and the verification in our scheme is short. Our scheme also allows the patient to modify his health records on the server. In the end, we design a testing EHR system which allows the patient to check the correctness of his health records.

We consider that the user may outsource his files to the server. To protect the confidentiality of his files, the user will outsource his files with encrypted form. As a result, the user can't search over the encrypted files easily. Searchable symmetric encryption (SSE) allows the user to outsource the storage of his data to the server in a private manner, while maintaining the ability to selectively search over it. We build a keyword-searching system based on SSE scheme. Our system also allows the user to download the encrypted files associated with the search result. Last, we analyze the performance of our system.

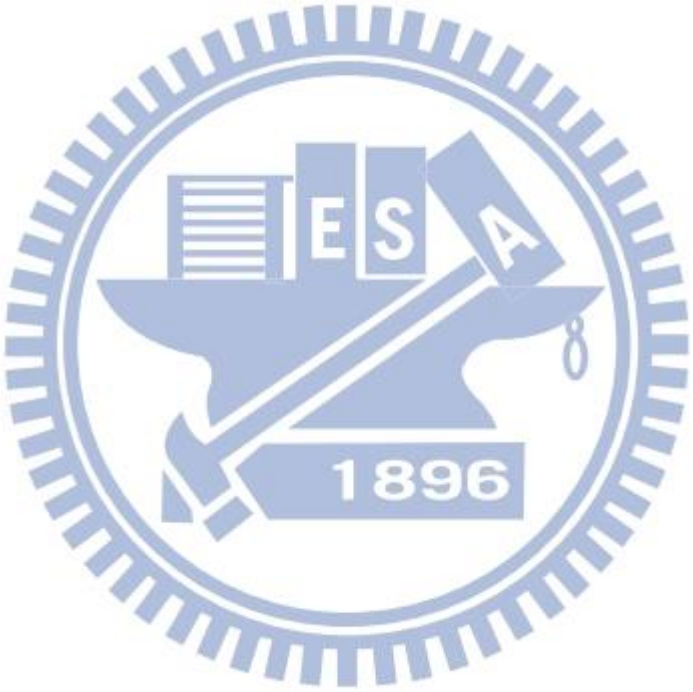
致　　謝

首先謝謝我的指導教授曾文貴老師，謝謝您帶領我進入密碼學的領域並且教導我如何思考問題。在與您做論文進度討論時，也不忘叮嚀我們該注意哪些細節以及提點我們疏忽的地方。在進研究所之前，對於上台報告實在是一竅不通，經過這兩年論文報告的訓練，也讓我漸漸地了解到如何在有限的時間內讓台下的聽眾能清楚地明瞭我報告的內容。其次感謝陳毅睿學長以及沈宣佐學長，感謝你們在我遇到論文上的困難時都不吝嗇地給予建議以及指導。另外感謝實驗室的同學們，平常互相都能給予幫忙以及鼓勵。感謝我的女友,每當我遇到挫折時都能聽我訴苦並且鼓勵我。感謝所有曾經幫助過我的人，讓我能順利地完成這篇論文。最後感謝我的父母親從小到大一路栽培，有你們的支持我才能順利地拿到碩士學位。

目錄

圖目錄

表目錄



一、 前言

病人在會診時，若清楚表達自身的健康狀況，能輔助醫生進行診斷，提升治療的成效。為了促進醫生病人之間溝通，病人在每次會診後，需保存且管理個人的病歷資料，以便往後每次的就醫中完整提供健康史給醫生做參考，尤其在用藥方面，需避免與其它藥物的交互作用或過敏現象。隨著資訊科技的發展與雲端儲存的興起，為了方便管理以及儲存個人的病歷資料，病人會選擇將這些病歷資訊儲存在雲端儲存服務商所提供的平台中。其中我們考量的是病人如何確保在不可信任的資料庫中的病歷資訊是完整的而沒有被竄改。舉例來說資料庫可能遭遇系統不穩定使得病歷資料出現錯誤，而資料庫可能會隱藏錯誤的資訊不讓病人知道。甚至由於為了節省儲存空間，資料庫可能會選擇性的儲存或是刪除較少存取的病歷資料。因此如何提出一個有效率的方法提供給病人驗證在資料庫中的病歷資訊的完整性是很重要的。

為了解決資料完整性檢測的問題，G. Ateniese 等人[1]提出 Provable Data Possession (PDP)的模型，在此模型底下，使用者會先對資料做前置處理，並且產生驗證所需的資訊。將資料上傳至伺服器後，使用者可刪除在使用者端的資料。在驗證的流程中，使用者隨機選擇要驗證的資料區塊做出挑戰(challenge)並將挑戰送至伺服器，伺服器根據使用者傳來的挑戰計算出資料沒被竄改過的證明(proof)給使用者，使用者只須持有一些機密的資訊便可驗證伺服器回傳的證明。因此這個架構(scheme)對於伺服器存有資料與否源自於使用者每一次的挑戰中所選的資料區塊(data block)提供機率性的保證。

之後也有許多學者針對 PDP 再作改進或是提出其他的觀點。在 PDP 中是採用公開金鑰系統(public key system)作標籤(tag)，而我們知道公開金鑰系統所需要的計算時間是比較長的，因此在[2][3]中提出 HMAC 的方式做標籤，所需的計算時間相對減少許多。另外因為 PDP 是使用公開金鑰，因此也可以做到公開驗證(public auditing)，即驗證的動作可以委託第三方的人來做，且第三方的人不需要得知使用者的機密資訊。這樣的好處是可以降低使用者端的計算負擔，而將計算成本轉移至具有更強大計算能力的第三方處理。在[4][5][6]中提出使用者可能會對在伺服器上的資料區塊做插入、刪除、修改的動作，在這種情況下提出了動態驗證的作法，每次只需要對欲修改的資料區塊作對應的標籤，而不需要將全部資料從伺服器取回重新產生新的標籤。

1.1研究動機

近幾年資訊科技的進步，病人會將個人的健康資訊，例如用藥記錄，儲存在網路平台中，再加上雲端儲存的技術興起，未來所有的病歷資訊都會轉移至雲端伺服器儲存，因此如何確保電子病歷的完整性是我們所需要關切的。此外我們也注意到電子病歷這種

資料只會有新增或是修改的動作，並不會將某個病人的電子病歷從資料庫移除，因此在這種情況下，我們希望提出一個有效率的方法可以讓每個病人對資料庫中有關於自己的病歷做完整性的檢測。

在[1][4][5][6]中所提出的方法都是利用公開金鑰系統的方式做驗證，處理大量病歷資料時所花費的時間會太長。在 Scalable PDP [2]中提出 HMAC 的方式做驗證，但是有次數上的限制，一旦挑戰都用完了必須取回所有資料重新做一次標籤以及挑戰。在[3]中提出的 HMAC 作法可以同時驗證多個資料區塊，但是對於資料修改或新增的動作並無法提供有效率的方法做驗證。

因此我們希望提出利用 HMAC 的方式做標籤，以節省計算時間，此外我們設計的架構能支援病歷資料的修改和新增的動作。

近年來雲端儲存的發展，使用者願意將資料上傳到這些雲端儲存伺服器中。我們假設這些伺服器是不可信任的情況下，若要保護使用者的資料，通常會將資料加密後上傳至伺服器儲存，同時為了降低使用者的計算負擔，加密方法會利用對稱式的加密系統，例如 Advanced Encryption Standard(AES)。由於檔案都是以加密形式儲存於伺服器，當使用者想查看在伺服器中的檔案，此時並無法直接對加密的檔案搜尋，因此最簡單的做法就是將所有跟使用者相關的檔案都下載到使用者端再作解密的動作，但我們知道這樣消耗的溝通成本(communication cost)過大，佔據大量的網路頻寬，因此必須提供一種方式能讓使用者在保護檔案隱私的同時，也能提供一種機制使得使用者得以在加密檔案做搜尋。最後使用者還可針對搜尋結果選擇要下載哪些加密檔案。為了達到這個目標，我們參考[7]中所提出的 Searchable Symmetric Encryption (SSE)，實作一個讓使用者可以搜尋在伺服器中的加密檔案的系統。我們會在第七章節獨立出來介紹這部分。

1.2 問題描述

使用者想要將個人的健康記錄(例如用藥記錄)儲存在伺服器中，假設伺服器是不可信任的情況下，我們希望提出一套方法，使得使用者只需要儲存固定少量的資訊，即可驗證儲存在伺服器上的電子病歷完整性。除此之外，使用者日後可能想要修改電子病歷的內容，我們也希望所提出的方法能有效率的支援修改病歷的操作並且能驗證修改過的病歷資料。

同樣地在伺服器是不可信任的情況下，使用者會將文件以加密形式儲存在伺服器。我們參考[7]提出的 SSE 架構，實作出一套系統使得使用者可以在加密文件中作關鍵字搜尋，除此之外還可以根據搜尋結果選擇下載加密的文件。

二、 相關文獻探討

G. Ateniese 等人[1]提出的 Provable Data Possession (PDP)是第一個提出機率性的資料完整性證明，他們檢閱先前的研究發現一些缺點發現之前提出的協定(protocol)在檢查資料上都需要大量的伺服器運算以及溝通成本，並且對資料擁有無法提供安全性上的證明。因此他們在[1]除了提出 PDP 的架構之外，也提供了一個完整的安全性證明，同時針對公開驗證(public auditability)的觀點也有提出修改過的 PDP 版本。但是 PDP 的驗證只能針對靜態(static)資料，並沒有考慮資料涉及更新的情況。也就是說一旦有新的資料要上傳至伺服器，就必須將伺服器上的所有資料重新下載並對所有的資料，包括新的，重新計算出新的標籤，如此一來才能對新的資料做驗證。

Shacham 和 Waters 提出的 Compact proofs of retrievability (CPoR)[3]有提出 HMAC 的方式來做標籤，優點是使用者作標籤的時間以及伺服器計算證明的時間會比較短，另外使用者端也只需要儲存固定少量的資訊即可驗證伺服器上的資料。除此之外，也提供了一套完整的證明。但是同樣的 CPoR 的驗證也只能針對靜態資料。

G. Ateniese 等人[2]另外也提出動態的(Dynamic)資料驗證協定，稱為 Scalable PDP。為了讓驗證有效率，他們也是利用 HMAC 的方式來做他們的驗證標記(verification token)來節省計算時間，此外他們協定的核心概念是先將未來會用的挑戰以及對應的驗證標記在一開始就計算好並將驗證標記加密後一起與資料上傳至伺服器。每當使用者要驗證時就挑一組挑戰給伺服器做資料擁有的證明，伺服器回傳證明以及加密驗證標記後，使用者比對解出的驗證標記與伺服器作的證明是否一致。因此此種協定只能做有限次的驗證，一旦挑戰耗盡，必須將檔案下載重新挑選新的挑戰並計算對應的驗證標記。此外，Scalable PDP 提出他們能支援資料區塊作更新的動作，但是此種方法所限制，對於每次資料做更新，使用者就必須將剩餘的驗證標記從伺服器下載回來並重新計算。

Erway 等人[4]第一個提出完整的動態的資料完整性檢測稱作 Dynamic Provable Data Possession(DPDP)，即對於資料新增、修改、插入、刪除四種操作都能有效率的做驗證。他們是利用 authenticated skip list 此種資料結構來檢查資料區塊的完整性，使用者將檔案以及相關的 skip list 傳到伺服器後，使用者端只需要保留 skip list 中起始點的資訊即可驗證檔案的完整性，而伺服器對應使用者的挑戰所做出的證明大小為 O(log n)，n 為檔案區塊個數。此外，DPDP 針對資料新增、修改、插入、刪除四種操作，伺服器計算時間、使用者計算時間複雜度也是 O(log n)。

Wang 等人[5]強調公開驗證的觀點，他們認為驗證伺服器的證明正確與否可以交由可信任的第三方來實現，如此一來可以降低使用者端的負擔使其計算成本轉移至計算能力較強的第三方。為了達到完整的動態資料檢測，他們認為先前研究所作的標籤中的

H(name||i)或是 H(v||i)沒有辦法支援資料插入的操作，因此在這篇的方法中使用 H(m_i)取代先前的做法，但是改成 H(m_i)後，由於沒有索引的資訊，伺服器就有更多的機會欺騙驗證者。因此為了解決這個問題，在每次驗證的步驟都必須額外檢查 H(m_i)的正確性。在驗證資料的部分，他們使用 Bilinear map 的運算性質來驗證資料的完整性，同時為了能支援資料的動態操作，他們採用 Merkle Hash Tree 的方式來達成。

Yang 和 Jia [6]同樣地使用 Bilinear map 的運算性質來驗證資料的完整性，除此之外他們提出 Index Table 的方式，將每一個資料區塊的現今索引編號，原先索引編號，目前版本編號以及時戳存放在 Index Table 中交由驗證者(auditor)保管此表格。利用 Index Table 他們也能支援資料的動態操作。最後他們還提出他們的方法能做到 batch auditing 的效果，也就是說驗證者可以針對不同的使用者儲存在不同的雲端伺服器上的資料只做一次的驗證動作。

我們的貢獻

相對於用公開金鑰的方式驗證，我們提出 HMAC 的方式有效率地來驗證電子病歷的完整性；不同於 Scalable PDP，我們的驗證方法沒有次數上的限制；不同於 CPoR，我們的方法可以有效率地支援修改病歷的操作。

接下來的文章組織如下：

第三章我們會介紹我們架構中所用到的密碼工具以及定義我們的系統模型和安全模型。第四章詳細地講解我們提出的架構。第五章分析我們架構的安全性。第六章說明我們實作的系統以及實驗方法與結果分析。第七章我們獨立出來講解我們另一部分實作 SSE 系統，包括 SSE 的設計原理以及我們實作的 SSE 系統，同時也會提到我們實驗的方法以及結果分析。第八章討論我們提出的架構以及第二部分實作的 SSE 系統的限制。第九章則是我們最後的結論。

三、 背景知識與基本定義

我們假設使用者 C 想要上傳一筆病歷 F 至伺服器 S 上，其中 F 裡包含著 n 個欄位資訊，記作 $F=(F_1,F_2..,F_n)$。

3.1 Keyed-hash Message Authentication Code(HMAC)

訊息驗證碼(message authentication code)是一段少量的資訊用來驗證訊息的完整性，其中 HMAC 是一種有金鑰參與的雜湊函數(hash function)，因為 HMAC 具有不可偽造性(unforgeable)，也就是說任兩個不同的訊息所做出來的 HMAC 值就不相同，所以也可以當作訊息驗證碼使用。本篇使用的 HMAC 是使用有金鑰參與的 SHA-1 雜湊函數，我們也利用 HMAC 的性質來做病歷欄位資訊的標籤。

3.2 Homomorphic Verifiable Tags(HVTs)

這部分我們會介紹 homomorphic verifiable tags(HVTs)以及如何將它應用到我們的架構上。

給定一個病歷欄位資訊 F_i，我們用t_i表示為 F_i 的 homomorphic verifiable tags，而這些 tag 都會跟病歷 F 一起儲存在伺服器 S 中。而這些 homomorphic verifiable tags 是給伺服器 S 用來作證明所需要的資訊，它除了不可偽造性之外還有以下兩種性質：

Blockless verification：使用者 C 若只想知道病歷中的部分欄位資訊是否有被竄改，他可以選擇特定幾種欄位資訊來做挑戰送給伺服器 S，讓伺服器只要證明這些特定的欄位資訊是否有完整存放，如此一來使用者就不需要真的存取整筆病歷資料 F 才能驗證那些他關心的欄位資訊。

Homomorphic tags：給定任意兩個t_i和t_j，我們都可以將它們組合成t_{i+j}，而這個值是與欄位資訊 F_i+F_j 相關的。這個性質能使得不管使用者要驗證幾個欄位資訊，伺服器 S 作的證明都能組合成一個值以節省溝通成本。

3.3 Merkle Hash Tree

Merkle hash tree(MHT)是一種可用來做驗證的資料結構一、[8]，它可以有效率地證明一組元素是沒有被更動或破壞的。MHT 是一個二元樹，我們將驗證的元素放在葉子

端，不在葉子端的節點 h_i 的值則是它的兩個子節點連結後帶入雜湊函數的結果，即 $h_i=H(h_{2i+1}||h_{2i+2})$，其中 H 為一雜湊函數。按照此種方式計算二元樹的非葉子端節點的值一直到根節點 h_r。我們知道雜湊函數代入的參數不同，計算出來的結果也會不同，倘若欲驗證的元素遭到竄改，所做出來的 MHT 的根節點 h_r 也必定與原先不同，因此驗證者只需要保留 h_r 即可驗證葉子端的元素，圖 1 為驗證的一個例子。假設資料為 $\{x_1, x_2, \ldots, x_8,\}$，而驗證者只擁有根節點值$h_r$。此時驗證者向證明者要求證明其中$\{x_1, x_6\}$是否正確，證明者便需要提供$x_1, x_6$以及對應的輔助驗證資訊(auxiliary authentication information) $\Omega_1 = (x_2, h_d)$，$\Omega_6 = (x_5, h_f)$給驗證者。驗證者首先計算$h_c = \mathrm{h}(x_1||x_2)$，$h_e = \mathrm{h}(x_5||x_6)$，接著計算$h_a = \mathrm{h}(h_c||h_d)$，$h_b = \mathrm{h}(h_e||h_f)$，最後計算$h'_r = \mathrm{h}(h_a||h_b)$後，比較$h'_r$與原先儲存的$h_r$是否一樣，若是一樣則驗證$\{x_1, x_6\}$成功。

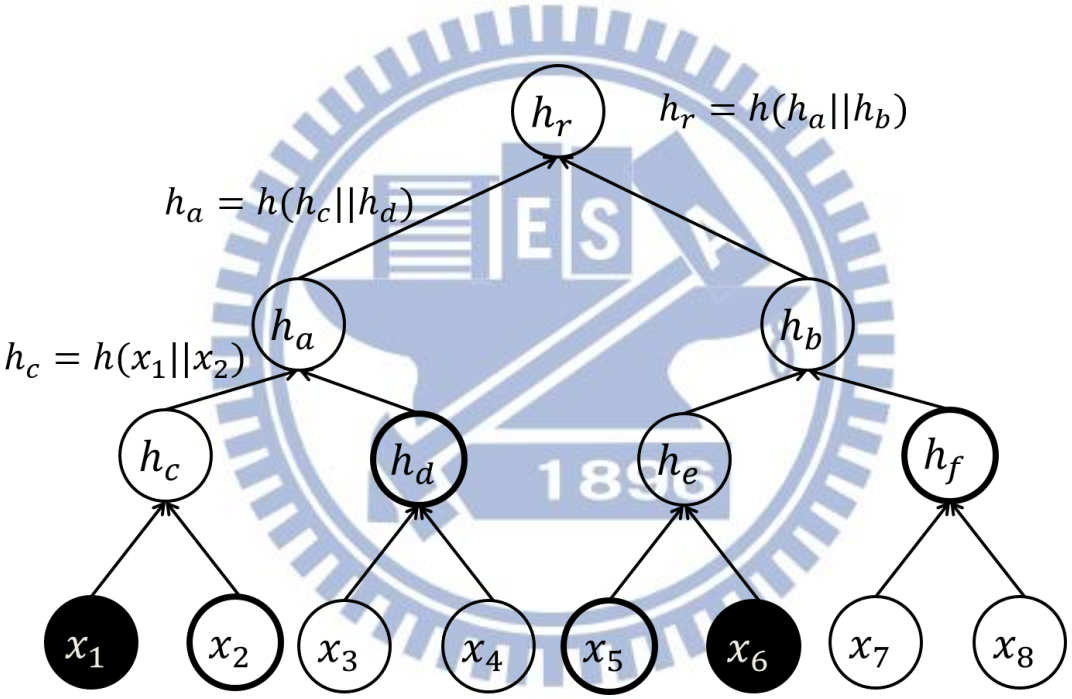


圖 1 Merkle Hash Tree 範例

3.4 系統模型

我們考慮的驗證系統如圖 2 所示，包括使用者以及不可信任的伺服器。使用者可以新增病歷到伺服器中，並且能存取伺服器中所屬的電子病歷。使用者可透過送出挑戰至伺服器驗證在伺服器中的電子病歷完整性。我們所提出的架構是由五個演算法 (KeyGen,TagGen,ChallGen,GenProof,CheckProof)所建造而成，接下來部份我們會定義這

五個演算法。表格 1 是我們架構中會用到的符號表示。

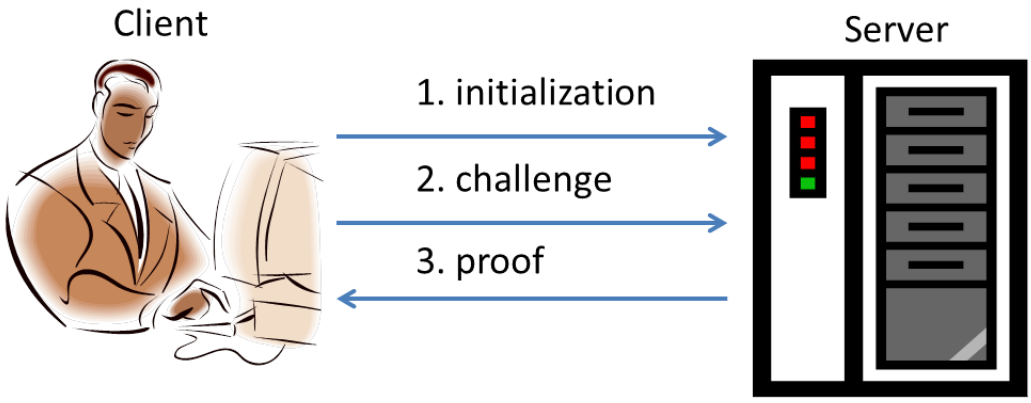


圖 2 系統概觀

表格 1 符號表示及意義

符號	代表意義
sk_h	HMAC 的機密金鑰
α	使用者端的機密係數
H(.)	無金鑰參與的密碼的雜湊函數，如 SHA-1
$H_k(.)$	有金鑰 k 參與的密碼的雜湊函數，如 HMAC-SHA-1
F	病歷欄位資訊的集合
T	病歷欄位資訊的標籤集合
$\mathcal{H}$	病歷欄位資訊的雜湊值集合
n	病歷欄位個數
V	病歷欄位的版本編號集合，是用來記錄欄位更新次數
h_{root}	MHT 的根節點的值

- KeyGen(1^k) → (sk_h, α)

是一個由使用者執行的金鑰產生演算法，將安全參數 k 輸入後，得到機密金鑰sk_h以及機密係數α後將兩個值保存在使用者端，目的是為了製作標籤使用。

- TagGen(F, sk_h, α) → (T,$\mathcal{H}$,V, h_{root})

是一個由使用者執行的標籤產生演算法，輸入參數為病歷 F，機密金鑰sk_h以及機密係數α。此演算法是對每一個病歷欄位資訊 F_i，透過sk_h，對應的版本號v_i，索引 i 以及α計算出標籤t_i。除此之外，根據欄位索引和版本編號代入$H_{sk_h}(.)$得到的雜湊值w_i，我們利用這些雜湊值$\{w_i\}$建造 MHT，並且保留根節點的值h_{root}。最後輸出結果為

T=$\{t_i\}_{i\in[1,n]}$，$\mathcal{H}$=$\{w_i\}_{i\in[1,n]}$，V = $\{v_i\}_{i\in[1,n]}$以及h_{root}，之後使用者會將(F,T, $\mathcal{H}$, V)上傳至伺服器，而h_{root}會保留在使用者端。

- ChallGen(n) → C

是一個由使用者執行的挑戰產生演算法，輸入參數為病歷欄位個數 n，根據 n 隨機輸出挑戰 C。

- ProofGen(F,T, $\mathcal{H}$,C) → P

是一個由伺服器執行的證明產生演算法，輸入參數為病歷 F，標籤集合 T，雜湊值集合$\mathcal{H}$以及挑戰 C，伺服器根據這些值計算出證明 P，證明它擁有這些欄位資訊。最後將 P 傳給使用者驗證。

- CheckProof(sk_h, α, h_{root},C,P) → 0/1

是一個由使用者執行的驗證演算法，輸入參數為機密資訊sk_h，機密係數α，根節點值 h_{root}，挑戰 C 以及伺服器計算的證明 P，根據這些資訊輸出驗證結果，若驗證成功輸出 1，反之輸出 0 代表驗證失敗。

3.5 安全模型

我們假設伺服器是不可信任的情形下，伺服器可能會有下列幾種攻擊[6]：

- Replace Attack. 當伺服器把F_l刪除時，它可能會挑選另一組合法的(F_k, w_k, t_k)來取代被挑戰的(F_l, w_l, t_l)來做證明。

- Forge Attack. 當伺服器把F_l刪除時，它可能會偽造$F_k \neq F_l$的合法標籤企圖欺騙使用者。

- Replay Attack. 伺服器透過先前作的證明或是其他資訊而不讀取真正的病歷欄位資訊的情況下作證明。

四、 我們提出的架構

4.1 驗證架構的演算法

這邊我們將詳細介紹我們針對電子病歷系統所設計的驗證演算法。首先我們選出一個很大的質數 p，我們所有演算法的運算都會在 Z_p 中。我們的驗證協定是由五個演算法組合而成的，作法如下：

4.1.1 KeyGen(1^k) → (sk_h, α)

金鑰產生演算法輸入安全參數 k 之後，我們選擇一個隨機的數$sk_h \in_R Z_p$當作是我們的機密金鑰以及挑選 1 個亂數$\alpha \in_R Z_p$，並且將它們作為輸出結果。

4.1.2 TagGen(F, sk_h, α) → (T,$\mathcal{H}$,V, h_{root})

標籤產生演算法將病歷欄位資訊集合 F，機密金鑰sk_h以及機密係數α當作輸入參數。首先對於每個欄位資訊F_i產生對應的版本編號v_i=0，接著我們會將版本編號v_i，病歷欄位索引 i 以及機密金鑰sk_h代入 HMAC 函數得到$w_i = H_{sk_h}(v_i||i)$。由於我們之後會將病歷版本編號v_i儲存在伺服器上，因此為了驗證v_i完整性，我們利用了 MHT 來達成此目的。我們將剛剛得到的$w_1, w_2, \dots, w_n$放在 MHT 的葉子端，透過$w_1, w_2, \dots, w_n$這些值代入 H(.)來建造 MHT 並且得到根節點的值 h_{root}。最後我們計算出每一個病歷欄位資訊 F_i對應的標籤 $t_i = (w_i + h_i + \alpha \cdot F_i)\ mod\ p$，其中$w_i = H_{sk_h}(v_i||i)$，$h_i = H_{sk_h}(i)$。此演算法輸出結果為病歷欄位資訊標籤集合T $= \{t_i\}_{i\in[1,n]}$，雜湊值集合$\mathcal{H} = \{w_i\}_{i\in[1,n]}$，版本編號集合V $= \{v_i\}_{i\in[1,n]}$以及 MHT 根節點的值 h_{root}。

4.1.3 ChallGen(n) → C

將病歷的欄位個數 n 當作輸入參數後，我們隨機挑選一些欄位建立我們的挑戰集合 Q 並且對被選到的每個欄位資訊 F_i產生對應的亂數$\beta_i \in_R Z_p$(i ∈ Q)。最後輸出結果為挑戰C $= \{i, \beta_i\}_{i\in Q}$。

4.1.4 ProofGen(F,T, $\mathcal{H}$,C) → P

此證明產生演算法將病歷 F，病歷欄位的標籤集合 T，病歷欄位的雜湊值集合$\mathcal{H}$以及挑戰 C 當作輸入參數。我們的證明 P 是由標籤證明 TP 以及版本號證明 VP 所組合而

成。版本號證明 VP 是根據挑戰 C 得知欲驗證的欄位索引$\{i\}_{i\in Q}$，透過 $\mathcal{H}$計算出對應的欄位雜湊值的輔助驗證資訊$\{\Omega(w_i)\}_{i\in Q}$，最後 VP 即為被挑選的雜湊值以及輔助驗證資訊的集合，VP=($\{w_i\}_{i\in Q}$, $\{\Omega(w_i)\}_{i\in Q}$)。接著 TP 分成兩部分計算，$\tau = \left(\sum_{i\in Q}\beta_i \cdot t_i\right) mod\ p$，$\mu = \left(\sum_{i\in Q}\beta_i \cdot F_i\right) mod\ p$，TP=(τ, μ)。最後此演算法輸出結果為 P=(VP,TP)。

4.1.5 CheckProof(sk_h, α, h_{root},C,P) → 0/1

此驗證演算法將機密金鑰sk_h，機密係數α，機密資訊h_{root}，挑戰 C 以及證明 P 當作輸入參數。我們驗證正確與否也是分為兩階段，第一部分是驗證挑選的$\{w_i\}_{i\in Q}$是否正確，方法是透過證明 P 中的 VP 裡的$\{w_i\}_{i\in Q}$以及$\{\Omega(w_i)\}_{i\in Q}$去計算出此 MHT 的根節點值h'_{root}，我們比對h'_{root}與h_{root}是否一樣，若是一樣，根據 MHT 的驗證方式，我們能確認$\{w_i\}_{i\in Q}$的正確性。我們確定挑選的$\{w_i\}_{i\in Q}$無誤之後，第二部分使用者計算 $\sigma = \left(\alpha \cdot \mu + \sum_{i\in Q}\beta_i \cdot w_i + \sum_{i\in Q}\beta_i \cdot h_i\right) mod\ p$，比對σ是否與 TP 中的τ一樣。最後若是兩部分的驗證都通過，此演算法輸出 1 表示驗證成功。若是任其一驗證有誤，則輸出 0 表示驗證失敗。

4.2 驗證架構的建造

如圖 3 所示，我們的驗證協定包括兩個部分：初始化階段以及驗證階段。在初始化階段使用者產生金鑰以及病歷欄位資訊的標籤，上傳病歷欄位資訊後，使用者便可選擇將使用者端的資料刪除。使用者在驗證階段確保病歷資料是否正確的儲存在伺服器上。

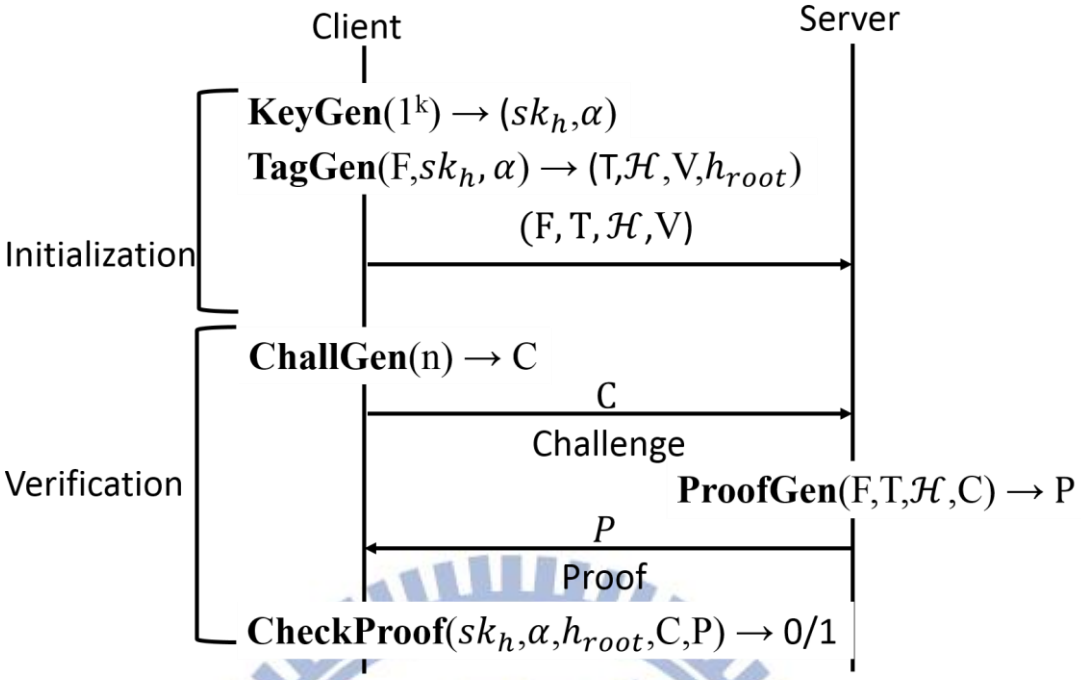


圖 3 驗證架構示意圖

4.2.1 初始化階段

使用者執行金鑰產生演算法 KeyGen 得到機密金鑰sk_h以及機密係數α。接下來執行標籤產生演算法 TagGen 得到病歷欄位資訊標籤集合$\mathrm{T}=\{t_i\}_{i\in[1,n]}$，雜湊值集合$\mathcal{H}=\{w_i\}_{i\in[1,n]}$，病歷版本編號集合$\mathrm{V}=\{v_i\}_{i\in[1,n]}$以及 MHT 根節點的值 h_{root}。完成之後將所有病歷欄位資訊$\mathrm{F}=\{F_i\}_{i\in[1,n]}$，對應的標籤$\mathrm{T}=\{t_i\}_{i\in[1,n]}$，對應的雜湊值$\mathcal{H}=\{w_i\}_{i\in[1,n]}$和病歷版本編號集合$\mathrm{V}=\{v_i\}_{i\in[1,n]}$上傳至伺服器。上傳後使用者可以選擇將使用者端的$\mathrm{F},\mathrm{T},\mathcal{H},\mathrm{V}$ 刪除，只需要保留機密金鑰sk_h，機密係數α，MHT 根節點的值h_{root}，以及病歷欄位個數 n。

4.2.2 驗證階段

使用者執行挑戰產生演算法 ChallGen 得到挑戰$\mathrm{C}=\{i,\beta_i\}_{i\in Q}$，將 C 傳給伺服器。

伺服器收到 C 之後，執行證明產生演算法 ProofGen 產生 P=(VP,TP)，隨後將 P 傳給使用者。

當使用者收到 P 之後，執行驗證演算法 CheckProof 檢查 P 的正確性並且產生驗證結果。倘若驗證結果為 1，我們可以確信伺服器確實擁有使用者挑選的欄位資訊；反之，儲存在伺服器上的欄位資訊可能遭受竄改。

4.3 動態驗證的演算法以及建造

我們提出的架構可讓使用者修改在伺服器上的病歷欄位資訊，接下來我們會說明我們的演算法以及建造方法。如圖 4 所示。

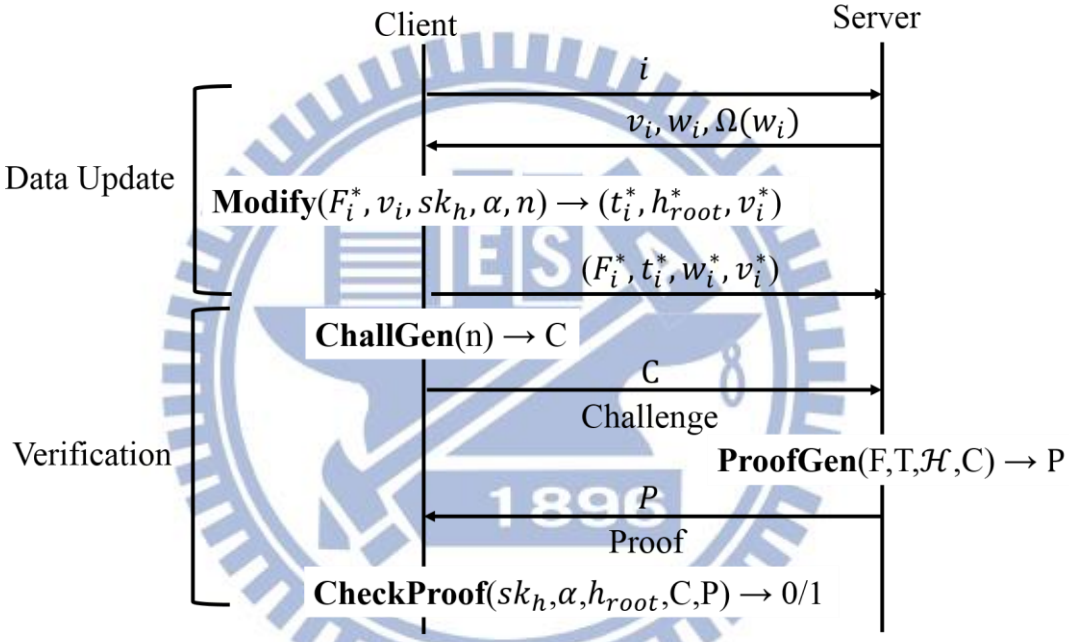


圖 4 動態的驗證架構示意圖

4.3.1 病歷資料更新

使用者要修改病歷之前先傳送要修改的病歷欄位索引 i 給伺服器，伺服器收到後會回傳病歷版本編號v_i以及雜湊值w_i。為了確保病歷版本編號v_i是正確的，使用者依據sk_h，i 以及v_i計算出w'_i，比對與伺服器傳來的w_i是否一樣。接著才執行 Modify 演算法來修病歷，方法如下：

Modify$(F_i^*, v_i, sk_h, \alpha, n) \rightarrow (t_i^*, h_{root}^*, v_i^*)$. 此演算法是修改病歷的演算法，輸入參數為新的病歷欄位資訊F_i^*，病歷版本編號v_i，機密金鑰sk_h，機密係數α以及病歷欄位個數

n。首先使用者會計算新病歷版本標號$v_i^* = v_i+1$。接著使用者產生新的標籤 $t_i^* = (w_i^* + h_i + \alpha \cdot F_i^*)\ mod\ p$, 新的雜湊值$w_i^* = H_{sk_h}(v_i^*||i)$以及透過$w_i^*$和 $\Omega(w_i)$計算新的根節點值h_{root}^*。最後將$(F_i^*, t_i^*, w_i^*, v_i^*)$上傳至伺服器後使用者可以選擇將使用者端的$(F_i^*, t_i^*, w_i^*, v_i^*)$刪除，保留新的根節點值$h_{root}^*$。

4.3.2 更新驗證

使用者修改病歷並且上傳至伺服器後，執行挑戰產生演算法 ChallGen 得到挑戰 $C = \{i, \beta_i\}_{i\in Q}$，將 C 傳給伺服器。

伺服器收到 C 之後，執行證明產生演算法 ProofGen 產生 P=(VP,TP)，隨後將 P 傳給使用者。

當使用者收到 P 之後，執行驗證演算法 CheckProof 檢查 P 的正確性並且產生驗證結果。倘若驗證結果為 1，我們可以確信伺服器確實擁有使用者挑選的欄位資訊；反之，儲存在伺服器上的欄位資訊可能遭受竄改。

4.4 偵錯機率

我們讓使用者可以挑選哪些欄位來驗證，所以如果有出錯的欄位沒被挑中的話，驗證還是能順利通過，因此我們在這節分析我們方法的偵錯機率。

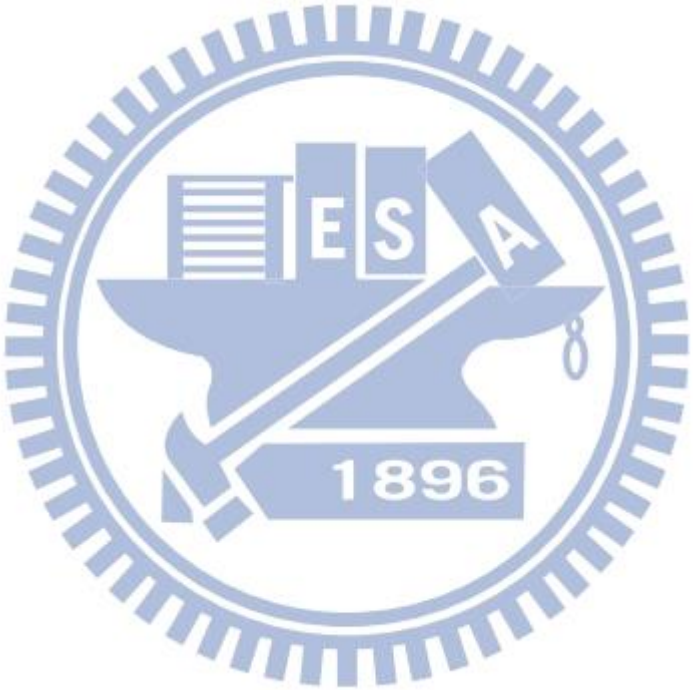
假設欄位個數為 n，被竄改的欄位個數為 m 以及 challenge 每次挑選 t 個欄位做驗證的情況下，偵錯機率為$Pr_{det} = 1 - (1 - \frac{m}{n})^t$。

4.5 效能比較

表格 2 是我們和 CPoR[3]的比較，我們的做法能支援病歷修改的操作。然而在使用者持有資訊的部分，需要額外儲存一個 MHT 的根節點值 R。在證明部分，必須額外計算被挑選到雜湊值的輔助驗證資訊$\{\Omega(w_i)\}_{i\in Q}$。伺服器傳給使用者的傳輸成本最差的情況下，也就是使用者驗證全部的欄位值，要額外傳$\{w_i\}_{1\le i\le n}$。使用者驗證部分則需要額外驗證挑選的$\{w_i\}_{i\in Q}$。以上這些是我們為了支援修改病歷所付出的額外成本。

表格 2 消耗成本比較

	使用者持有資訊	伺服器計算證明	傳輸成本	使用者計算驗證結果
CPoR[3]	α, k	τ, μ	τ, μ	τ
我們的架構	α, k,R	1. $\{\Omega(w_i)\}_{i\in Q}$ 2. τ, μ	$\tau, \mu, \{w_i\}_{1\le i\le n}$	1. $\{w_i\}_{i\in Q}$ 2. τ



五、 安全性分析

定理 1：我們提出的架構可阻擋 Replace Attack。

證明：

伺服器若是將被挑戰 C=$\{i,\beta_i\}_{i\in Q}$挑選的(F_l,w_l,t_l)置換成(F_k,w_k,t_k)，所做的證明 P 中的 VP*=$\{\{w_i\}_{i\in Q,i\neq l},\{\Omega(w_i)\}_{i\in Q,i\neq l},w_l,\Omega(w_l)\}$。我們知道 cryptographic hash function 有 collision-resistant 的性質，因此經由 VP*所計算的 MHT 的h^*_{root}會不等於使用者儲存的h_{root}。

除此之外，伺服器作的 TP=(τ,μ)其中$\tau=\left(\sum_{i\neq l,i\in Q}(t_i\cdot\beta_i)+t_k\cdot\beta_l\right)mod\ p$，$\mu=\left(\sum_{i\neq l,i\in Q}(F_i\cdot\beta_i)+F_k\cdot\beta_l\right)mod\ p$。使用者驗證時計算$\sigma=\left(\alpha\cdot\mu+\sum_{i\neq l,i\in Q}\beta_i\cdot w_i+\beta_l\cdot w_k+\sum_{i\in Q}\beta_i\cdot h_i\right)mod\ p$。若伺服器作的 TP 能通過驗證，我們可得到$0=\tau-\sigma=\beta_l(h_k-h_l)$，同樣地利用 cryptographic hash function 有 collision-resistant 的性質，我們知道$h_k\neq h_l$，因此伺服器所做的 TP 無法通過驗證。由上述可知，我們的架構可以阻擋 Replace Attack。

定理 2：我們提出的架構可阻擋 Forge Attack。

證明：

假設伺服器是偽造挑戰 C=$\{i,\beta_i\}_{i\in Q}$所挑選的F_l的標籤，也就是說伺服器拿$F_l^*\neq F_l$企圖製作第 l 個位置的合法標籤，則伺服器計算的證明 P 中的 TP=(τ,μ)其中
$\tau=\left(\sum_{i\neq l,i\in Q}(t_i\cdot\beta_i)+t_l^*\cdot\beta_l\right)mod\ p$，$\mu=\left(\sum_{i\neq l,i\in Q}(F_i\cdot\beta_i)+F_l^*\cdot\beta_l\right)mod\ p$。使用者驗證時計算$\sigma=\left(\alpha\cdot\mu+\sum_{i\in Q}\beta_i\cdot w_i+\sum_{i\in Q}\beta_i\cdot h_i\right)mod\ p$。若伺服器偽造的標籤能通過驗證，我們可得到$0=\tau-\sigma=t_l^*\cdot\beta_l-(\alpha\cdot F_l^*\cdot\beta_l+w_l\cdot\beta_l+h_l\cdot\beta_l)\ mod\ p$。由於伺服器不知道$sk_h$以及 α，伺服器對於h_i無法分辨它是由 PRF 產生或是隨機值，因此伺服器要成功偽造標籤的機率為$\frac{1}{p}$，而這個機率是可忽略的，所以我們的架構可阻擋 Forge Attack。

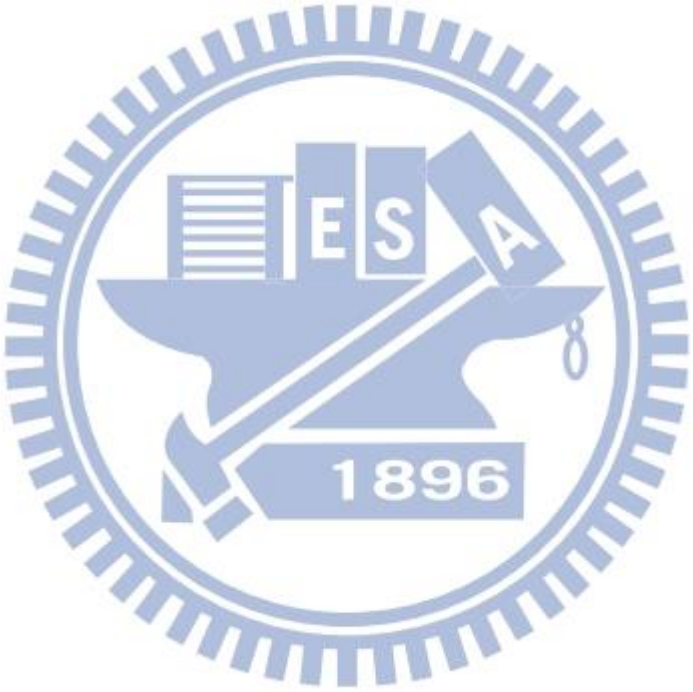
定理 3：我們提出的架構可阻擋 Replay Attack。

證明：

使用者修改病歷時所製作的$t_i^*=(w_i^*+h_i+\alpha\cdot F_i^*)\ mod\ p$,其中的$w_i^*=h_{sk_h}(v_i^*||i)$裡的$v_i^*$也會更改，因此伺服器若使用舊的$w_i$作擁有$F_i^*$證明，使用者利用 MHT 驗證$w_i^*$時

便會發現有錯誤。

除此之外，使用者每次挑選的挑戰 C 中的$\{\beta_i\}_{i\in Q}$皆不同，因此伺服器無法用先前的證明來產生證明，因此我們的架構可阻擋 Replay Attack。



六、 系統實作以及實驗

我們在這個章節會說明我們的實驗環境以及使用工具，接著會介紹我們的電子病歷系統實作方法，最後則是會說明我們的實驗結果。

6.1 實驗環境以及使用工具

這節會介紹我們使用的工具以及平台，表格 3 說明使用者端以及伺服器端的電腦軟硬體配備。第四章節中提到的演算法我們是以 C++來撰寫，並且採用 GMP-5.1.1 函式庫[12]支援我們的大數運算。為了設計使用者可以操作的介面以及存放使用者的電子病歷，我們使用 PHP 來撰寫網頁來完成我們的電子病歷系統以及使用者介面，並且用 MySQL 資料庫來存放使用者的電子病歷。為了保護使用者的隱私資料，我們大部分的網頁是放在使用者端操作的，只有伺服器作證明的網頁是放在伺服器端。因此，為了能順利開啟網頁，使用者端以及伺服器端都必須安裝 Apache 以便開起網頁。我們會在 6.2 節詳細介紹各個網頁的功能。

表格 3 軟硬體配備

	使用者端	伺服器端
作業系統	Windows7 64bit	Windows7 64bit
CPU	Intel Core i5 2.53GHz	Intel Core i5 2.80GHz
RAM	4.00GB	8.00GB
資料庫	無	MySQL
Apache	有	有

6.1.1 GMP 函式庫

為了在 windows 作業系統上使用這套函式庫，我們參考[13]使得我們能在 visual studio 2010 上使用 GMP，接下來我們會介紹設定的方法。

首先從 http://www.cs.nyu.edu/exact/core/gmp/下載 dynamic GMP library and header file for Visual C++: gmp-dynamic-vc-4.1.2.zip，解壓縮之後放到 visual studio 專案資料夾底下，接著在專案中的標頭檔加入 gmp.h，資源檔加入 gmp.lib 以及 gmpDebug.lib，之後撰寫程式時只要 #include "gmp.h"即可使用 GMP 函式庫，如圖 5 所示。

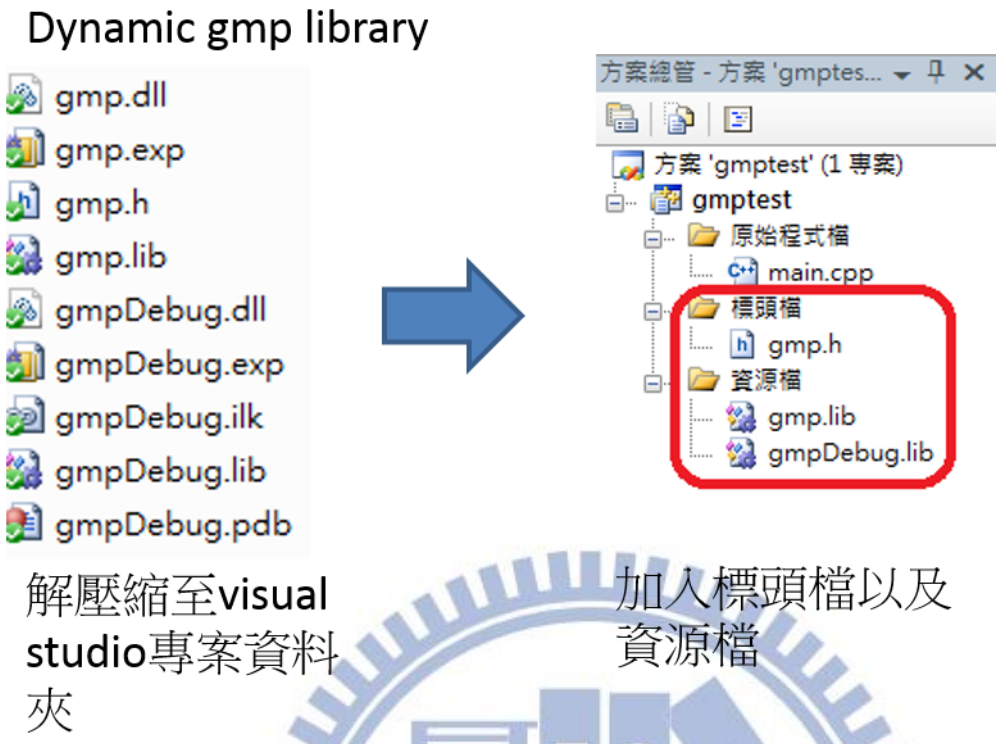


圖 5 GMP 函式庫使用方法

6.1.2 電子病歷格式

這邊我們參考行政院衛生署電子病歷推動專區[14]裡提供的門診用藥交換欄位與格式之標準規範，我們採用其中的門診用藥交換之欄位作為我們系統中電子病歷的格式範本，如表格 4 所示。

表格 4 門診用藥紀錄基本格式欄位

行政院衛生署 2010.09.17 版本

項次	區塊描述	欄位名稱	LOINC 對應名稱	欄位說明
1	醫事機構	醫事機構代碼 Hospital Id		[1..1]
2		醫事機構名稱 Hospital Name		[1..1]
3	病人基本資料	身分證號 Personal ID Number		身分證號、護照號碼或居留證號[1..1]

項次	區塊描述	欄位名稱	LOINC 對應名稱	欄位說明
4		病歷號碼 Chart No.		[1..1]
5		姓名 Name		[1..1]
6		性別 Gender		[1..1]
7		出生日期 Birth Date		[1..1]
8	門診日期	門診日期 OPD Date		[1..1]
9	科別	科別 Department		[1..1]
10	診斷	診斷 Diagnosis	Diagnosis	[1..*]
11	藥品細項	項次 Item		[1..1]
12		處方箋種類註記 Types of Prescription		[1..1]， 一般處方，慢性病連續處方箋 ，特殊等…
13		藥品代碼 Drug Code		[1..1]
14		藥品商品名稱 Brand Name		[1..1]
15		學名 Generic Name		[1..1]
16		劑型 dosage form		[1..1]，如藥丸、藥水等
17		劑量 Dose		[1..1]
18		劑量單位 Dose unit		[1..1]，如：顆、CC等
19		頻率 Frequency		[1..1]
20		給藥途徑 Route of Administration		[1..1]
21		給藥日數 Medication Days		[1..1]
22		給藥總量 Total Amount		[1..1]
23		給藥總量單位 Units		[1..1] ，如：顆、CC、瓶等

項次	區塊描述	欄位名稱	LOINC 對應名稱	欄位說明
24		實際給藥總量 Actual Amount		[0..1]
25		實際給藥總量單位 Units		[0..1]，如：顆、CC、瓶等
26		磨粉註記 powdered		[1..1]，磨粉標註為“Y”，未磨粉標註為“N”
27		註記 Note		[0..1]，空白欄位用於註記說明
28	醫師姓名	醫師姓名 Physician Name		[1..1]
附註說明： (1) [0..*]：此欄位為可選，可重複出現。 (2) [1..*]：此欄位為必要，可重複出現。 (3) [0..1]：此欄位為可選，且只有一次。 (4) [1..1]：此欄位為必要，且只有一次。				

6.1.3 資料庫建置

我們採用 AppServ 2.5.10 套件包[14]來設置我們的電子病歷資料庫，其中包含了安裝 Apache 2.2.8，PHP 5.2.6，MySQL 5.0.51b，phpMyAdmin-2.10.3，讓我們很方便的架設伺服器以及資料庫。

根據用藥記錄的欄位格式我們建立一張資料表，其中包含所有電子病歷欄位，欄位標籤，欄位雜湊值，欄位版本編號以及 T 和 Miu。而 T 和 Miu 分別提供伺服器計算證明中的 TP=(τ, μ)存放位置，如圖 6 所示。

表格 5 資料庫中的使用者資料表欄位

欄位名稱	欄位意義
UserID	使用者帳號
Password	使用者密碼
Username	使用者姓名
Gender	使用者性別
PersonalID	使用者身分證字號
Birthdate	使用者出生日期

圖 6 資料庫中的電子病歷資料表欄位

另外我們也建立使用者資訊的資料表，包括使用者帳號，密碼，姓名，性別，身分證字號以及出生日期，如表格 5 所示。提供我們實現使用者登入系統以及使用者登入後可以觀看與他相關的病歷資料。

6.2 我們的電子病歷系統實作

這節我們會詳細的介紹我們電子病歷系統中每個網頁的功能，如圖 7 所示，其中黑色實線框的網頁是放在使用者端；紅色虛線框的網頁是放在伺服器端。接下來我們會依功能性分成四個部分一一說明。

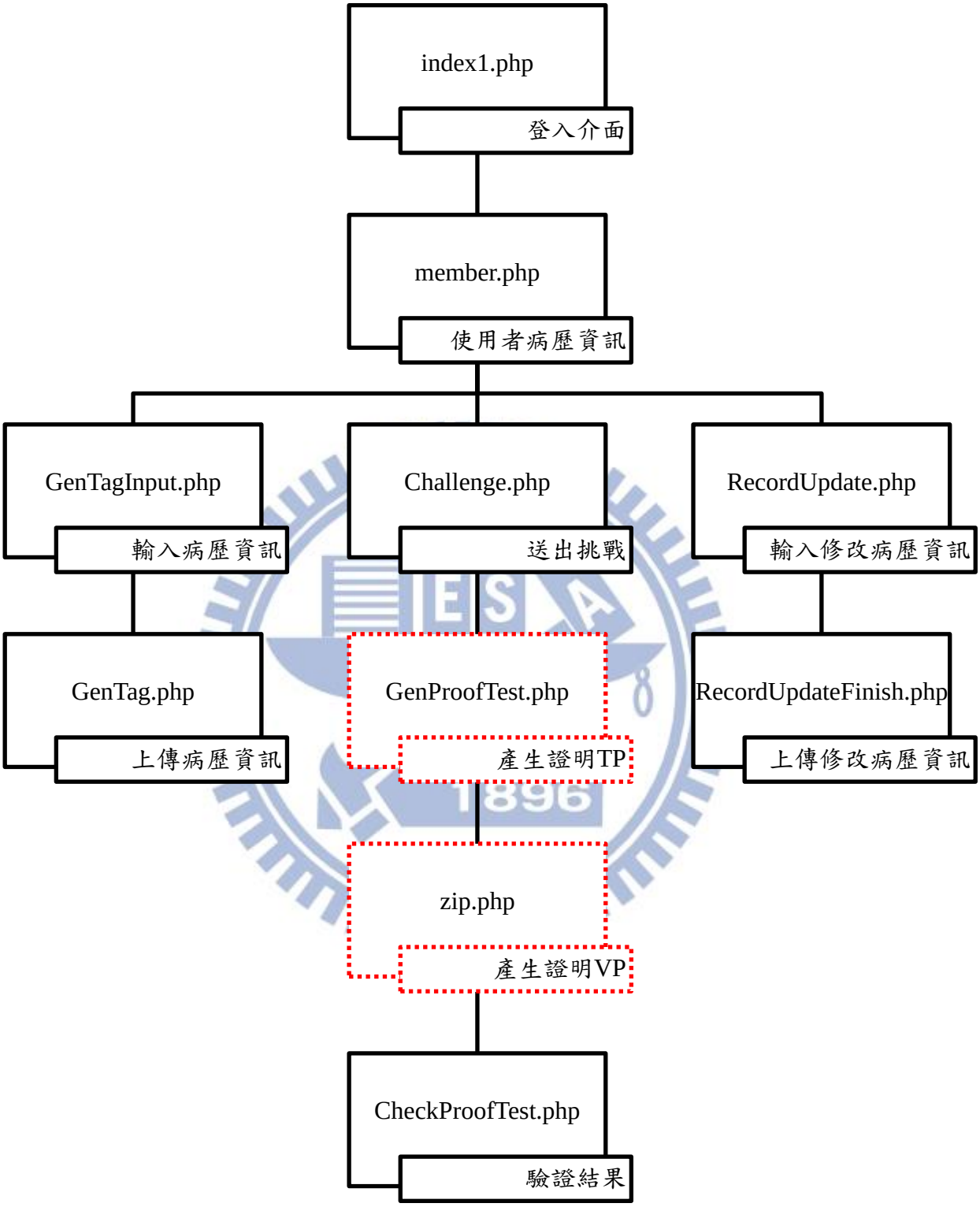


圖 7 電子病歷系統架構圖

6.2.1 登入介面以及使用者病歷資訊

若是第一次使用，使用者在 index1.php 的介面可以註冊自己所屬的帳號，隨後即可透過輸入帳號密碼登入觀看自己的電子病歷。此外，我們要求使用者在網頁資料夾中的 key.txt 設定自己的機密金鑰sk_h以及機密係數α。

登入後，在 member.php 我們會透過 SQL 語法去找出與使用者相符的電子病歷並且呈現在網頁上，使得使用者可瀏覽自己的電子病歷，另外我們也在這個頁面中提供三個功能：新增病歷，送出挑戰以及修改病歷，如圖 8 所示。

圖 8 使用者病歷資訊

6.2.2 新增病歷

使用者點選新增病歷後會進入 GenTagInput.php，在這個網頁使用者可輸入自己的用藥記錄資訊。輸入完之後點選送出則會進入到 GenTag.php。在 GenTag.php 根據使用者輸入的病歷會執行我們以 C++撰寫的 GenTag.exe。GenTag.exe 會根據輸入的病歷以及使用者設定的 key.txt 產生對應的標籤以及雜湊值，另外 GenTag.exe 也會透過雜湊值計算出 MHT 的根節點h_{root}並且將他寫入 RootValue.txt 中。接著我們會透過 SQL 語法將病歷資訊，標籤以及雜湊值一起上傳至伺服器當作一筆電子病歷儲存，如圖 9 所示。

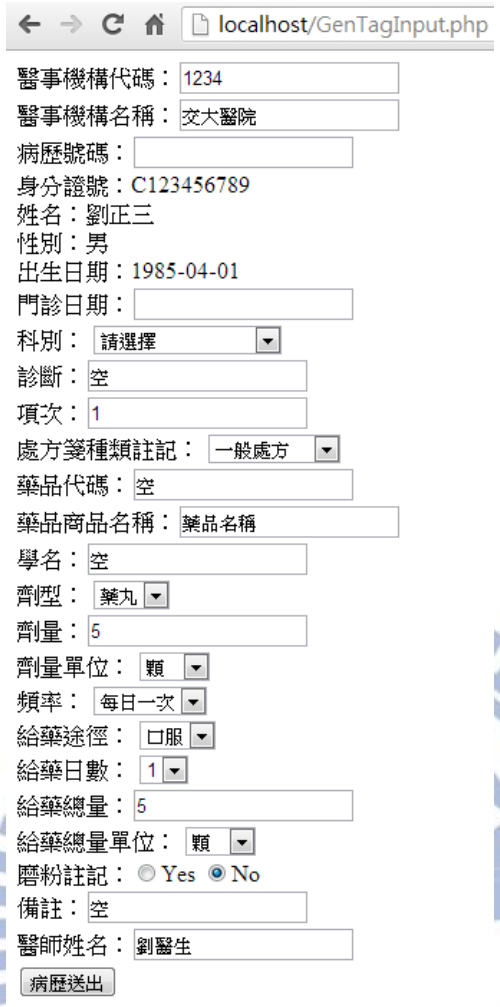


圖 9 新增病歷

6.2.3 送出挑戰及驗證結果

使用者點選送出挑戰後會進入 Challenge.php，在這個網頁中使用者可以輸入產生挑戰的參數 K 以及選擇要驗證哪幾個欄位資訊，如圖 10 所示。

圖 10 送出挑戰

點選送出進入 GenProofTest.php 後，伺服器會透過使用者的個人資料讀取出所有相符的電子病歷，再根據挑戰參數 K 產生證明 P。第四章提到伺服器的證明 P=(VP,TP)，伺服器會在 GenProofTest.php 執行我們以 C++撰寫的 GenProof.exe。根據所選的病歷欄位索引，所選的病歷欄位資訊以及挑戰參數 K 產生 TP=(τ, μ)並且將τ, μ各自儲存在每筆病歷的 T 以及 Miu 欄位，提供使用者驗證使用。產生 TP 之後，伺服器會執行 tree_GenProof.exe。根據所選的病歷欄位索引以及病歷欄位雜湊值計算出 VP 後依照每一筆病歷的病例號碼寫入 ClientVerifyInput_病歷號碼.txt(例如: ClientVerifyInput_000001.txt)檔案中。

為了讓使用者拿到 VP 的資訊，接著我們會進入 zip.php 將所有的 ClientVerifyInput_病歷號碼.txt 壓縮成一個壓縮檔提供使用者下載。使用者將壓縮檔解壓縮到網頁資料夾後，即可點選驗證的選項，如圖 11 所示。

圖 11 壓縮證明檔案

進入到 CheckProofTest.php 後，會執行以 C++撰寫的 tree_verify.exe，透過 ClientVerifyInput_病歷號碼.txt 提供的 VP 計算出根節點h'_{root}與 RootValue.txt 中的h_{root}比對是否相同。接著會執行 CheckProof.exe，透過 SQL 語法得到的 TP=(τ, μ)以及 key.txt 中所儲存的機密金鑰sk_h以及機密係數α當作參數，算出σ之後與τ比對是否相同。若兩者皆驗證通過，則輸出驗證成功訊息；若其中一個驗證失敗，則輸出驗證錯誤訊息，如圖 12 所示。

圖 12 驗證結果

6.2.4 修改病歷

使用者點選修改病歷後，在 RecordUpdate.php 中我們會先透過 SQL 語法讀取出先前使用者輸入的病歷資訊，並且提供使用者作修改的動作。使用者修改完點選送出後會進入 RecordUpdateFinish.php。系統先檢查使用者修改了那些欄位，並且從伺服器取回對應的版本號，隨後透過執行 vcheck.exe 檢查伺服器傳來的版本號是否正確。確認版本號無誤後將版本號自動加 1，根據使用者修改的病歷資訊以及 key.txt 中的機密金鑰sk_h以及機密係數α當作參數執行 modify.exe 產生新的標籤以及雜湊值。最後透過新的雜湊值產生新的 MHT 的根節點h_{root}寫入 RootValue.txt 中，並且將修改後的病歷資訊，新的標籤，新的雜湊值以及新的版本號上傳到伺服器中，如圖 13 所示。

圖 13 修改病歷

6.3 實驗方法與結果分析

6.3.1 實驗目的

我們提出的方法是利用 HMAC 的方式來檢測資料的完整性，我們針對系統中所使用的五個執行檔執行一筆病歷的結果來測量它們執行時間以及記憶體使用的情況，來分析我們的方法是不是有效率的。

6.3.2 實驗結果與分析

實驗結果如表格 6 所示。可以看出所耗費的時間都非常短，時間消耗最長的 GenTag.exe 倘若執行 1000 次也才花費大約 1.6 秒的時間。至於 GenTag.exe 以及 tree_GenProof.exe 為何相對其他程式所花費的時間較高，我們推測是因為在這兩個程式執行時還會建造 MHT 來產生雜湊值以及計算出根節點h_{root}，所以花費時間會再多一些。

在記憶體消耗的部分，每支程式消耗大約 3MB 的記憶體空間，以現今的記憶體空間大小，我們相信不會對使用者端或伺服器端的電腦效能造成太大影響。

表格 6 時間消耗及記憶體消耗

執行檔案	時間消耗(ms)	記憶體消耗(KB)
GenTag.exe	1.653	3,080
GenProof.exe	0.0156	2,892
tree_GenProof.exe	1.268	2,896
tree_verify.exe	0.256	2,900
CheckProof.exe	0.0125	3,010

七、可搜尋的對稱式加密系統實作

第一章前言部分我們提到雲端儲存蓬勃發展，使用者為了方便管理以及節省購買硬碟的成本，漸漸地會將個人的資料儲存在雲端上。為了保護其機密性，儲存在雲端中的資料皆以加密方式保護。使用者上傳資料後，日後如果想搜尋在雲端上它儲存了哪些資料，由於資料已經加密過，並不能直接對密文搜尋。因此[7]提出的 Searchable Symmetric Encryption(SSE)就是設計一套架構，在不洩漏資料的情形下，使得我們能在加密的資料中作搜尋的操作。

7.1 我們會介紹為何我們要參考 SSE 的架構以及 SSE 的演算法，7.2 則會說明我們實作上所需要用到的工具，7.3 則會介紹我們系統的架構，7.4 介紹我們系統的介面以及操作方式，最後 7.5 說明我們實驗方法以及結果分析。

7.1可搜尋對稱式加密方法

我們是參考 R. Curtmola 提出的 SSE-1，SSE-1 與先前研究比較如表格 7 所示。我們可以發現在 SSE-1 的伺服器計算時間只有 O(1)而先前的研究都是 O(n)，在實際運用中 SSE-1 會比較有效率，因此我們採用 SSE-1 來實作關鍵字搜尋的加密系統。接著 7.1.1 會介紹所使用的符號以及意義；7.1.2 會概略介紹 SSE；7.1.3 會詳細的介紹 SSE 的演算法。

表格 7　SSE 與先前研究比較

Properties	[9]	[10]	[11]	SSE-1
Server computation	O(n)	O(n)	O(n)	O(1)
Server storage	O(n)	O(n)	O(n)	O(n)
Number of rounds	1	1	1	1
Communication	O(1)	O(1)	O(1)	O(1)
Adaptive adversaries	no	no	no	no

7.1.1 符號表示

令$\Delta=\{w_1,\ldots,w_d\}$為 d 個關鍵字的字典，則我們可以得到所有可能的文件集合大小為2^{Δ}，此外我們假設在Δ中的所有關鍵字長度最多以 p 位元表示。

令 id(f)為文件 f 的識別號，以及文件集合 F={$f_1, f_2, \ldots, f_n$}。D(w)代表一個含有關鍵字 w 的文件清單，其中清單是儲存以字母順序編排的文件識別號，也就是說 D(w)={ id(f_1),id(f_2),…,id(f_n)}。

給定一個陣列 A，用 A[i]表示在第 i 個位置的元素；addr(A(x))則表示 x 在陣列 A 中是第幾個位置。

7.1.2 SSE 架構概述

首先利用鏈接串列L_i來儲存每一個關鍵字$w_i \in \Delta'$所產生的 D(w_i)，其中每一條L_i的節點存放著在 D(w_i)中的文件識別號。接著將所有鏈接串列L_i中的所有節點經過隨機產生的金鑰加密後以隨機順序的方式儲存在陣列 A 中。這邊使用了一個技巧，我們將第 j 個節點加密儲存在陣列 A 之前，我們會連帶把第 j+1 個節點解密所需要的金鑰以及第 j+1 個節點在陣列 A 中的位置一起與第 j 個節點的資訊加密後再儲存到陣列 A。透過這種方式，使用者只要給予伺服器鏈接串列L_i第 1 個節點的解密金鑰以及第 1 個節點在陣列 A 中的位置，伺服器便能解出所有在L_i的節點資訊。因此另外建立了速查表 T 使得使用者可以存取和解密每一個鏈接串列L_i的第 1 個節點。在 T 中對於每一個關鍵字$w_i \in \Delta'$儲存一對(address,value)。其中 address 是每一個關鍵字$w_i \in \Delta'$在 T 中的位置，而 value 來儲存每一個關鍵字$w_i \in \Delta'$所產生的鏈接串列L_i的第 1 個節點在陣列 A 中的位置以及第 1 個節點的解密金鑰，此外 value 還會以w_i代入 pseudo-random function 產生的值加密。

使用者根據尚未加密的文件集合 F 計算出陣列 A 以及速查表 T，並且將 A，T 和加密過的文件集合 F 一起上傳至伺服器。當使用者想要取出含有關鍵字w_i的文件，使用者計算出w_i在 T 中的位置以及對應 value 的解密金鑰，將這兩個值傳給伺服器。伺服器利用這兩個值解出w_i存放在 T 中的 value 得到鏈接串列L_i的第 1 個節點在陣列 A 中的位置以及第 1 個節點的解密金鑰，因為L_i的每一個節點在陣列 A 中的元素都有包含下一個節點的資訊，因此伺服器能透過此種方式得到所有在 D(w_i)的文件識別號。

7.1.3 SSE 架構介紹

接著我們要介紹 SSE 的架構，它是由四個多項式時間的演算法組成。

令 k,l 為安全參數以及(G,Enc,Dec)為一個 semantically secure symmetric encryption。我們還會用到一個 pseudo-random function f 和兩個 pseudo-random permutation π和φ，它們的參數如下：

$$f: \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^{l+\log_2 m}$$

$$\pi: \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^p$$

$$\varphi: \{0,1\}^k \times \{0,1\}^{\log_2 m} \rightarrow \{0,1\}^{\log_2 m}$$

- KeyGen(1^k,1^l)：由使用者執行的機率性的金鑰產生演算法，它產生隨機的金鑰 s,y,z$\in_R \{0,1\}^k$，並且輸出 K=(s,y,z, 1^l)。

- BuildIndex(K,F)：由使用者執行的索引產生演算法，輸入為機密金鑰 K 以及文件集合 F。演算法可以分為三個階段：

  - 初始化：

    掃描文件集合 F 擷取出其中的關鍵字 w，根據每一個獨立的關鍵字 w $\in \Delta'$建立 D(w)，並且將每一 D(w)都用一個鏈接串列 L 儲存，其中 L 裡的每一節點存放含有關鍵字 w 的文件識別號。另外再初始化一個全域的計數器 ctr = 1，這個 ctr 是為了隨機產生每一個節點在陣列 A 中的位置。

  - 建造陣列 A：

    對於每一個關鍵字 $w_i \in \Delta'$我們建立鏈接串列L_i，其中$N_{i,j}$代表在L_i的第 j 個節點，接著：

    - 產生$sk_{i,0} \in_R \{0,1\}^l$。
    - 對於$1 \leq j \leq |D(w_i)|$：
      - (i) 產生$sk_{i,j} \in_R \{0,1\}^l$以及設定$N_{i,j}$=(id($F_{i,j}$)|| $sk_{i,j}$||$\varphi_s(ctr+1)$)，其中 id($F_{i,j}$)代表在D(w_i)中的第 j 個文件識別號。
      - (ii) 計算$Enc_{sk_{i,j-1}}(N_{i,j})$並且存在 A[$\varphi_s(ctr)$]。
      - (iii) ctr=ctr+1;
    - L_i的最後一個節點($N_{i,|D(w_i)|}$)我們會設置他的下一個節點的位置是NULL，也就是$N_{i,|D(w_i)|} = (\text{id}(F_{i,|D(w_i)|})||\, sk_{i,|D(w_i)|}||\text{NULL})$。讓伺服器解到最後一個節點時，讀取到 NULL 得知不需要再繼續解密下去。

令$m' = \sum_{w_i \in \Delta'} |D(w_i)|$。如果$m' < m$，將陣列 A 中剩餘的$(m - m')$個位置寫入與現有$m'$個位置長度一樣的隨機值。使得陣列 A 中皆是密文且沒有解密金鑰的情況下，沒有辦法得知哪些位置是真正含有資訊的。

- 建立速查表 T：

對於每一個關鍵字 $w_i \in \Delta'$，我們計算 value=(addr(A($N_{i,1}$))$\| sk_{i,0}$)$\oplus f_y(w_i)$，並且設置 T[$\pi_z(w_i)$]=value。

如果$|\Delta'| < |\Delta|$，將速查表 T 中剩餘的$(\Delta - \Delta')$個位置寫入隨機值。

- 輸出索引 I=(A,T)。

- Trapdoor(K,w)：由使用者執行的後門演算法，根據輸入的關鍵字 w 以及機密金鑰 K 輸出後門$T_w = (\pi_z(w), f_y(w))$。

- Search(I, T_w)：由伺服器執行的搜尋演算法，根據輸入的索引 I 以及後門T_w，可以得到含有 w 的所有文件識別號。作法如下：

  - 令(γ, η)= T_w，透過γ取出$\theta = T[\gamma]$。接著令(α||sk)= $\theta \oplus \eta$。

  - 利用α提供在陣列 A 的位置以及 sk 提供解密 A[α]所需要的金鑰，伺服器可以解密出串列 L 的第一個節點，再利用第一個節點得到的下一個節點的位置 A[α_{next}]以及解密金鑰sk'，解出下一個節點的資訊直到最後一個節點。

  - 輸出儲存在 L 中的所有文件識別號。

7.2 使用工具設定

在這節會介紹如何設定我們使用的工具，包括先前的 GMP 函式庫，Crypto++函式庫以及 Alchemy API。由於我們實作的系統介面是用 JAVA 呈現，我們也會說明 JAVA 的設定。

7.2.1 Alchemy API

SSE 的架構中一開始需要先對所有文件擷取出其中的關鍵字，我們利用 Alchemy API[15]來完成此項工作。Alchemy API 是一套分析自然語言的雲端軟體，它可以分析網

頁內容或是一般的文件內容，我們是採用其中關鍵字擷取的功能來達成我們的目標。使用 Alchemy API 必須要到它們的網站 http://www.alchemyapi.com 註冊一個免費的 API 金鑰(免費的版本一天只能使用 1000 次)，並且下載 Alchemy 的 SDK，在我們的系統是採用 JAVA 版本的 SDK。

7.2.2 JAVA 環境設定

使用 JAVA 撰寫程式必須先到 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下載 JDK，開發者可依照自己的作業系統選擇下載檔案。安裝完成後，在 windows 7 作業系統中使用者必須設定環境變數才能使用 JAVA。

在"我的電腦"右鍵點擊"內容"後，點選左方的進階系統設定後點選"環境變數"選項。接著在"系統變數"選項編輯其中的"Path"，在"變數值"選項中新增 JDK 的 bin 資料夾位置(例如：C:\Program Files\Java\jdk1.7.0_21\bin)並以分號；作區隔。另外在"系統變數"的選項新增"CLASSPATH"，在"變數值"選項中新增 JDK 以及 JRE 的 lib 資料夾位置(例如：C:\Program Files\Java\jdk1.7.0_21\lib;C:\Program Files\Java\jre7\lib;)，同樣地以分號；作區隔。最後透過"命令提示字元 CMD"輸入"javac -version"檢查是否能得到 JDK 的版本資訊，如此一來即完成 JAVA 的環境設定，如圖 14 所示。

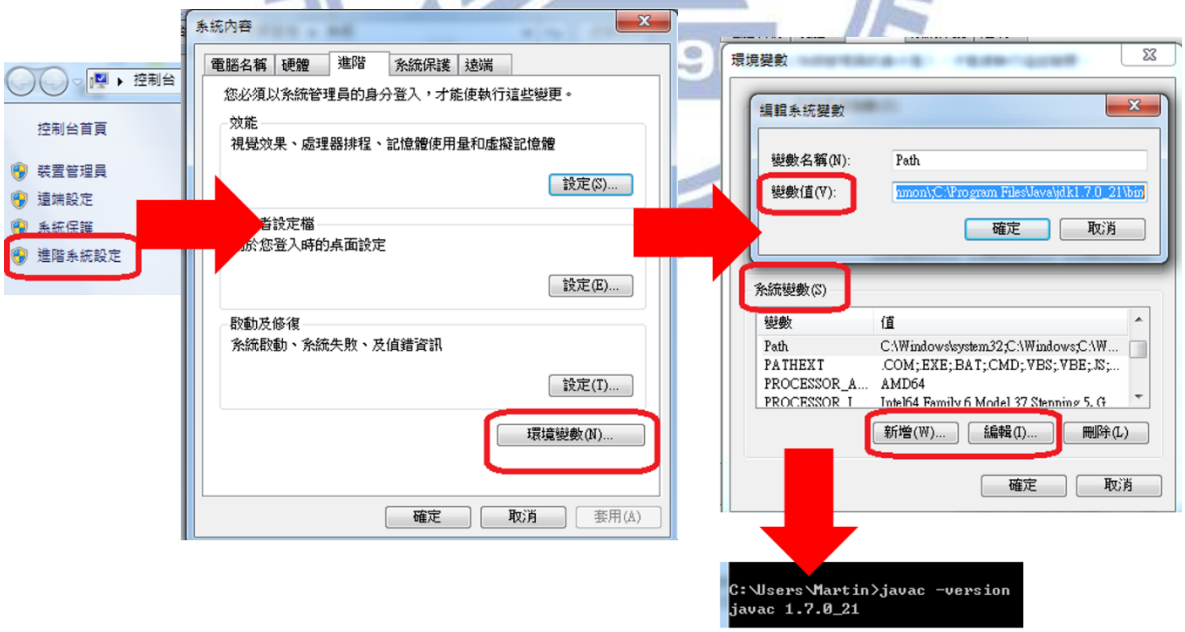


圖 14 JAVA 環境設定

7.2.3 Crypto++函式庫

Crypto++函式庫[16]是一個免費的 C++函式庫，它提供許多密碼學的演算法，例如對稱式加密 AES、DES，雜湊函數 SHA-1，非對稱式加密 RSA 等等。我們呼叫 Crypto++函式庫來實作我們的密碼工具，包括對稱式加密我們採用 AES 以及 pseudo-random function 我們採用 HMAC-SHA1。接下來我們說明如何在 visual studio 2005 設定使得我們可以呼叫 Crypto++函式庫。

從 Crypto++網站 http://www.cryptopp.com/#download 下載最新版的壓縮檔(Crypto++ 5.6.2)後，解壓縮之後，設定方法如下(如圖 15 所示)：

(i) 點選 cryptest.sln 打開專案後，建置裡面的四個專案後，在 Crypto++ 5.6.2 資料夾中的 Win32\output\debug 可以得到 cryptlib.lib

(ii) 開啟新的工作專案(例如：sse)，在”專案”選項點選”sse 屬性”後，在左邊點選”組態屬性”中的”C/C++”底下的”一般”之後，點選右邊的”其他 Include 目錄”，新增 Crypto++ 5.6.2 的資料夾位置。接著點擊” C/C++”底下的”程式碼產生”後，在右邊的”執行階段程式庫”的選項更改為”多執行緒偵錯 (/MTd)”。最後在左邊點選”組態屬性”中的”連結器”底下的”一般”之後，點選右邊的”其他程式庫目錄”，新增 cryptlib.lib 的資料夾位置。如此一來，完成了新的工作專案設定。

(iii) 若程式碼需要使用 Crypto++時，除了 include 所需要的標頭檔之外，還必須多加” #pragma comment(lib, "cryptlib.lib") ”，使得成式能順利使用 Crypto++函式庫。

圖 15 Crypto++函式庫設定

7.3 SSE 實作系統架構

我們的系統架構大致上可分成兩個階段，第一部份是初始化階段，此階段使用者選取要上傳的文件作加密並且建立伺服器搜尋所需的陣列 A 以及速查表 T，將加密文件，陣列 A 以及速查表 T 上傳至伺服器。第二部分是搜尋階段，使用者輸入關鍵字經由系統產生出搜尋標記(search token)後送給伺服器，伺服器根據搜尋標記找出對應的加密文件提供使用者下載，如圖 16 所示。

7.3.1 初始化階段

使用者事先須設定 key.txt 以及 password.txt，password.txt 是加密文件所需要的金鑰而 key.txt 是 SSE 中所需要的金鑰。使用者選取哪些文件上傳之後，我們系統會透過

Alchemy API 擷取出其中的關鍵字集合並且將關鍵字與其關聯的文件名稱以及文件識別號寫入 text.txt，如圖 17 所示。接下來根據 text.txt 在使用者端會執行以 C++撰寫的 construct.exe 建立陣列 A 以及速查表 T 並且分別寫入 array.txt 以及 table.txt 兩個檔案中。array.txt 的大小我們是根據 text.txt 的行數做動態配置，因此 array.txt 的大小與關鍵字個數有關。前面提過 SSE 架構中的後門 $T_w = (\pi_z(\mathrm{w}), f_y(\mathrm{w}))$是搜尋 table.txt 所需要的值，在實作中我們將$\pi(\mathrm{w}) = (z_1 \cdot \mathrm{w} + z_2) \bmod \mathrm{p}$，其中的 p 是找大於 array.txt 的大小的質數，因此 table 的大小與 p 有關。而$f_y(.)$則是套用 HMAC 函數。此外產生 fileID.txt，裡面儲存文件識別號與文件名稱的對應關係，提供使用者下載加密文件時能知道文件名稱。執行完 construct.exe 之後，使用者透過 password.txt 當作金鑰將要上傳的文件以 AES 方式加密，其中加密的文件檔案名稱為文件識別號。最後使用者將加密過的文件，array.txt 以及 table.txt 透過 SFTP 上傳至伺服器完成系統設置。

7.3.2 搜尋階段

使用者在我們提供的介面中輸入關鍵字後，在使用者端透過以 C++撰寫的 index.exe 產生出對應的搜尋標記並且將之送給伺服器。伺服器收到搜尋標記後，查詢 table.txt 得到在 array.txt 中的位置以及解密金鑰，透過先前介紹的方法，伺服器可以解出所有相關的文件識別號回傳給使用者，使用者端接收到文件識別號之後，會透過 fileID.txt 將它們轉換成文件名稱，並且讓使用者選擇要下載哪些文件，經由 SFTP 下載選擇的加密文件之後，使用者端在依據 password.txt 所儲存的金鑰將加密文件解出原文，並且放在以關鍵字為名的資料夾底下。

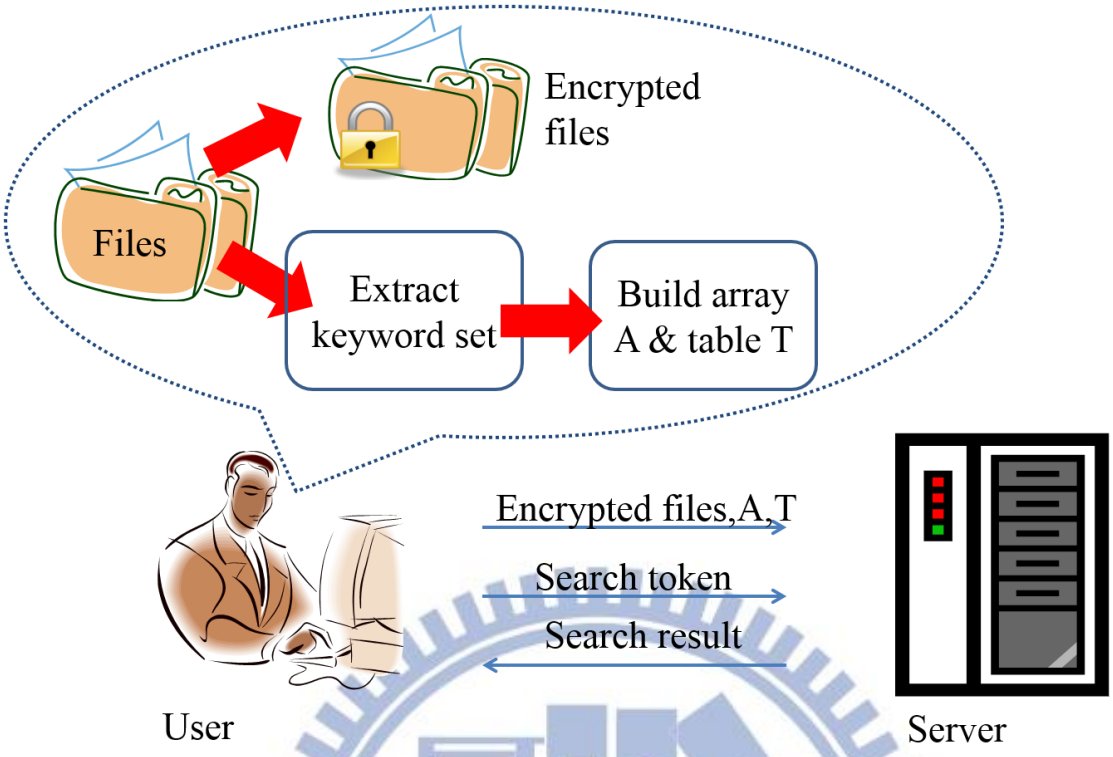


圖 16 實作系統架構

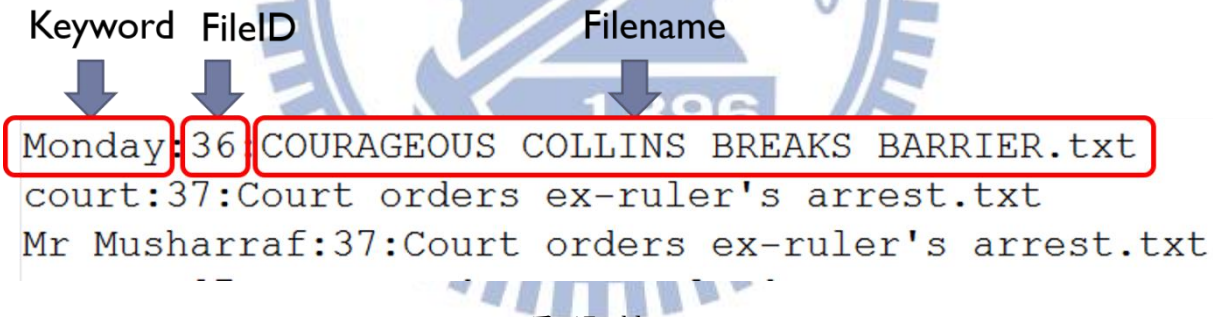


圖 17 table.txt

7.4 使用者介面以及操作方式

使用者介面如圖 18 所示，使用者開啟使用者介面後，等待程式與伺服器連線，連線完成會在下方顯示"Connection successful"。介面中總共有四個按鈕，只要執行其中一個按鈕，最下方的進度表會告知目前選項完成進度。

若使用者點擊"選擇要上傳的檔案"如圖 19 所示，選擇上傳哪些檔案後，按下"Setup"，程式會開始對選取的檔案擷取出關鍵字以及建立陣列 A、速查表 T 並且將選取的檔案作加密，透過 SFTP 將搜尋所需要的資料上傳至伺服器。

上傳完成後，使用者在關鍵字搜尋的欄位輸入關鍵字後點擊”Keyword Search”，系統會將關鍵字製作出一個搜尋標記並將搜尋標記傳給伺服器。伺服器根據搜尋標記，array.txt 以及 table.txt 計算出搜尋結果，並且將搜尋結果回傳給使用者。系統會將搜尋結果顯示在”Search Result”欄位中，接著使用者可以選擇點擊”Download & Decrypt files”，使用者便可選擇要下載與搜尋結果相關的文件，點擊”Download”之後，我們的系統會將解密過的文件放在以關鍵字為名的資料夾底下，如圖 20，圖 21 所示。

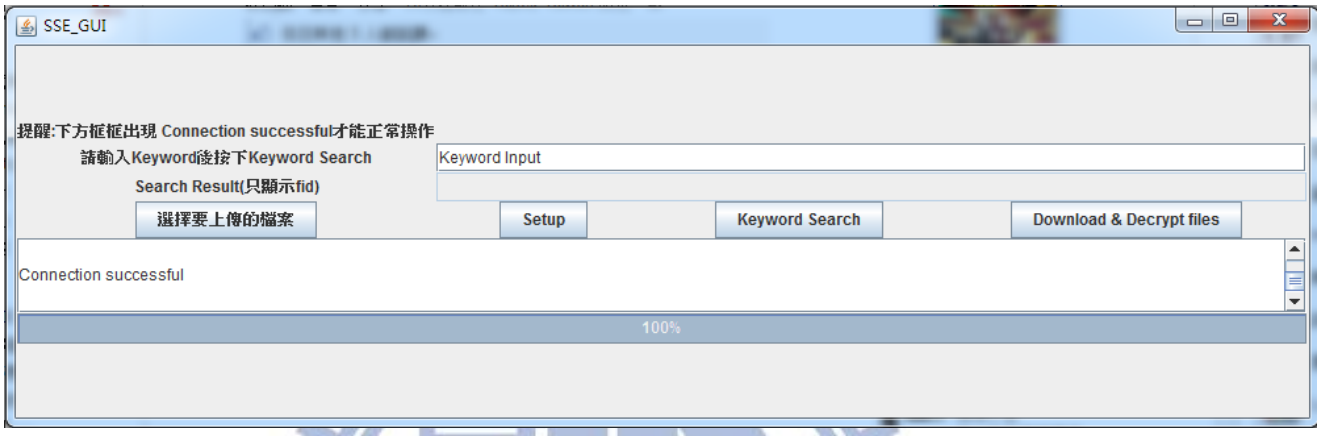


圖 18 使用者介面

圖 19 選擇上傳檔案

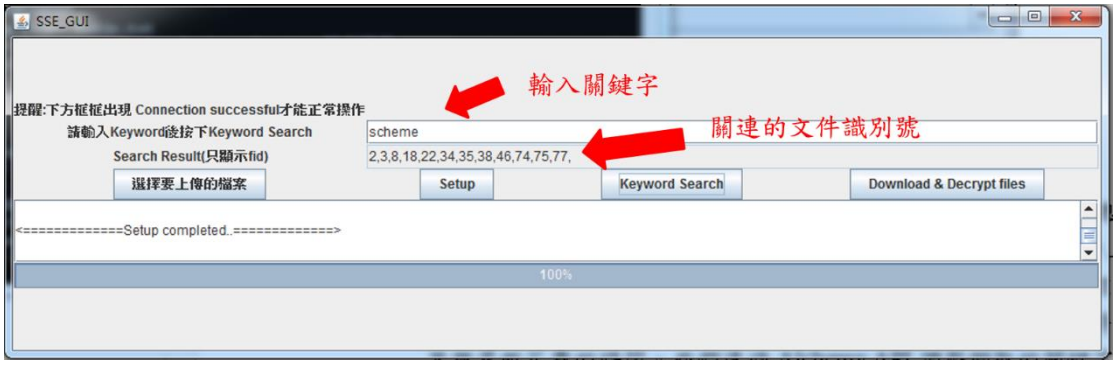


圖 20 關鍵字搜尋

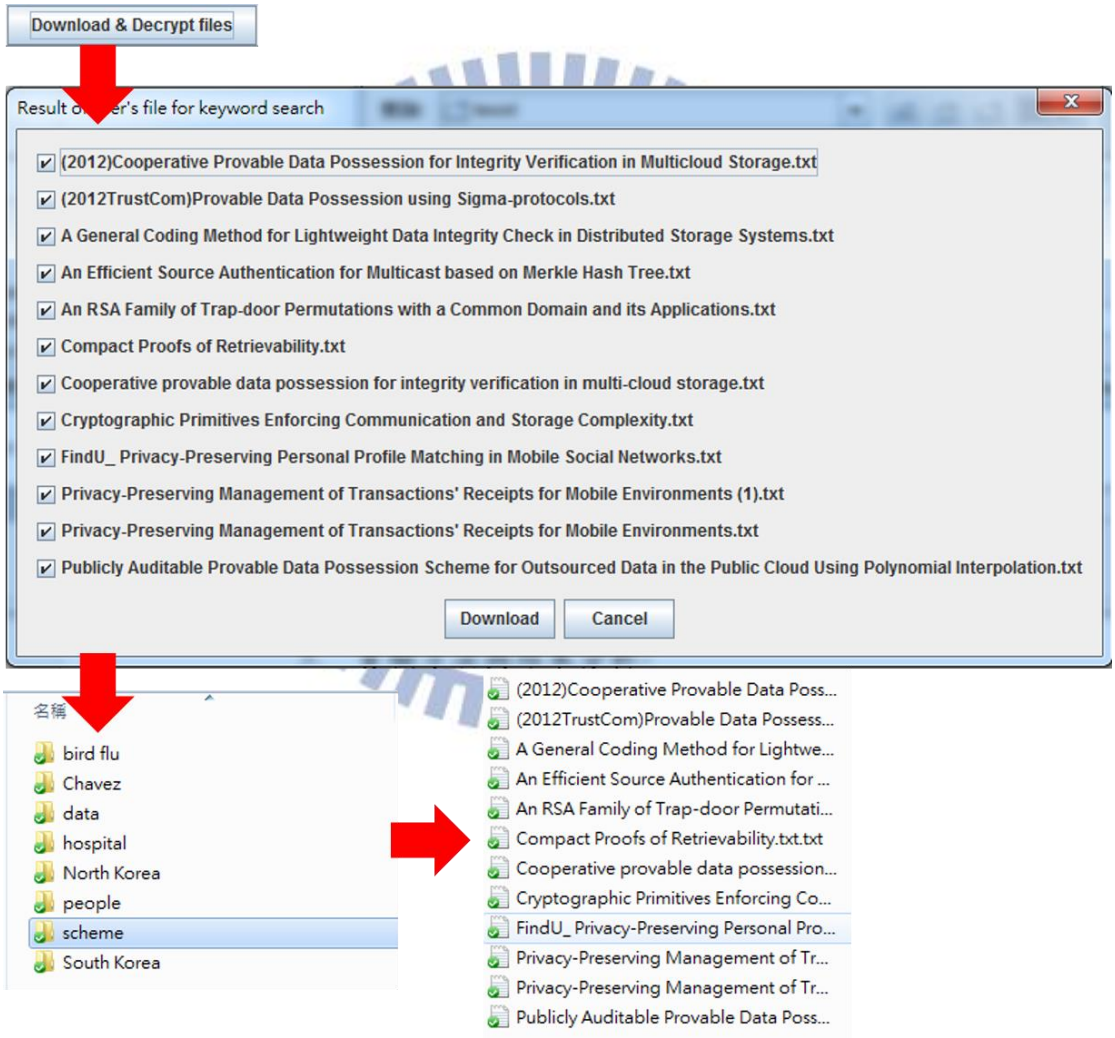


圖 21 下載及解密文件

7.5 實驗方法與結果分析

7.5.1 實驗環境

使用的電腦配備：CPU: intel i5 2.8G, 記憶體: 8G。
使用的作業系統為 windows 7。
測試文件總共 100 篇，合計大小 2.14MB。

7.5.2 實驗目的

由於擷取關鍵字是透過 Alchemy API 將文件上傳至 Alchemy 的雲端伺服器處理，我們不考慮 API 的時間消耗。因此我們只關注在擷取出關鍵字的個數不同的條件下，所建立陣列 A 以及速查表 T 所花費的時間以及伺服器搜尋所花費的時間以及兩者的記憶體消耗比較。

7.5.3 實驗結果及分析

實驗結果如表格 8 所示。我們可以看出產生的關鍵字種類個數越少，Setup 與搜尋的時間消耗和記憶體消耗都會變少。原因是因為不同的關鍵字種類個數，寫入 array.txt 的資料量也不同，一旦關鍵字種類變少，寫入 array.txt 的資料也會減少。此外我們發現 Setup 部分，程式執行的大部分時間都是消耗在寫入檔案 array.txt 以及 table.txt 的時間；搜尋部分由於伺服器收到搜尋標記後會將 array.txt 以及 table.txt 讀取至搜尋程式中，因此大部分的執行時間都消耗在讀取檔案。 因此 array.txt 以及 table.txt 的資料量越少，則 I/O 的時間越短，同時記憶體使用也越少。

表格 8 SSE 時間消耗與記憶體消耗

一個文件平均擷取關鍵字個數	100 個文件產生關鍵字個數(包括重複)	關鍵字種類個數	Setup 時間消耗(秒)	Setup 記憶體消耗(KB)	搜尋時間消耗(秒)	搜尋記憶體消耗(KB)
30	3852	3134	6.910	12,570.6	0.530	4,100.1
20	2116	1728	3.619	8,912.9	0.452	3,878.9
10	1112	959	1.794	6,684.7	0.421	3,784.7

八、討論

8.1 電子病歷系統中使用者對象以及金鑰管理

在我們實作的電子病歷系統，使用者對象是病人本身，他們所儲存的是個人的健康記錄資訊。倘若我們的方法要應用在醫院的電子病歷系統上，使用者對象則改為醫生本身。此外我們的系統使用者的金鑰是儲存在 key.txt 中，若應用在醫院的系統上，可以將醫生的金鑰儲存在醫生卡之中，醫生將病人的診斷記錄輸入系統後，透過醫生的機密金鑰對病人的診斷記錄作標籤一起儲存在電子病歷系統中。

8.2SSE 系統的伺服器搜尋

我們的系統中伺服器每次接收到使用者送來的搜尋標記便會讀取 array.txt 以及 table.txt 以完成搜尋的動作。由先前的實驗結果我們知道讀取檔案的 I/O 時間是搜尋時間的瓶頸。因此如果伺服器執行搜尋操作時，能事先讀取 array.txt 以及 table.txt 的資訊放至記憶體中，每當有搜尋請求便可馬上計算搜尋結果。相信如此一來搜尋時間可以大幅降低，這個部份我們希望未來可以朝這點作改進。

8.3SSE 系統的限制

我們系統在擷取文件的關鍵字是採用 Alchemy API 來幫我們完成，目前只有針對純文字且內容是英文的文件才有辦法擷取關鍵字。因此希望後續我們能找到更合適的關鍵字擷取工具或是自行開發以支援更多種格式的文件，例如 word、pdf、html、email 等等，同時能支援中文語系的文件。

其次，我們的系統目前只能給單一使用者操作，因此未來希望能改進我們的系統，使我們的系統可以支援多個使用者操作。除此之外，若是使用者想要新增新的文件或是修改伺服器上的文件，目前的做法必須要拿所有的文件重新建立新的 array.txt 以及 table.txt，而這種做法會消耗大量的時間以及網路頻寬。因此希望未來我們能提出新的方法使得我們的系統在使用者想要新增新的文件或是修改伺服器上的文件的情況下，能有效率地更新。

九、結論

我們提出了 HMAC 的方式提供使用者可以有效率地驗證伺服器中的電子病歷的完整性，除此之外，我們的方法也可以有效率地支援修改病歷的操作。我們也將我們提出的方法實作出一個電子病歷系統，讓使用者可以新增個人的病歷資訊並且提供使用者驗證上傳的病歷資料完整性。最後做實驗來量測我們的方法執行時間。

在第二部份我們參考 SSE 的架構實作一套能讓使用者在加密檔案上作關鍵字搜尋的系統，我們撰寫使用者介面提供使用者可以上傳加密的文件至伺服器，並且可以對加密的文件作關鍵字搜尋，最後讓使用者選擇下載與搜尋結果相符的加密文件。我們透過實驗發現影響搜尋時間的最大因素在於 I/O 存取。

十、參考文獻

[1] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song, “Provable Data Possession at Untrusted Stores,” Proc. 14th ACM Conf. Computer and Comm. Security (CCS’07), pp. 598-609, 2007.

[2] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. SecureComm, 2008.

[3] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In ASIACRYPT, 2008.

[4] C. Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia, “Dynamic Provable Data Possession,” Proc. 16th ACM Conf. Computer and Comm. Security (CCS ’09), 2009.

[5] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li, “Enabling public auditability and data dynamics for storage security in cloud computing,” IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 5, pp. 847–859, 2011.

[6] Kan Yang and Xiaohua Jia. "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing." (2012): 1-1.

[7] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Proceedings of the 13th ACM conference on Computer and communications security (pp. 79-88). ACM.

[8] Ralph C. Merkle, “Protocols for Public Key Cryptosystems,” Proc. IEEE Symp. Security and Privacy, pp. 122-133, 1980.

[9] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searching on encrypted data. In IEEE Symposium on Security and Privacy, pages 44–55, May 2000.

[10]Eu-Jin Goh. Secure indexes. Technical Report 2003/216,IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

[11]Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In Applied Cryptography and Network Security Conference, 2005.

[12]GMP-5.1.1 The GNU Multiple Precision Arithmetic Library

http://gmplib.org/

[13]GMP Install Instruction for Windows Platform

http://www.cs.nyu.edu/exact/core/gmp/

[14]AppServ open project

http://www.appservnetwork.com/

[15]Alchemy API

http://www.alchemyapi.com/

[16]Crypto++ Library

http://www.cryptopp.com/