允許範圍查詢的分散式 STL 密文容器與演算法

黃柏嶽　　丁培毅　　吳宗杉　　邱垂邦
國立台灣海洋大學
10357018@ntou.edu.tw　pyting@ntou.edu.tw　ilan543@gmail.com　00257034@ntou.edu.tw

摘　要

本文結合可保存明文順序關係的「次序保存加密機制」與 C++的標準樣板函式庫，在分散式元件架構下提出一組輕量化、具有高度安全性的輔助設計元件，在符合標準樣板函式庫的應用模型下擴充其容器與演算法，使得當容器內儲存密文資料時，遠端伺服器上運作的容器物件能夠支援以明文為基礎的範圍查詢與各種資料比較的操作，目標是希望在程式開發者熟悉的模型下引入具有高度安全的自動加解密機制，開發出來的系統實際運作時僅需額外的加解密時間與互動。

關鍵字：次序保存加密、C++ STL、遠端程序呼叫

一、介紹

隨著網路通訊技術與硬體虛擬化技術的進步，各種雲端運算應用技術與概念快速地演進，由伺服器層級的功能性合作運算 (例如：MapReduce[1] 資料比對與探勘)、各種層級虛擬化技術、雲端資料儲存技術、到可配合物聯網發展的輕量級虛擬化技術 Docker [2]等等，當這種輕量化的趨勢與成熟的分散式元件技術[3] 以及服務導向架構 (Service-oriented architecture)[3] 結合時，大量創新的應用快速地推出，由於參與儲存與運算的伺服器以動態方式組成，資料與運算安全的重要性快速地提昇，雖然過去在系統設計層面上安全性常常被歸類為非功能性的需求，然而藉由可証明安全的加密/簽章/認證技術，可搜尋的加密機制[4]及安全委託運算[5]逐步發展成為設計安全雲端應用的基礎，安全技術與分散式技術的密切結合成為系統開發的核心關鍵。

在分散式的元件運作模型中，各個元件能夠透通地在遠端伺服器上運作，為了維護機密資料的安全性，以一般加密方法處理過的資料雖然可以安全地儲存和傳輸，但是運算能力強大的雲端伺服器對於密文資料卻無法進行任何有意義的運算，本文運用可保存明文順序關係的「次序保存加密機制」(order-preserving encryption, OPE)[8, 9, 10, 11]，結合 C++ 語言中標準樣板函式庫 (standard template library, STL)[6] 的運作模型，提出一組輕量化、具有高度安全性的一般化系統設計元件，在符合 STL 的運作模型下擴充 STL 的容器以及演算法，當容器內儲存密文資料時，遠端伺服器上運作的物件能夠支援以明文為基礎的範圍查詢與各種資料比對操作。

「次序保存加密機制」是一種確定式 (deterministic) 的對稱式 (symmetric) 加密機制，這種加密機制的特性是任何人可以由兩個密文直接得到對應明文的大小順序，使得雲端伺服器可以在不解密的情況下有效率地查詢明文在某範圍內的所有資料，組織與分配相關的資料儲存與運算。在使用對稱式加密的應用環境中（例如委外的資料庫），雲端伺服器並沒有加密與解密金鑰，確定式的加密方法雖然會洩漏明文間相等的關係，在明文具有一定亂度 (entropy) 的情況下還是具有相當的安全性，所換得的好處是可以讓被委託的雲端伺服器在不解密的情況下直接比對密文是否相等來線性地搜尋資料。次序保存加密的概念在 Boldyreva[8] 探討時並沒有提出有效率且安全的實作方式，反而提出在一般實用的加密概念下不可能實作的證明，其後有一系列的研究[9, 10, 11]，然而一直有安全性的問題[7]，直到 Popa[7] 才藉由互動式的加密方法換得一種密文會變動的 mOPE (mutable order preserving encoding) 機制，此機制可以達到理想上選擇明文攻擊下的次序保存明文不可分辨 (IND-OCPA)[8] 安全性，但是系統運作時密文的大小隨著全部密文數量增多而逐漸增大，OPE 機制有許多資料庫相關的應用[12, 13, 14]，主要用來處理密文資料庫的 SQL 查詢，以及安全的雲端應用服務[15]。

STL 函式庫中包含兩種類型的容器物件，第一種是序列的容器包括 vector、 deque 以及 list，在使用這一類的容器時，使用者希望容器運用指定的順序來儲存資料，配合 find 演算法可以線性地搜尋指定的資料，前兩者配合 sort 演算法可以排列順序，排序過的容器可以運用 binary_search 演算法來快速搜尋指定的資料，也可以運用 lower_bound 以及 upper_bound 演算法查詢指定範圍的資料；第二種是關聯式的容器包含 map、set、multimap、multiset 等等，關聯式容器需要指定元素之間比較大小的函式物件以便內部以平衡搜尋樹的方式儲存，也提供資料比對的介面 find 和範圍查詢的介面 lower_bound, upper_bound。不論是用何種容器儲存 mOPE 密文時，指定關鍵字的密文來查詢資料的介面內部就是直接比對密文，如果替所儲存的密文建立二元搜尋樹就可以快速地以密文的比對來查詢指定密文，但是範圍的查詢介面就不能用原本的方式運作，需要運用 mOPE 密文所保存的明文順序資訊來查詢才有意義，資料的委託客戶想要查詢明文 m_1 以及 m_2 之間的所有資料時，只需要提供 m_1 以及 m_2 對應的密文，執行容

器物件的伺服器就可以將範圍內所有的密文資料傳送回來，除了基本的範圍查詢外，STL 還有 next_permutation 以及 make_heap 等等演算法也需要伺服依照明文的順序處理資料才具有意義。本文主要的貢獻在於審視 STL 函式庫的功能，擴充與明文順序相關的介面，將 mOPE 的互動式加密機制以分散式元件的機制實作，期望能夠建立出符合原本 STL 運作模型的密文容器與操作演算法，提供建立安全雲端應用時基層的抽象化元件。

第二節首先敘述「密文可變的次序保存加密機制」– mOPE 互動式加密機制，其次說明 STL 函式庫中相關的容器操作介面與演算法，第三節提出運用 mOPE 擴充的密文容器 OPEmap 及 OPEvector 之介面函式與參數化演算法之實作，第四節為相關設計問題之討論，第五節為結論。

二、背景知識

密文可變的次序保存加密機制 – mOPE

mOPE 加密機制的運作環境和概念有別於一般傳統的加密機制，傳統的加密機制在一個私密的運算環境下產生密文，每一個明文對應的密文在時間和空間的改變下不會有任何改變；mOPE 機制下每一個明文對應的密文隨著一個伺服器上公開的資料結構的改變而變動，產生密文必須透過使用者和伺服器間的互動程序。以下藉由範例說明 mOPE 之運作機制，假設有五個明文 5、10、15、20、25，以 Enc(5)、Enc(10)、Enc(15)、Enc(20)、Enc(25)代表用任意安全的確定式加密演算法（例如 AES 加密機制）加密的密文，這些密文看起來像是亂數，其大小順序與原本明文的大小順序完全不同，mOPE 讓不知道明文的伺服器僅僅得知明文間的順序關係。如圖一所示客戶端把這五個明文加密後透過互動式的程序上傳至伺服器，伺服器建立一個表達明文大小關係的 mOPE 編碼樹及一個 mOPE 密文表。

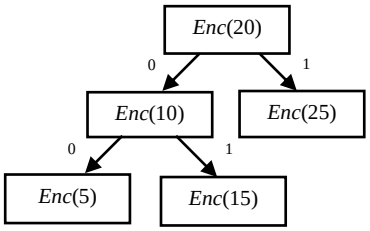

圖 1 mOPE 編碼平衡樹

為了降低新增密文時客戶端與伺服器的互動次數，同時降低搜尋密文時比對的次數，以及 mOPE 編碼的長度，伺服器需要以 B-tree 演算法建立一個平衡的搜尋樹，在搜尋樹中每一節點 v 左側節點的值都小於 v，右側節點的值都大於 v；平衡樹則在新增或刪除任何節點時維持最小深度。此加密機制中每一個密文包含 Enc(m)以及對應的 mOPE 編碼，由於 B-tree 在新增或刪除節點時需要局部調整節點的位置，亦即 Enc(m)對應的 mOPE 編碼隨著客戶端後續加密上傳的資料而逐漸改變，因此稱為密文可變的加密機制。

密文	編碼([路徑]10..0)
Enc(20)	[]100 = 4
Enc(10)	[0]10 = 2
Enc(25)	[1]10 = 6
Enc(5)	[00]1 = 1
Enc(15)	[01]1 = 3

表 1 mOPE 密文表

圖 1 以二元平衡樹為例，每個節點存放確定式加密的密文 Enc(m)，這些密文皆由同一把客戶端的密鑰加密，伺服器與客戶端互動式地依照明文 m 的大小順序將密文 Enc(m)放入 mOPE 編碼樹中，任一節點 Enc(m)左側分支裡的節點對應之明文都小於 m，右側分支裡的節點對應之明文都大於 m。圖 1 中標示由根節點至每一個節點的路徑，每一節點左側分支標示為 0，右側分支標示為 1，每一密文節點對應的編碼為：

mOPE 編碼 = [路徑]10. . . 0

編碼的長度為 mOPE 編碼樹的深度，中括號代表密文節點的路徑，路徑之後補一個 1 位元及適當長度的 0 位元，例如 Enc(15)的路徑為 01，則 mOPE 編碼為 011（十進位數值 3），如表 1 的 mOPE 表格，每一列為密文 Enc(m)以及對應的 mOPE 編碼，此兩者即為 mOPE 機制中 m 所對應的密文，為了方便新增與刪除密文，mOPE 的密文表實作時以密文為關鍵字，建立平衡之二元搜尋樹。

mOPE 機制包含四個主要功能：

1. 新增
2. 刪除
3. 查詢
4. 範圍查詢

「新增」功能將客戶端上傳的密文 Enc(m) 儲存在伺服器端並且指定其 mOPE 編碼，「刪除」功能將客戶端指定的密文刪除，執行此兩項功能時也會影響到少數相關密文的 mOPE 編碼，「查詢」功能直接比對密文表以尋找客戶端指定的密文，mOPE 機制特別著重於「範圍查詢」功能 – 客戶端指定 $Enc(m_1)$及 $Enc(m_2)$，伺服端將所有符合 $m_1 \le m < m_2$ 的密文 Enc(m)傳回。

新增

新增密文時，客戶端以對稱式加密演算法 Enc(·)加密關鍵字明文 m 得到密文 c=Enc(m)，傳送 c 至伺服器，伺服器收到密文後與客戶端執行「互動式 mOPE 編碼樹巡訪演算法」，配合平衡樹的平

衡演算法得到 c 在 mOPE 編碼樹中的正確位置，根據節點的路徑指定 c 的 mOPE 編碼，並且調整在平衡過程更動過的節點的 mOPE 編碼。

「互動式 mOPE 編碼樹巡訪演算法」如下:

1. 伺服器首先拜訪 mOPE 編碼樹的根節點
2. 伺服器將目前所在節點的密文 c'傳給客戶端
3. 客戶端收到 c'用密鑰解出 m'，比較 m 和 m'的大小，將比較結果回傳給伺服器（如果想要加入的訊息 m > m'則傳回 1，m ≦ m'則傳回 0）
4. 伺服器收到訊息如為 0 則移至左節點，1 則移至右節點，如目前所在節點不存在(找到 c'的正確位置)的時候就停止，否則重複步驟 2

以圖 1 為例，如果需要新增一明文 23，客戶端加密 23 後將 Enc(23)傳給伺服器，伺服器接到密文後，開始與客戶端互動，伺服器把根節點 Enc(20)傳給客戶端，客戶端解密得知 23 > 20，並把比較結果 1 傳回給伺服器，伺服器把右節點 Enc(25)傳給客戶端，客戶端解密後把比對結果-1 傳回伺服器，此時 Enc(25)的左節點不存在，伺服器以密文 Enc(23)作為 Enc(25)的左節點，接下來伺服器執行平衡樹的平衡演算法。伺服器將 c 加入表一的 mOPE 密文表中並且更新其它平衡時有變動位置的密文之 mOPE 編碼。

刪除

當客戶端要求刪除密文 c 時，把 c 傳給伺服器，伺服器在 mOPE 密文表中尋找密文 c，由其對應的 mOPE 編碼取得 mOPE 編碼樹中密文節點之路徑，刪除該節點，平衡 mOPE 編碼樹，更新位置有變動的節點在 mOPE 密文表中的 mOPE 編碼；如果找不到 c 則回傳錯誤訊息。

例如在圖 1 中客戶端要求刪除密文 Enc(15)，將密文傳給伺服器，伺服器在表 1 中找到 Enc(15)，Enc(15)對應的 mOPE 編碼的路徑 01 對應到 mOPE 編碼樹的節點 Enc(15)，把這個密文從 mOPE 編碼樹以及 mOPE 密文表中刪除。重新平衡 mOPE 編碼樹，更新位置有變動的節點在 mOPE 密文表中的 mOPE 編碼。

查詢

客戶端傳遞密文 c 給伺服器，伺服器在 mOPE 密文表中尋找密文 c，並且傳回 c 對應的相關訊息，如果找不到 c 則回傳錯誤訊息。

範圍查詢

客戶端上傳兩個密文 c1=Enc(m1)、c2=Enc(m2)給伺服器，查詢符合 m1≤m<m2 範圍所有的密文 Enc(m)，伺服器查詢 mOPE 密文表中 c1、c2 的 mOPE 編碼，得到兩個對應節點的路徑，運用中序(in-order)查訪演算法將 mOPE 編碼樹在這兩個節點之間所有節點的密文以及相關資料傳回客戶端；如果 c1 或是 c2 不在 mOPE 密文表中則回傳查詢失敗。

以圖 1 及表 1 為例，假設 c1=10, c2=25，客戶端將 Enc(10)及 Enc(25)傳給伺服器端，伺服器查詢 mOPE 密文表得到它們在 mOPE 編碼樹中的路徑，以中序查訪將密文編碼是 2 到 6 之間(不包含 6)的節點資訊傳回，亦即 Enc(10)、Enc(15)、Enc(20)。

C++ STL 函式庫

以下介紹本文擴充的兩個 STL 容器 map、vector 以及與比較大小相關的 STL 演算法，擴充 set 與擴充 map 的考量點接近，擴充 deque 與擴充 vector 的考量點相近。

map

map 是參數化的關聯式容器，主要有兩個參數是 key 的型態以及 value 的型態，這兩個型態可以是任意的，但是 key 需要能夠以 operator<() 比較順序，或是提供第三個參數是一個比較用的函式物件，map 容器內部的架構是一個儲存 key 欄位的平衡搜尋樹，應用程式中主要的運用模型是以 key 作為快速存取 value 的索引，為了說明本文擴充 map 容器來存放密文所需要的功能，表二中列出 map 容器類別中幾個與 key 比較大小相關的介面函式：

介面函式	說明
erase()	移除符合 key、指定位置、或是指定區段的元素
find()	尋找符合 key 的元素
insert()	將元素(key, value)加入容器
lower_bound()	傳回 ≥key 元素的起始迭代器
upper_bound()	傳回 <key 元素的結尾迭代器

表 2 map 容器與密文範圍查詢相關的介面

vector

vector 是參數化的循序容器，主要參數是一個 value 的型態，可以是任意型態，不需要能夠比較大小，儲存資料時不自動排列所有資料，因此不提供與比較大小或是順序相關的介面函式，可以運用 insert()或 push_back()在容器中加入元素或是在序列尾端加入元素，也可以運用 erase()或 pop_back()來刪除容器中指定位置、指定區段或是序列尾端的元素；線性的查詢需要透過 STL 演算法 find()來進行，排序需要運用 sort()演算法，分割需要運用 partition()演算法，排序過的資料可以運用 binary_search()進行快速的二分搜尋，也可以運用 lower_bound()取得大於或是等於指定 value 的元素的起始迭代器，運用 upper_bound()可以取得小於指定 value 的元素的結尾迭代器，可以運用 max_element()或是 min_element()取得最大或是最小的元素，可以運用 make_heap()/push_heap()/

pop_heap()/sort_heap()來建立及操作堆積(heap)物件，也可以運用 next_permutation()取得下一個可能的排列，上面這些動作當然可以運用密文的順序來完成，但是得到的結果通常沒有太大意義，因此需要運用 mOPE 機制設計儲存密文的循序容器。

遠端程序呼叫

遠端程序呼叫(Remote Procedure Call, RPC)[16,17]，是一個電腦通信協定，允許執行於一台電腦的程式呼叫另一台電腦的程式，程式設計者無需額外地替這個互動編寫特殊化的程式碼，可以讓資料與程序透通式地在遠端伺服器上運算，讓運算以及儲存可以分散於不同伺服器上執行，可以透過它快速地建構分散式的應用系統以支援分散式的元件模型。在遠端(伺服器)存在可以執行的服務元件，在近端(客戶端)透過代理物件執行遠端程序，本文中客戶端與伺服器的互動與資料傳輸都建立在這樣的機制上。

三、系統建構

以下運用次序保存加密的主要特性建構能夠在遠端運作的密文儲存容器與配合之演算法，在維持 STL 設計的基本樣式前提下，擴充必要的界面。

3.1 OPEmap

儲存 mOPE 密文的 OPEmap 容器需要兩個容器，一是如圖 1 的 mOPE 編碼平衡樹，一是如圖 2 的 mOPE 密文表。在 OPEmap 中的 mOPE 編碼平衡樹的每個節點存放著兩個參數，一個是關鍵字密文 key，一個是主要密文 value，而密文表中的每個節點存放著關鍵字密文 key 和 OPE 編碼 encoding，加速密文查詢並且對照其 mOPE 編碼的表格，如表 1 的密文表，考慮新增、刪除、與搜尋的需求以二元搜尋樹實作。Map 是參數化的型態，主要包含兩個參數：key 和 value，關鍵字的密文為 key，對應資料的密文為 value，第三個參數 compare 預設是一個比較 key 大小的函式物件，在此密文容器中我們設計一個 OPE_compare()取代。

在 mOPE 機制中伺服器完全不知道 key 欄位所對應的明文，無法自行比對兩個密文的大小，所以明文的順序必須由客戶端提供，伺服器將兩個打算比較的關鍵字密文送回客戶端，由客戶端依照明文的順序告訴伺服器端比較的結果，因此另外設計一個 OPE_compare 函式來實現伺服器與客戶端互動式的比較。為配合整個 mOPE 機制運作，我們也擴充其它函式，但客戶端應用模型基本上與原本 STL 中的 map 相同。

型態參數 OPE_compare 函式

伺服器執行 OPE_insert 在密文中新增密文時需要呼叫此函式，透過 RPC 機制在客戶端執行，輸入兩密文 c 與 c'，其中 c 是客戶端欲存入的密文，c' 是伺服器在新增過程中需要比較的密文，客戶端執行的函式本體需要客戶端儲存的密鑰 sk 以解密兩密文，如果 Dec(c, sk) > Dec(c', sk)回傳 1，否則回傳 0。

```
OPE_compare(c, c')
    m ← Dec(c,sk)
    m' ← Dec(c',sk)
    if (m > m')
        return 1
    else
        return 0
```

輔助函式 OPE_encode

伺服器執行 OPE_insert 時需要呼叫此函式以得到欲存入密文 c 的 mOPE 編碼，由伺服器執行，輸入密文 c 在 mOPE 編碼樹中的路徑 path，輸出 c 的 mOPE 編碼：在路徑的二進位編碼後面補一個位元 1，再補 0 到長度 N，N 的大小為樹深度加 1。

```
OPE_encode(path)
    path.append(1)
    while(path.length < N)
        path.append(0)
    return path
```

介面函式 OPE_initialize

在客戶端執行，輸入伺服器 ID，紀錄 ID 於物件中，指定容器物件在名稱為 ID 的遠端伺服器上執行，初始成功回傳 1，反之回傳 0。

介面函式 OPE_insert

在伺服器上執行，輸入關鍵字密文 k 及相關聯的密文 c，其中 k 需為 mOPE 密文以進行範圍查詢，c 則為一般密文，OPE_insert 則將密文存入伺服器端。如下圖演算法所示，首先設定 iter 為 mOPE 編碼樹的根節點，path 為空字串，從樹的根節點開始，以密文 k 及 iter 所指向節點的 key 當作輸入呼叫 OPE_compare(k, iter->key)，把比較結果 t 加入路徑 path 當中，並依照結果在 m_OPE 編碼樹中移動 iter，如果要前往的節點是空的則跳出迴圈，若不是空的則當比較結果 t = 1 時往右節點移動，t = 0 時往左節點移動，重複路徑加入 path.append(t)到 OPE_compare(k, iter->key)之間的動作，下面演算法中省略平衡樹調整相關節點的機制，如果密文 k 和 iter->key 相等時代表關鍵字重複的錯誤。接下來編碼路徑 path 得到編碼 e，把關鍵字密文 k、密文 c、編碼 e，做成 mOPE 編碼平衡樹的節點物件，依照最後一次比較的 t 值判斷要將物件插入 iter 所在位置的左節點或是右節點，最後呼叫 CipherTable.insert(k,e) 把關鍵字密文 k 和 mOPE 編碼 e 新增到密文表中。

```
OPE_insert(k, c)
    iter = mOPEBtree.root
    path = ϕ
    t←OPE_compare(k, iter->key)
```

```
while (iter->key != k)
    path.append(t)
    if (t == 1)
        if (iter->rightnode == NULL)
            break
        iter = iter->rightnode
    else if (t == 0)
        if (iter->leftnode == NULL)
            break
        iter = iter->leftnode
    t ← OPE_compare(k, iter->key)

if (iter->key != k)
    e ← OPE_encode(path)
    if (t == 1)
        iter->rightnode=make_node(k, c)
    else if (t == 0)
        iter->leftnode=make_node(k, c)
    CipherTable.insert(k, e)
    若平衡樹有作平衡的動作此處將會更新密文表的 OPE 編碼
else
    輸出錯誤訊息
```

介面函式 OPE_erase

此函式在伺服器端執行，刪除容器內關鍵字密文 k 以及相關聯的密文 c，利用 mOPE 編碼平衡樹及密文表找出密文 k 並刪除，刪除成功回傳 1，否則回傳 0。整個函式先利用二元搜尋找出 k 在密文表中的位置 cptr，如果不存在則回傳 0，否則以其對應之 mOPE 編碼 e = cptr->encoding 二元搜尋編碼平衡樹，得到密文 k 的節點位置 iter，刪除 cptr 和 iter 所指到的相關資料 (包含相關聯的密文 c)。

```
OPE_erase(k)
    cptr ← CipherTable.binary_search(k)
    if (cptr==0) return 0
    e = cptr->encoding
    iter ← mOPEBtree.binary_search(e)
    delete_from_mOPEBtree(iter)
    delete_from_CipherTable(ptr)
    return 1
```

介面函式 OPE_find

在伺服器端執行，搜尋指定的關鍵字密文 k，利用 mOPE 編碼平衡樹及密文表找出關鍵字密文 k 並傳回相關聯的密文 c，否則回傳 0：利用二元搜尋找出 k 在密文表中的位置 cptr，如果不存在則回傳 0，否則以其對應之 mOPE 編碼 e = cptr->encoding，由編碼平衡樹中得到密文的節點位置 iter，取得相關聯的密文 c。

```
OPE_find(k)
    cptr ← CipherTable.binary_search(k)
    if (cptr==0) return 0
    e = cptr->encoding
    iter ←mOPEBtree.binary_search(e)
    return iter
```

介面函式 OPE_upper_bound

此函式在伺服器端執行，在 mOPE 編碼樹中尋找大於關鍵字密文 k 的節點，首先在 mOPE 密文表中尋找密文 k，根據其 mOPE 編碼找到 mOPE 編碼樹中的下一個節點的位置，如果找不到則必須透過 OPE_compare 與客戶端互動得到 mOPE 編碼樹中明文大小大於 k 的最小節點位置並回傳：函式一開始呼叫 CipherTable.binary_search(k)，如果 cptr 不等於 0，以其對應之 mOPE 編碼 e = cptr->encoding，由 mOPE 編碼樹中得到密文的節點位置 iter，iter++依照中序(in-order traversal)在 mOPE 編碼樹中找到下一個節點；如果 cptr 等於 0，代表此密文不在容器內，需要透過呼叫 OPE_compare(k, iter->key)得到比較結果 t，t = 1 往右節點移動，t = 0 往左節點移動，重複呼叫 OPE_compare 並依照比較結果移動，直到找到最接近的葉節點時停止，如果最後一次比較的 t 值為 0 回傳 iter，t 值等於 1 回傳 iter++。

```
OPE_ upper_bound(k)
    cptr ←CipherTable.binary_search(k)
    if (cptr!=0)
        e = cptr->encoding
        iter ← mOPEBTree.binary_search(e)
        iter++
    else
        iter = mOPEBtree.root
        t←OPE_compare(k, iter->key)
        while (true)
            if (t == 1)
                if (iter->rightnode == NULL)
                    break
                iter = iter->rightnode
            else if (t == 0)
                if (iter->leftnode == NULL)
                    break
                iter = iter->leftnode
            t ← OPE_compare(k, iter->key)
        if (t == 1) iter++
    return iter
```

介面函式 OPE_lower_bound

此函式在伺服器端執行，尋找小於或是等於指定關鍵字密文 k 的密文，如果找到 k，回傳 mOPE 編碼樹中 k 的節點位置，如果找不到則必須透過與客戶端互動得到平衡樹中明文大小大於 k 的最小節點位置並回傳。此函式的實作與 OPE_upper_bound 幾乎完全一樣，如下圖演算法中唯一不同的地方在於如果在 mOPE 密文表中找到指定的密文，直接回傳 mOPE 編碼樹中 k 的節點位置。

```
OPE_ lower_bound(k)
    …
    if (cptr!=0)
        e = cptr->encoding
        iter ← mOPEBtree.binary_search(e)
    else
        …
```

範圍搜尋應用實例

下面是一個在客戶端利用上面的 OPEmap 類別以及擴充的介面函式實現範圍搜尋的例子，先運用 OPE_lower_bound(k_1)介面得到範圍的起始迭代器位置 iter_low，然後透過 OPE_upper_bound(k_2)介面得到範圍的結尾迭代器位置 iter_up，最後利用 getCipher()及 iter++將大於等於 iter_low 且小於 iter_up 的密文取回，並以客戶端的密鑰 sk 解密。

RangeQuery:

```
OPEmap<…> myMap(serverID)
…
iter_low ← myMap.OPE_lower_bound(k1)
iter_up ← myMap.OPE_upper_bound(k2)
for (iter = iter_low; iter != iter_up; iter++)
      keyM←Dec(iter->key, sk)
      valueM←Dec(iter->value, sk)
```

3.2 OPEvector

接下來介紹的是 OPEvector 的設計以及配合的 STL 演算法的設計，這些演算法提供給有需要在 OPEvector 密文容器中以明文排序或執行範圍搜尋的使用者，假設使用者把密文存在遠端未經排序的 vector 中，之後若想要進行密文的範圍搜尋，只需把 vector 作明文大小的排序，就能快速找到範圍之間的密文，後面會講述作明文大小排序的函式。STL 原本 vector 容器主要的參數是元素的型態例如 vector<int>，使用此容器時可以運用 operator[]以註標存取任何一個元素，前述的 OPEmap 中參數包含 key 和 value 的型態，使用時儲存兩個有關聯的密文，然後依照 key 的明文大小建構 mOPE 編碼平衡樹，由於儲存架構的不同，需要調整相關的函式以及介面，但是以保留 vector 原本的應用模型為原則。

vector 的儲存架構不像 map 是樹狀而是序列(如圖 2)，儲存方式是把新加入的資料加到 vector 的尾端或是指定的位置而且不會自動排序，然而為了實現 mOPE 的機制以提供明文排序的選擇，除了需要額外建立依照密文大小排列的二元搜尋樹，也就是 mOPE 的密碼表之外，也需要建立依照明文大小順序排列的 mOPE 編碼樹。

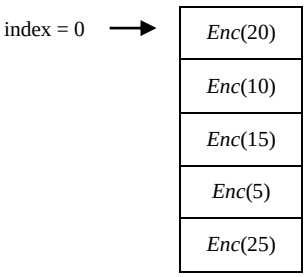

圖 2 data vector

OPEvector 有三個儲存架構：第一是如圖 2 存放密文的 data vector，其中第一筆資料 index = 0、第二筆資料 index = 1，依此類推，密文的先後順序為插入的順序。第二是 mOPE 的編碼平衡樹(圖 1)，但與 OPEmap 不一樣的是每個節點只儲存密文在 vector 中的註標 index，第三是 mOPE 的密文表(表 1)對應的二元搜尋樹，每個節點紀錄的資訊有兩個，密文在 data vector 中的註標 index 及 OPE 編碼 encoding。

基本上 OPEvector 的介面函式與內部參數概念上與 OPEmap 的相同，不過在 mOPE 編碼平衡樹中不直接紀錄密文，而是紀錄密文在 data vector 中的註標，OPE_push_back 或是 OPE_insert 時需要間接地到 data vector 中取得密文，以下是 OPEvector 的擴充函式。

OPEvector 的 OPE_initialize、OPE_compare 和 OPE_encode 函式與 OPEmap 裡的相同這裡我們就不多做贅述。

介面函式 OPE_push_back

在伺服器執行，輸入密文 c，將密文存入伺服器。一開始先初始 mptr 指標為平衡樹的根節點，path 為空字串，從樹的根節點開始，以 mptr 所指到節點的 index 找到 data 中的密文 c'，並以 c 及 c'為參數呼叫 OPE_compare(c, c')，把結果 t 加入 path 中，依照結果 t 移動 mptr，如果要前往的節點是空的則

```
OPE_push_back(c)
    mptr = mOPEBtree.root
    path = ϕ

    c' = data[mptr->index]
    t ← OPE_compare(c, c')

    while (c' != c)
        path.append(t)
        if (t == 1)
            if (mptr ->rightnode == NULL)
                break
            mptr = mptr ->rightnode
        else if (t == 0)
            if (mptr ->leftnode == NULL)
                break
            mptr = mptr ->leftnode
        c' = data[mptr->index]
        t ← OPE_compare(c, c')

    if (c' != c)
        data.push_back(c)
        newIdx = data.size()
        if (t == 1)
            mptr->rightnode = make_node(newIdx)
        else if (t == 0)
            mptr->leftnode = make_node(newIdx)
        e ← OPE_encode(path)
        CipherTable.insert(newIdx,e)
        若平衡樹有作平衡的動作這裡會更新密
        文表的 OPE 編碼
    else
        錯誤訊息
```

跳出迴圈，若不是空的則 t = 1 往右節點移動，t = 0 往左節點移動，並再次呼叫 OPE_compare(c, c')，重複執行 path.append(t)到 OPE_compare(c, c')之間的動作，若密文 c 和 c'相等亦結束迴圈，若是因 c=c' 跳出，則回傳錯誤訊息，若不是，把密文 c push_back()到 data，並記錄密文 c 在 data vector 中的註標，也就是演算法中的 newIdx，依照最後一次比較的 t 值判斷要將新節點插入 mptr 的左節點或是右節點，如果 t = 1 把 newIdx 做成節點插入 mptr 的右節點，如果 t = 0 把 newIdx 做成節點插入 mptr 的左節點，最後把密文 c 在 data vector 中的 index 和 path 的 mOPE 編碼 e 存入密文表中，插入新密文後若 mOPE 編碼樹作平衡的動作，則必須更新密文表裡的 mOPE 編碼。

介面函式 OPE_erase

在伺服器端執行，輸入在 OPEvector 中要刪除的密文位置迭代器 iter，這個迭代器常常是藉由 OPE_find 演算法取得，把 iter 指到的密文*iter 的相關資料從 data vector、mOPE 編碼樹及 mOPE 密文表刪除，回傳 iter 的下一個位置。函式內先利用二元搜尋找出*iter 在密文表中儲存相關資料的位置 cptr，利用 cptr 的編碼欄位 encoding 找到*iter 在 mOPE 編碼樹中相關資料的位置 mptr，最後把 cptr、mptr 及 iter 指到的資料刪除。

```
OPE_erase(iter)
    cptr←CipherTable.binary_search(*iter)
    e = cptr->encoding
    mptr←mOPEBtree.binary_search(e)
    delete_from_CipherTable (cptr)
    delete_from_mOPEBtree (mptr)
    return data.erase(iter)
```

參數化演算法 OPE_find

在伺服器端執行，輸入密文 c，找到 c 在 data vector 中的位置並回傳其迭代器，找不到則回傳 data vector 的結束位置迭代器。初始 iter 為 data vector 的開頭位置迭代器，函式一開始以密文 c 對密文表作二元搜尋得到位置 cptr，如果 cptr 等於 0，回傳 data.end()，如果不等於 0，則利用 cptr->index 得到在 data vector 中的位置迭代器並回傳。

```
OPE_find(c)
    iter = data.begin()
    cptr← CipherTable.binary_search(c)
    if (cptr!=0)
        return iter + cptr->index
    return data.end()
```

參數化演算法 OPE_sort

在伺服器端執行，輸入 OPEvector 的範圍起始迭代器 first 及範圍結尾迭代器 last，依照明文大小排序此區間的密文。函式一開始必須將 data 先拷貝到一個輔助的 vector tmp 中，初始 iter 為 data 中範圍的開頭 first，因為要對平衡樹做 in-order traversal，所以初始 mptr 為 in-order traversal 的開頭位置，從頭開始巡訪平衡樹，並用 iter++以中序的方式往下一個節點移動，若找到的密文其迭代器大小是在 first 和 last 之間則把該密文從 tmp 中取得並覆蓋 iter 所指到的密文*iter，覆蓋後 iter++，當 iter 等於範圍結尾 last 時跳出迴圈。另外一種可行的實作方法則是到 mOPE 密文表中找到 first 及 last 之間的每一個密文的 mOPE 編碼，以此為關鍵值進行排序，再將排序好的密文拷貝回 data vector 的 first 及 last 迭代器的範圍之間。

```
OPE_sort(first, last)
    vector<…> tmp
    for (i=0;i<data.size();i++)
      tmp.push_back(data[i])
    iter = first
    mptr = mOPEBtree.begin()
    while (iter!=last)
      if (data.begin() + mptr->index >= first&&
            data.begin() + mptr->index < last)
        *iter = tmp[mptr->index]
        iter++
      mptr++
```

參數化演算法 OPE_upper_bound

在伺服器端執行，輸入密文 c，如果找到 c，回傳 c 在 data vector 中的下一個位置迭代器，如果找不到則必須透過與客戶端互動得到範圍結尾的位置迭代器。在正確執行 OPE_upper_bound()之前，使用者必須執行 OPE_sort 演算法讓 data vector 裡的資料依照明文大小排序，否則呼叫此函式會得到無意義的結果。先在密文表以二元搜尋法找到密文 c 的節點，紀錄在 cptr 中，如果 cptr != 0，則透過 cptr 取得節點的 index 欄位，就可以知道 c 在 data vector 中的位置，將找到的位置加 1 後回傳，如果 iter = 0 則必須透過互動才能得知位置，作法與 OPEmap 的 OPE_upper_bound 相同，以下列出主要差異。

```
OPE_upper_bound (c)
    iter = data.begin()
    cptr = CipherTable.binary_search(c)
    if (cptr != 0)
        iter = iter + (cptr->index +1)
    else
        …
    return iter
```

參數化演算法 OPE_lower_bound

在伺服器端執行，輸入密文 c，如果找到 c，回傳 c 的位置迭代器，演算法與 OPE_upper_bound 大致相同，不一樣的地方是當找到指定的密文時，直接回傳該位置迭代器，不用再向後移動一個位置。

OPEmap 與 OPEvector 的使用模式大致相同，客戶端在使用前都必須先宣告物件，並呼叫 OPE_initialize(ID)將要互動的遠端 ID 輸入到物件中，完成這些動作後便能使用以上的介面函式。

STL 容器 deque 也是一個可以支援排序以及 lower_bound, upper_bound 演算法的循序容器, 所以在擴充為 OPEdeque 時的考量和 OPEvector 類似; 擴充 STL 容器 set 容器的考量點則和 OPEmap 類似,其它相關但是可以用相同設計精神實作的參數化演算法還包括 make_heap、push_heap、pop_heap、next_permutation、prev_permutation 等。

四、討論

設計前一節中擴充的 mOPE 密文容器時，mOPE 密文的儲存結構可以有另外一種相當不同的設計思考,假設有一個伺服器裡面有十個擴充的 mOPE 密文容器,那麼前一節的設計裡十個容器包含十個 mOPE 編碼平衡樹及十個 mOPE 密文表，如果在存入容器時只用一個全域的編碼平衡樹以及一個全域的密文表,這兩種設計有什麼功能上的差別？舉例來說,如果對第一種設計裡同一個遠端的十個密文容器都新增同一份密文,那麼每一個編碼平衡樹以及密文表裡都有同樣的一份密文,但是各自的 mOPE 編碼是不同的，這也意味著任意兩個在不同的 mOPE 密文表裡的密文是沒有辦法比對大小的，雖然在新增密文時，比較小的編碼樹會降低互動的次數，但是就儲存的考量來說，伺服器上存放十份相同的密文卻是相當浪費的,每個密文容器的獨立運作使得伺服器並無法偵測這些重複上傳的密文。如果是第二種設計，由於密文都存放在同一個編碼平衡樹以及同一個密文表中，為了區隔每個密文是存放在哪個密文容器中,所以密文表必須多加一個欄位來標示此密文存放在哪些密文容器中，當我們對不同的密文容器新增相同密文時，因為密文都存放在同一棵編碼樹中，所以能判斷密文是否已經上傳過,並且把重複上傳的密文屬於哪個 map 紀錄到密文表中，這種設計可以為我們省去一些儲存空間,同時密文在容器之間移動時可以完全不需要與使用者互動,也不需要修改編碼樹以及密文表;如果從查詢密文的效率來看，要以某兩個密文進行範圍搜尋時,第一種設計裡可以很快地取得某一個密文容器中範圍的開頭與結尾並獲得此範圍的密文,如果是第二種設計，在查找範圍的開頭與結尾時需要花費較多時間,不管是不是屬於這個密文容器的密文都需要比對,找到範圍的開頭與結尾後，還不能把這範圍內的密文全數取得,不屬於這個容器的密文不需要回傳,需要花時間去判斷,甚至需要額外的資料結構來紀錄密文表中同一個容器內的密文,設計時這些都需要在儲存空間和運算時間上取捨。

五、結論

我們利用次序保存加密機制設計一組可以存放密文的擴充 STL 容器，擴充功能的介面除了基本的插入、刪除、查詢外還附加了範圍搜尋的功能,這些功能都是在儲存的資料已經加密的情況下在伺服器上執行，目標是希望透過 C++ STL 熟悉的運作模型，設計輕量化的雲端安全應用建置工具,以提供相當程度安全性的條件下增加系統設計者的便利性,未來在物聯網的環境下一定有更多應用發展的空間。

參考文獻

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In 6th Symposium on Operating System Design and Implementation, 2004.

[2] https://www.docker.com/

[3] Component-based software engineering, https://en.wikipedia.org/wiki/Component-based_software_engineering

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In Advances in Cryptology-Eurocrypt 2004.pp.506-522.

[5] S. Goldwasser, Y. T. Kalai, G. N. Rothblum. Delegating computation: Interactive Proofs for Muggles. Journal of the ACM, Vol.62, No. 4, 2015.pp.1-27.

[6] The C++ standard library: a tutorial and reference, N.M. Josuttis, 2nd edition.

[7] R. A. Popa, F. H. Li, and N. Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. In Security and Privacy (SP), pp.463 – 477, 2013

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order preserving symmetric encryption. In EUROCRYPT, 2009.

[9] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In CRYPTO, 2011.

[10] D. Liu and S. Wang. Programmable order-preserving secure index for encrypted database query. In IEEE International Conference on Cloud Computing, 2012.

[11] L. Xiao, I.-L. Yen, and D. T. Huynh. A note for the ideal order preserving encryption object and generalized order-preserving encryption. Cryptology ePrint Archive, 2012/350.

[12] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In ACM SOSP, 2011.

[13] D. Liu and S. Wang. Nonlinear order preserving index for encrypted database query in service cloud environments. Concurrency and Computation: Practice and Experience, 2013.

[14] L. Xiao, I.-L. Yen, and D. T. Huynh. Extending order preserving encryption for multi-user systems. Cryptology ePrint Archive, 2012/192.

[15] CipherCloud. Tokenization for cloud data. http://www.ciphercloud.com/tokenization-cloud-data.aspx

[16] OSF DCE RPC.http://www.opengroup.org/dce/

[17] Microsoft RPC. https://msdn.microsoft.com/en-us/library/windows/desktop/ms691207(v=vs.85).aspx