國立交通大學

資訊科學與工程研究所

碩 士 論 文

一個使用 HYL 編碼的可比較大小的加密演算法以及混淆方法

A New Approach for Efficiently Orderable Encryption Using HYL Encoding And Obfuscation

研 究 生：楊其仁

指導教授：曾文貴　教授

中 華 民 國　102　年　6　月

一個使用 HYL 編碼的可比較大小的加密演算法以及混淆方法

A New Approach for Efficiently Orderable Encryption Using HYL Encoding and Obfuscation

研 究 生：楊其仁　　Student：Chi-Jan Yang

指導教授：曾文貴　　Advisor：Wen-Guey Tzeng

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2013

Hsinchu, Taiwan, Republic of China

中華民國 102 年 6 月

一個使用 HYL 編碼的可比較大小的加密演算法以及混淆方法

學生: 楊其仁　　指導教授: 曾文貴

國立交通大學資訊科學與工程研究碩士班

摘要

在資料庫中，range query 是一個很常被使用的搜尋功能，但是隨者資料量的增加，架設一個提供 range query 的資料庫的成本也隨之攀升，最終使得使用者難以負擔。而隨者雲端儲存媒介的發展，使用者便可以在雲端伺服器上架設提供 range query 的資料庫，減少非常多的成本。但是使用外部的儲存媒介就會有安全性上的問題，它們可能會竊取你存在上面的資料，所以你就必須使用能支援 range quey 的加密演算法來保障你資料的安全性，同時也能在加密的資料上執行 range query。而很多提供 range query 的加密演算法在執行 range query 時都必須搜尋整個資料庫，造成執行時間太長。而我們提出一個使用 HYL 編碼的可比較大小的加密演算法，其可比較大小的特性使得伺服器可以直接對密文比較大小，進而建立資料索引來加速 query 的執行，而再使用 bloom filter 的技術來減少比較大小所需時間，雖然安全性會比起上述演算法差，但我們會比較我們的演算法在此類可比較大小的演算法中，在安全性、密文大小和比較大小所需時間上達到不錯的平衡。而我們的演算法仍然存在者一些問題，所以我們使用一混淆方法希望解決這個問題，並用實驗來顯示出我們混淆的效果。

A New Approach for Efficiently Orderable Encryption Using HYL Encoding and Obfuscation

Student: Chi-Jan Yang Advisor: Dr. Wen-Guey Tzeng

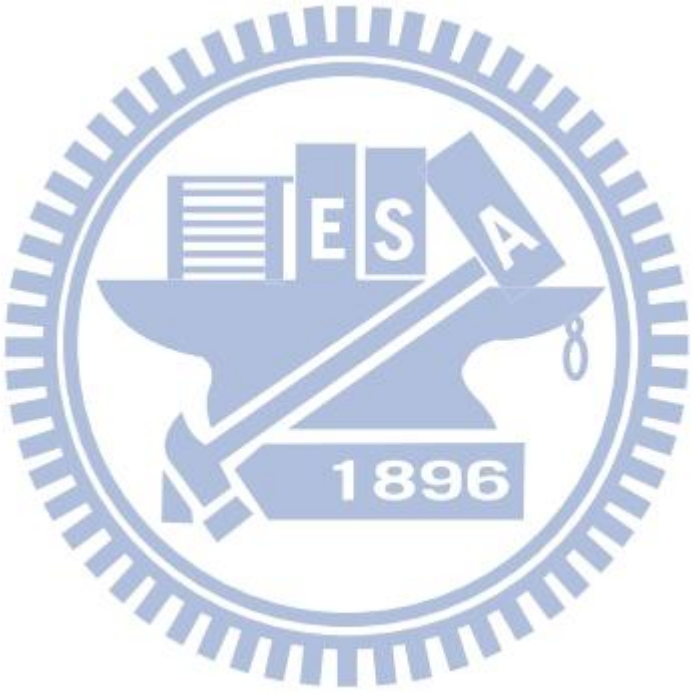
Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

Abstract

Range query is one of the most frequently used search function in database, but the cost of building database server supporting range query grows as the data size increases, finally makes user hard to afford it. With the development of cloud storage, user can build database server on cloud storage, and greatly reduces their cost. But using this outsourced storage has security problem, the outsourced service provider may be curious about hosted data. To protect the hosted data and maintain the ability of executing range query, user must use some specialized encryption scheme. And most of this kind of encryption scheme require scan over the entire database when executing query, which cost a lot of time on it. We propose an efficiently orderable encryption scheme using HYL encoding. Storage server can order the ciphertexts because they are orderable, and then can build index on encrypted data to greatly reduce the time of query executing. We also use bloom filter to reduce the computation time of ordering ciphertexts. Although our scheme has the lower security than above, but we achieve a good balance among security, ciphertext size and comparison time in this kind of efficiently orderable encryption. Our scheme still has some security problem, so we adopt an obfuscation method to enhance our scheme. We also show our obfuscation effort with experiment.

致謝

首先感謝指導教授曾文貴教授兩年來的教導，幫助我提昇自己的學識和報告技巧，並指點我的不足之處。接者感謝實驗室的學長們，能親切的指導我報告的方向以及學術上的問題。再來感謝我的同學們，一起參加比賽、研討會等都是我一聲中的珍貴回憶。接下來感謝我的家人，爸爸、媽媽，給我經濟上的支援讓我得以專心在研究上而不去煩惱生活上的瑣事，還有時常打電話找我聊天的阿姨，而做後要感謝去年過世的外公，從小到大以及這兩年間的照顧，謹以這篇論文獻給您。



目錄

圖片目錄

表格目錄

注釋

一、背景介紹

1.1 問題敘述

資料庫是一個很受歡迎的儲存媒介，使用者常把自己的資料放在資料庫裡面，除了儲存功用外，更重要的是可以對資料庫做些字串搜尋，比較大小等等的搜索功能，但隨者資料量越來越多，維持一個資料庫的成本也越來越高，而在近期隨者雲端的興起，把資料放在雲端上相對來說是一個降低成本的好方法，但這意味者資料會放在外部的伺服器提供者，如何防止資料洩漏給外部伺服器提供者便是一個問題，而現存非常多的加密方法如 AES 等都可去保護資料不被洩漏，但是此種加密方法雖然很安全，但是使得使用者也沒辦法對資料進行前述字串搜尋等功能，所以為了要讓雲端可以執行前述功能同時也保護資料不被竊取，比較常採取的方法是只洩漏一些資訊讓雲端可以執行某些功能，而不洩漏其他資訊，比如[2]利用 deterministic 的加密方法，deterministic 的方法會將相同的明文加密成相同的密文，如此一來便可以透過密文來比較明文是否相等，進而去做字串比對，而其只洩漏明文的相等性。

我們的研究則專注在處理比大小的功能上，資料庫在處理 range query 會需要用到此功能，而這些 query 則是資料庫中很常用到的 query，以比大小的方式來找出特定區段的資料，假設有以下資料如表格 1:

名字	成績
杜蘭特	88
阿克利	56
費南德斯	99
中田翔	64
香川真司	26
李大浩	77
陽岱鋼	47

表格 1:資料表範例

Range query:此資料表放在伺服器上，而當我想找成績大於 80 分的資料時，就傳送”搜尋成績大於 80 的資料 ”這樣的 range query 給伺服器，伺服器就會去比較成績那欄，

並把大於 80 分的資料回傳。 又或者我想找成績落在［50,60］這個區間內的人名(也就是成績大於 50，小於 60)，一樣送出”搜尋成績落在[50,60]的資料 ”這個 query 給伺服器後，便可收到阿克利這一欄的資料。

而對於 range query 來說，資料索引機制可大幅提升 query 執行的速度，所謂資料索引機制是建立索引值，query 進來時跟索引值比較，再根據比較結果去讀取特定範圍的資料，避免每次 query 時都讀取全部資料庫裡的資料，而建索引方法有很多，以下舉例以二元搜尋樹建立索引。

首先伺服器可將資料排序和建立二元搜尋樹索引如表格 2 和圖 1:

名字	成績
香川真司	26
陽岱鋼	47
阿克利	56
中田翔	64
李大浩	77
杜蘭特	88
費南德斯	99

表格 2:排序後資料表

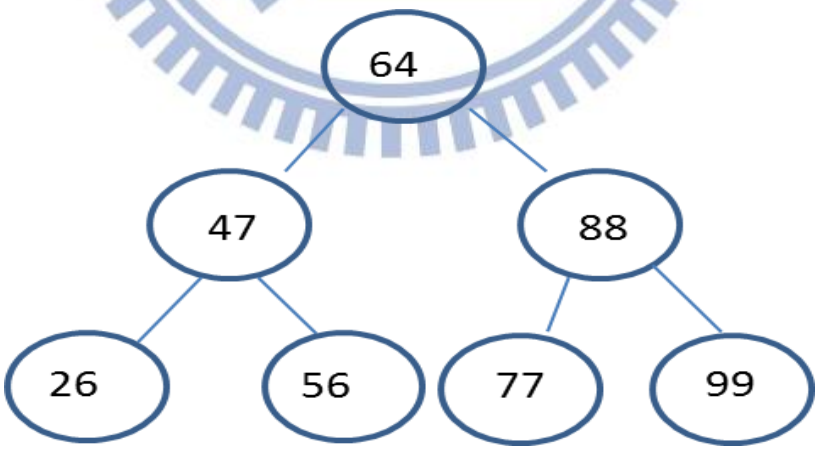


圖 1:二元搜尋樹索引

則 range query 傳進來只要跟這索引比對，比如說跟剛剛一樣搜尋大於 80 分的資料，則只要比較 64、88 和 77 就可以搜尋出結果，比起之前需要全部比較快上許多，而

在此架構下，平均每次 query 所需比較次數為 O(log m)，m 為資料總數，而不論索引架構為何，若想在數字資料上若要建立有效索引就必須讓資料可以比較大小。

1.2 相關研究

而現存可以讓伺服器執行 range query 同時又保有一定的安全性的做法可分為 2 類，一為 Non-Orderable Encryption(簡稱 NOE)，一為 Efficiently Orderable Encryption(簡稱 EOE)，兩者以密文有無洩漏明文大小關係為分類依據，前者無後者有，而兩者的介紹如下。

NOE:

NOE 作法首先使用者在儲存資料時會先加密後再儲存，在做 range query 的要求時會對 query 進行轉換，而伺服器就用這個 query 和加密進行一些特殊的計算，這些計算依照加密方法的不同而有所不同，比如說[8]這篇是比較 query 的集合和密文的集合有無交集，而後回傳符合結果的加密資料，使用者拿到後再解密取得結果，而這過程中伺服器不會拿到任何有關資料的資訊，而由於此方法不允許伺服器直接比較密文大小，所以可以達到比 EOE 更高的安全性，但其缺點則為伺服器無法為加密資料建立索引，在每次 query 時都必須對所有的資料進行運算，雖然[13]這篇針對多維度的數字資料提出一加密方法容許 range query 且又可建立索引，但即使如此該篇的索引效果能比本來索引效果慢了 10 倍，而我們的作法則屬於 EOE 那一類。

EOE

EOE 這種方法會洩漏明文大小關係給雲端知道，也就是說，存在一演算法 W，輸入 2 個密文給 W，W 會判定誰大誰小，而 efficiently 這個詞代表演算法 W 會在多項式時間內完成，使用者透過 EOE 加密後上傳資料，在傳送 range query 也跟 NOE 一樣要進行轉換，但此轉換通常只是加密 range query 上的數字，而伺服器接收到 query 後便利用 W 來對 query 和資料算出結果並回傳，使用者拿到後解密，而由於其洩漏明文大小關係給雲端知道，雲端可有效地建立索引，讓其處理 range query 的速度快上許多，在[10]這

篇提出有提出一種 EOE，其明文和密文皆為單一數字，此種明文和密文皆為一數字，並且 W 是直接比較密文大小的方法又稱 Order Preserving Encryption（簡稱 OPE），OPE 在比較大小上和密文大小可謂最佳，但[10]加密需把明文的機率分布當成輸入，造成實作上的困難，而在[1]這篇則提出一較可實做的 OPE，其密文和明文也皆為單一數字。上述兩者的加密方法其實都是 deterministic 的加密方法，而在[3]這篇則提出一 randomized 的 OPE，也就是相同明文不一定會加密成相同密文，方法為一明文數字對應到某一密文範圍，加密時從範圍者隨機挑一數字，但此舉造成使用者必須記得所有明文對應的範圍，才能做出正確的 query，而以上三者密文皆為單一數字的加密方法，除了洩漏明文大小關係外，也會洩漏一些其他的資訊，如明文間距大小關係:

假設今天看到 3 個明文為 a,b,c，而其經加密後為 3,6,15，則攻擊者可推測 b 和 c 的差距比起 a 和 b 的差距來得大這件事擁有很高的機率，除非密文的範圍為明文範圍的指數次方倍。

之後在[5]這篇有針對[1]這篇的安全性做詳細量化，並提出一安全性更高的加密方法，此方法由一種對稱式加密(如 AES)和一經 hash 函數產生的索引值所組成，此 hash 函數的產生會需要所有可能出現的明文才能產生，舉例:

假設明文為{1000,2000,4000,8000}，則此 hash 函數會把 1000->0 2000->1 4000->2 8000->3，除此以外的輸入皆輸出空元素，而 4000 的密文就為(2,AES(key,4000))，比較大小時只需比較索引值大小即可，但需要所有明文當輸入使得跟[10]一樣難實作。

以上的方法都沒有達到 EOE 理想的安全性 IND-OCPA(詳細解釋見注 1)，而在[6]這篇則有提出一加密方法，其密文不再為單一數字而為一集合，而安全性可達到 IND-OCPA，除了明文大小關係外不會洩漏其它的資訊，但是它的密文集合的元素數量和比較大小所需時間皆為 O(n)，n 為明文範圍大小，也就是光 8 bit 數字(總數 256)，一個單一數字所成的密文就是一 256 個元素的集合，令其難以實作。

但其將密文變成一集合的手法是一個不錯的想法，似乎比起以單一數字為密文可以有更好的安全性，而後我們從[7]這篇中找到了一種 coding 技巧 HYL encoding([7]中的 0-1 encoding)，將數字轉化成集合，並透過比較兩集合是否有交集來比大小，且集合

大小為 O(log n)，比較時間為 O((log n)^2)，使用 bloom filter 加速後為 O((log n)*k)個 hash 函數，符合我們對時間和空間的要求。

我們的貢獻在於提出以 HYL encoding 為主構成的加密技巧，此方法會洩漏明文大小關係給雲端知道，方便讓其建立索引，使得每筆 query 平均比較次數為 O(log m)，m 為資料總數，而密文為一集合和兩 bloom filter，集合大小為 O(log n)，而比較大小所需時間為 O((log n)*k)個 hash 函數，安全性比[1]篇好，最後我們還嘗試使用 RS encoding 技巧來對密文集合增加 r 個新元素以克服使用 HYL encoding 所造成的安全性問題，再以實驗去顯示克服效果，而下表格 3 為比較。

	密文形式	比較演算法複雜度	安全性
[1]	單一數字	O(1)	低
[6]	一集合，大小 O(n)	$O(n^2)$個字串比對	高
我們的演算法	一集合，大小 O(log n+r)，兩 bloom filter	O((log n+ r)*k)個 hash 函數	介於[1]和[6]之間

表格 3: EOE 加密演算法比較

1.3 組織架構

本篇論文的組織架構如下，第二章介紹加密演算法會用到的技術，第三章介紹加密演算法和 range query 轉換，第四章分析加密演算法的安全性並和[1]做比較，而也會提出我們演算法的弱點，並提出依解決之道，第五章則做實驗分析解決的效果，第六章是一些問題討論，而第七章是結論。

注 1

IND-OCPA:

IND-OCPA 是修改自 IND-CPA，而 IND-CPA 介紹如下:

全名 Indistinguishability under chosen-plaintext attack 主要透過一安全遊戲來呈現，此遊戲有挑戰者和攻擊者，照以下步驟進行遊戲:

1. 攻擊者丟任意數量的明文對(M0,M1)給挑戰者

2. 挑戰者隨機選擇 1 數 b 為 0 或 1，並加密M_1^b, M_2^b ….並回傳

3. 攻擊者根據回傳的資訊，去猜 b 究竟是 0 還是 1，並傳回猜的結果b'

4. 挑戰者檢查 b 是否等於 b'

而我們定義攻擊者在這個遊戲裡，所能拿到的優勢為絕對值(攻擊者猜中 b 的機率-0.5)，而這個優勢若趨近到 0 的話就表示攻擊者沒辦法從密文中拿到有關密文的任何資訊。

接者回來講 IND-O(Ordered)CPA，因為 EOE 會洩漏大小關係，所以若照上面的遊戲，則攻擊者只需傳兩對明文組合(a,b)和(c,d)，a>c 且 d>b，因其 a 和 c 的大小關係與 b 和 d 的到小關係不同，就可以從回傳的兩個密文彼此之間的大小關係來判斷 b 為 0 或 1，所以在 IND-OCPA 裡面，我們限制攻擊者所傳的明文組合，條件為對任 2 個攻擊者所傳的明文組合(M_i^0, M_i^1)和 (M_j^0, M_j^1)，$M_j^0 > M_i^0 <=> M_j^1 > M_i^1$。而其餘同 IND-CPA，則若攻擊者在 IND-OCPA 這個遊戲中，所拿到的優勢趨近於 0，我們就可說此加密方法除了洩漏明文大小關係外，並沒有洩漏其他有關明文的資訊。

二、 技術介紹

在此一章節中，我們介紹加密演算法會用到的相關技術。

2.1 HYL encoding

是採自[7]這篇裡的 0-1 encoding，此方法會將數字轉化成兩集合，0 集合和 1 集合，而若有兩數 a 和 b，則可透過檢查 a 轉化的 1 集合和 b 轉化的 0 集合是否有交集來推得 a 是否大於 b，以下為詳細步驟。

1.對於輸入數字 k(舉例為 188)，把其化為 2 進位形式，也就是 10111100。

2.從第一個 bit 開始往下數，第一次截取長度 1，以例子來說是 1，第二次截取長度 2，也就是 10，以此類推直到數到最後一位元為止，以範例來說截取了 8 個字串，分別是 1, 10, 101, 1011, 10111, 101111, 1011110, 10111100。

3.接者檢測這些字串結尾的位元，若為 0，則把結尾改成 1 後加入 0 集合，若為 1，則直接把字串加入 1 集合，以範例來說:

0 集合: 11, 1011111, 10111101

1 集合: 1, 101, 1011, 10111, 101111

這樣就完成編碼，而比較大小方法如下例，我們假設有 2 數字，207 和 188，而經過 HYL encoding 後的集合如下:

207（二進位形式 11001111):

0 集合:111, 1101

1 集合:1, 11, 11001, 110011, 1100111, 11001111

188（二進位形式 10111100):

0 集合: 11, 1011111, 10111101

1 集合: 1, 101, 1011, 10111, 101111

可以看到 207 的 1 集合和 188 的 0 集合有 11 這個交集，故可推得 207 大於 188，而以下是證明。

證明:

假設兩數 a 與 b，而如果 a 的 1 集合和 b 的 0 集合有一交集 c，c 為 0 和 1 所構成的字串，假定 c 長度為 n，其前 n-1 個 bit 的字串為 d，即 c=d1(因在轉化時，字串結尾為 0 的會改成 1 後加入 0 集合，結尾為 1 的直接加入 1 集合，故 0 集合和 1 集合裡所有的字串都以 1 結尾)，則代表 a 的二進位形式其前 n 的 bit 的字串為 d1，而 b 的二進位形式其前 n 個 bit 的字串為 d0，可推得 a 是大於 b 的。

2.2 HMAC

全名為 hash-based message authentication code，此 code 的主要功能是附在訊息裡，讓擁有鑰匙的使用者去算出訊息的 hash 值，並透過檢查 HMAC 來確認訊息有無遭受竄改，形式為 HMAC(key,訊息)=hash 值，而 HMAC 具有以下特性:

1.若使用者擁有 key 和訊息，則可在多項式時間內解出此訊息的 hash 值。

2.給定一 hash 值，使用者無法在多項式時間內解出此 hash 值所對應的訊息。

而在此我們在加密方法使用 SHA-512 這個 HMAC。

2.3 Bloom filter

通常，我們要檢查一個集合是否包含某個元素，必須要檢查此集合中所有的元素，但當集合一大時就比較費時間，於是 Burton Howard Bloom 提出了 bloom filter 來節省時間和空間，伴隨者一點點錯誤機率，以下為詳細介紹。

Bloom filter 是一個長度為 m 的陣列，陣列裡每個元素皆為 0 或 1，另有 k 個 hash 函數，其功用為將字串對應到一整數，範圍 1~m。

當我們要對某一集合 A 做 bloom filter 時，先初始一陣列 b，長度 m，並將陣列 b 裡的值都設為 0，接者對 A 裡每個元素，都對其做 k 個 hash 函數，而每個 hash 函數都會把集合內的元素對應到陣列裡的某個位置，然後把該位置的值設成 1(不論該值為 0 或 1)，如下圖 2，每個箭頭代表一個 hash 函數:

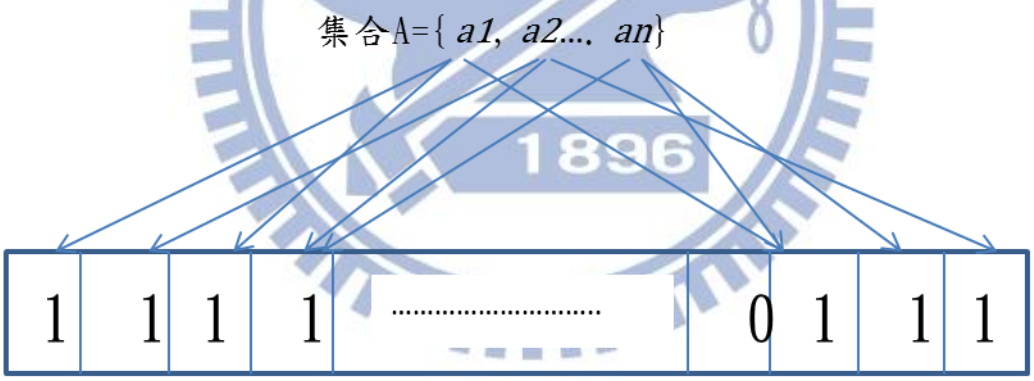


圖 2: Bloom filter 製作示意圖

之後若有檢測元素 d 是否在 A 集合中，便拿 b 來做 k 個 hash 函數，依照輸出的值找其在陣列 b 中對應的位置是否都為 1，如圖 3

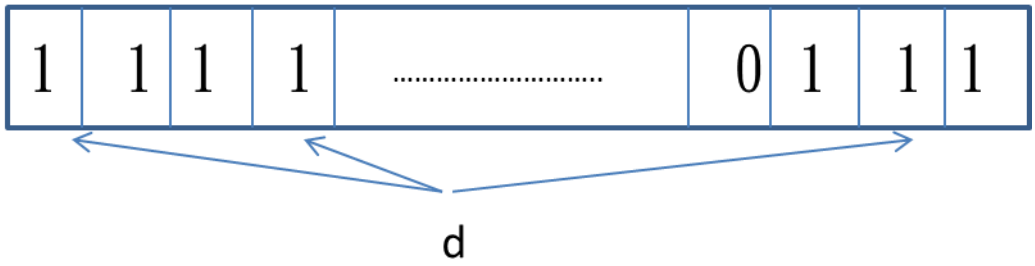


圖 3:用 bloom filter 測試元素在不在集合內之示意圖

若都為 1，則表示 d 在 A 集合內，反之則否。

這方法可以把一集合縮成一長度 m 的陣列，並把比較所需時間從 O(n)變成 O(k)，但是他有錯誤機率，有可能檢測元素並不在集合內，但檢測結果卻說有在，這個機率跟陣列長度 m、集合元素個數 n 及 hash 函數的個數 k 有關，而當滿足$k = m \ln 2 / n$ 時，此機率達到其最低值$\frac{1}{2^k}$。

2.4 Reed Solomon Code

是一種錯誤更正編碼，其透過增加傳送的資料量，使得即使中間傳輸有出錯，只要在一定比例以下，還是可以還原成本來的資料，而他是一個非二元的 code，以 RS(k, b) code 為例，輸入為 b 個 m-bit symbol，輸出為 k 個 m-bit symbol，k 大於 b 且小於$2^m + 2$，而輸出的 k 個 m-bit symbol 中，前 b 個為原本的 symbol，後 k-b 個為新增的 symbol，而其擁有可以修復$\frac{k-b}{2}$個 symbol 的能力，並且具有最小漢明距離為 k-b+1，也就是說不論你輸入的 b 個 symbol 為何，只要不是相同的 2 組 b 個 symbol，2 組輸出的 k 個 symbol 必定至少在 k-b+1 個位置上不相同。

而其詳細 encoding 如下:

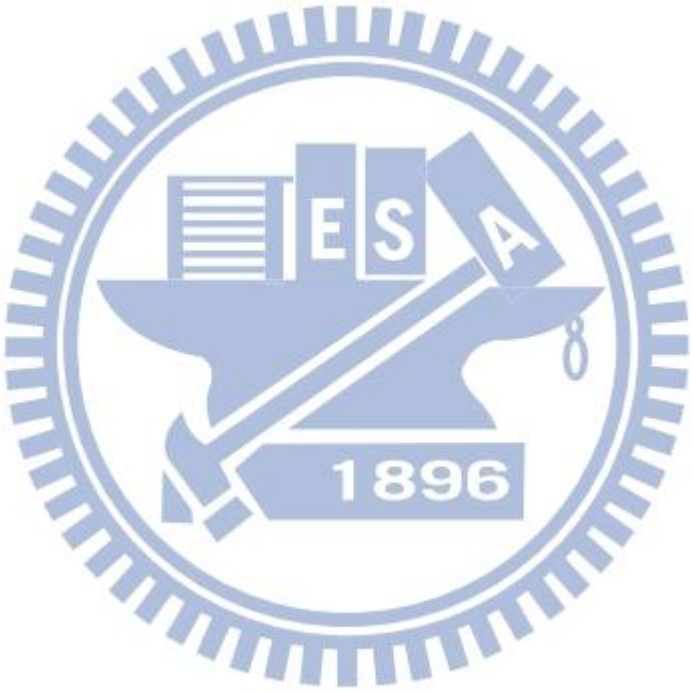
1. 針對 m-bit symbol，首先找出 GF(2^m)的 Primitive polynomial 的根α，
α 也是 GF(1024 的生成元)，並作出 $g(x)=(x-\alpha^1)*(x-\alpha^2)\dots(x-\alpha^{k-b})$

2.接者我們將輸入的 b 個 symbol 看成是方程式係數，並產生方程式

n(x)=$b_1 + b_2x + b_3x^2 + \cdots b_bx^{b-1}$

3.將 n(x)乘上x^{k-b}後除上 g(x)，並得到餘數 r(x)。

4.最後輸出$x^{k-b} * n(x) + r(x)$的方程式係數，共 k 個，每個係數為 m-bit symbol。



三、 演算法介紹

3.1 加密演算法

我們的演算法包含 KeyGen , Encryption , Decryption。

公開常數:k 個 hash 函數，bloom filter 陣列長度 m。

KeyGen(1^{γ}): 給定一安全係數γ，隨機產生γ長度的字串 2 個，sk1 和 sk2 並輸出。

Encryption(資料 D , sk1 , sk2):

首先，把 D 轉化為二進位形式，接者對其做 HYL encoding 可得到 D 的 0 集合和 1 集合，而由於 0 集合和 1 集合其元素個數會因為各數字的二進位形式不同而不同，舉例來說:

156(二進位 10011100)

0 集合:11,101,1001111,10011101

1 集合:1,1001,10011,100111

因其二進位形式由 4 個 0 和 4 個 1 構成，所以 0 集合元素個數 4 個，1 集合元素個數 4 個，為了防止攻擊者可從元素個數得到資訊，我們稍微修改 HYL encoding，在不影響比較結果的情況下，把 0 集合和 1 集合的元素個數都補到上限，也就是 bit 數(以上 156 的例子來講的話，0 集合和 1 集合都補到 8 個元素)。

新增元素到 0 集合和 1 集合中，如下

0 集合: 11,101,1001111,10011101 ,x_1, x_2, x_3, x_4

1 集合: 1,1001,10011,100111, y_1, y_2, y_3, y_4

而為了避免影響比較結果,x_1, x_2, x_3, x_4不能與本來的 1 集合元素有交集,y_1, y_2, y_3, y_4也不能和本來的 0 集合元素有交集而且x_1, x_2, x_3, x_4也不能和y_1, y_2, y_3, y_4有交集。

補足方法:首先觀察本來的 0 集合和 1 集合的元素皆為 1 收尾，而且其長度最多為 bit 數(8)，所以我們把本來 0 集合的元素尾數改 0 後加到 1 集合內，即為x_1, x_2, x_3, x_4，而對本來 1 集合內的元素,後頭補 0 補到長度為 bit 數加 1 後加入 0 集合內,即為y_1, y_2, y_3, y_4，如下(同樣以 156 為例)

0 集合: 11,101,1001111,10011101 ||100000000, 100100000,100110000,100111000

1 集合: 1,1001,10011,100111 ||10,100,1001110,10011100

此方法不只補足個數，也讓任 2 密文，其 0 集合與 0 集合的交集個數等同於其 1 集合和 1 集合的交集個數，而此數也等同於兩者原本 0 集合與 0 集合的交集個數加上 1 集合和 1 集合的交集個數，降低可能洩漏的資訊(本來可得到 0 集合與 0 集合的交集個數和 1 集合與 1 集合的交集個數，但現在只能得到兩者總和)，如下例(156 和 131)

131(二進位形式 10000011)

0 集合:11,101,1001,10001,100001 ||100000000,100000100,100000110

1 集合:1,1000001,10000011 ||10,100,1000,10000,100000

可以看到若比較原本的 coding(只看||左邊),156 的 0 集合和 131 的 0 集合交集個數為 2，1 集合和 1 集合交集個數為 1，而經過補足後，兩者皆為 3。

在經過 HYL coding 和補足完後，我們可得到 2 元素個數相等的集合，0 集合和 1 集合，以D_0, D_1稱之，$D_0 = \{d_{01}, d_{02} \dots d_{0b}\}$、$D_1 = \{d_{11}, d_{12} \dots d_{1b}\}$，b 為 bit 數。

之後使用 HMAC 將每個集合裡的元素加密，做出$HMAC_{sk1}(D_0)$、$HMAC_{sk1}(D_1)$，

$HMAC_{sk1}(D_0)$={ $HMAC_{sk1}(d_{01}), HMAC_{sk1}(d_{02})\cdots. HMAC_{sk1}(d_{0b})$}

$HMAC_{sk1}(D_1)$={ $HMAC_{sk1}(d_{11}), HMAC_{sk1}(d_{12})\cdots. HMAC_{sk1}(d_{1b})$}

接者為了加速伺服器處理速度，將$HMAC_{sk1}(D_0)$和$HMAC_{sk1}(D_1)$做成 bloom filter，先產生一長度 m 的陣列BD_0和BD_1，並把$BD0_1$ ….$BD0_m$和$BD1_1$ ….$BD1_m$初始值皆設為 0，然後對$HMAC_{sk1}(D_0)$裡每個元素 e 都做 k 個 hash 函數，並把陣列中，輸出數字位置($b_{h_{1(e)}}$, $b_{h_{2(e)}}$,… $b_{h_{k(e)}}$)的值設成 1，如圖 4 所示(每個箭頭代表一個 hash 函數):

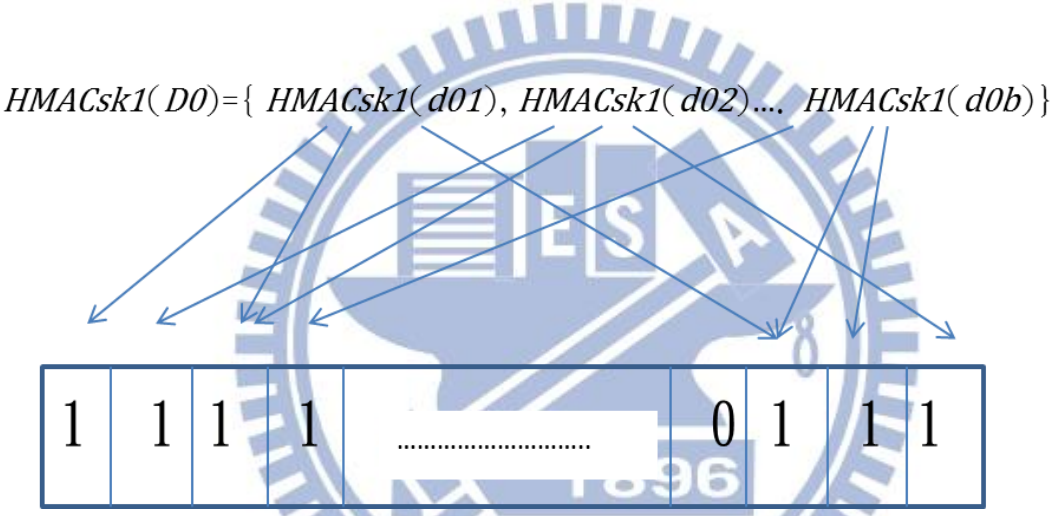


圖 4:HMAC 集合的 bloom filter 製作示意圖

做出$HMAC_{sk1}(D_0)$和$HMAC_{sk1}(D_1)$的 bloom filter BD_0和BD_1後，在用第二把密鑰 sk2 以一對稱式加密方法 SENC 加密成密文C_D=SENC(sk2,D)，最後輸出 bloom filter BD_0、BD_1、$HMAC_{sk1}(D_1)$及C_D。

最後在為加密方法做個總結:

1. 把資料 D 化成二進位形式，並對其做 HYL encoding 和補足，得到 D 的 0 集合和 1 集合D_0和 D_1。

2. 將D_0和D_1裡的元素利用 sk1 做 HMAC，得到$HMAC_{sk1}(D_0)$、$HMAC_{sk1}(D_1)$。

3. 把$HMAC_{sk1}(D_0)$、$HMAC_{sk1}(D_1)$這兩個集合，利用 k 個 hash 函數，做成兩長度為 m 的 bloom filter 陣列 BD_0和BD_1。

4. 對資料 D 用 sk2 進行對稱式加密，得到C_D= SENC(sk2,D)。

5. 輸出$HMAC_{sk1}(D_1)$、BD_0、BD_1和C_D。

Decryption(sk2, C_D):

就用密鑰 sk2 去解C_D，輸入解密結果資料 D=SDEC(sk2, C_D)。

3.2 比較大小方法

接者介紹比較演算法 W，雲端可利用 W 直接比較兩密文的大小，步驟如下:

W(密文 A,密文 B):

1. 對所有在$HMAC_{sk1}(A_1)$裡的元素，每一個都做 K 個 hash 函數，去檢查其對應位置在 B 的 bloom filterBD_0上是否都為 1，如圖 5:

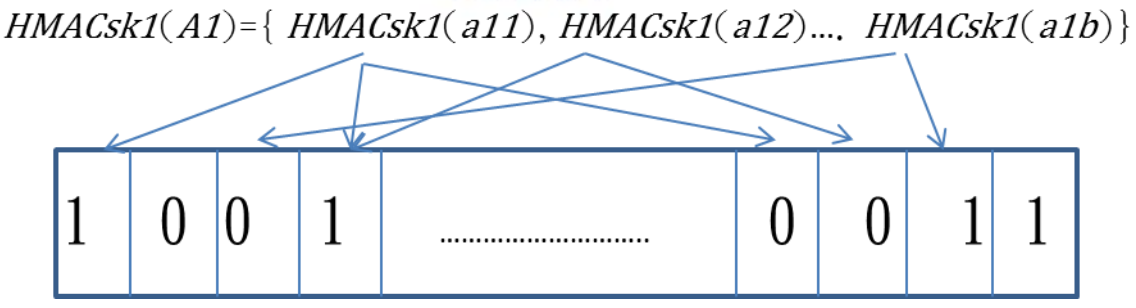


圖 5:使用 bloom filter 檢測交集示意圖

若找到一元素，其做 k 個 hash 函數後在 B 的 bloom filter 上的對應位置皆為 1 便中斷，輸出 A 大於 B，若搜尋完畢沒有找到的話，則進行下一步。

2. 這次換對所有在$HMAC_{sk1}(B_1)$裡的元素每一個都做 k 個 hash 函數，去檢查其對應位置在 A 的 bloom filterBA_0上是否都為 1，同上圖的動作，而若有找到一元素其做 k 個 hash 函數後在 A 的 bloom filter 上的對應位置皆為 1 便中斷，輸出 A 小於 B，若搜尋全部後仍未找到，則輸出 A 等於 B。

3.4 Range query 轉換方法

假設要求尋找區間為[A,B]的資料，步驟如下:

1. 把 A 和 B 轉成二進位形式，並對其進行 HYL encoding 和補足。

2. 只針對 A 的 0 集合和 B 的 1 集合，進行 HMAC 加密，得出$HMAC_{sk1}(A_0)$和$HMAC_{sk1}(B_1)$。

3. 傳送$HMAC_{sk1}(A_0)$和$HMAC_{sk1}(B_1)$給雲端。

4. 雲端利用首先利用$HMAC_{sk1}(A_0)$和密文 D 中的BD_1這個 bloom filter，用 K 的 hash 函數去檢查兩者有無交集，並對有交集的密文 D 進行下一步。

5. 對剛剛暫存的密文 D，用 K 個 hash 函數去檢查其BD_0這個 bloom filter 和 $HMAC_{sk1}(B_1)$有無交集，若有，則把此密文 D 加入結果中

6. 最後回傳每個密文 D 中的C_D供使用者解密。

四、安全性分析

4.1 安全性比較

我們的目標在於防止伺服器透過加密資料來取得有關明文的資訊，而有關 EOE 的高安全性 IND-OCPA 我們沒有達到，也就是說我們會洩漏除了明文大小關係以外的某些資訊，然而，究竟這些洩漏的資訊有沒有辦法量化，攻擊者有沒有辦法從這些資訊推出確切的明文，我們的演算法究竟比[1]安全性高還是逼，於是，我們採用[5]和[12]這兩篇，這兩篇分別提出自己的安全性標準，並且有算出[1]在此標準下的表現，此兩標準皆為安全遊戲，將攻擊者分成兩個等級，[5]這篇是第一等級，攻擊者只知道密文，[12]這篇是第二等級，攻擊者知道一定數量的明文和密文組合，共同的目標都是解出確切的明文，分析在這兩個等級裡，攻擊者所能得到的資訊，並與[1]做比較。

4.1.1 第一等級安全性比較

首先介紹針對第一等級的攻擊者所做的安全遊戲，這安全遊戲為攻擊者在拿到一定數量的密文情況下，可否解出這些密文中，包含某個特定明文，此事發生的機率，以其明文範圍 M 和攻擊者拿到的相異加密資料總數 Z 為變數，詳細定義如下:

(Z,M)安全遊戲:

1.挑戰者隨機從明文範圍為 0~M-1 中，挑選 Z 個不一樣的數字$r_1..r_z$，並把其加密後成$c_1..c_z$並傳給攻擊者。

2.從這些資訊中，挑選 1 個數字 x，認定這些密文中包含此數的密文，並把此數 x 回傳。

3.檢查傳過來的數字 x 是否在一開始隨機選擇的數字$r_1..r_z$中，若有，則算攻擊成功，反之攻擊失敗。

我們定義攻擊者在(Z,M)安全遊戲中的優勢為攻擊成功的機率，而若這個優勢趨近於 0，則表示即使攻擊者拿到了 Z 個數量的密文，也沒辦法從中找出某個特定明文。

首先分析我們的密文，由兩個 bloom filter 和一個 HMAC 集合及一個 AES 加密的密文所組成，而由於 AES 加密的密文不回洩漏資訊，兩個 bloom filter 實為兩個 HMAC 集合所做成，因此我們只考慮此兩集合，而此兩集合裡的元素皆用 HMAC 加密，且集合的元素個數也都調成與明文 bit 數相同，使得攻擊者單從單一密文的此兩 HMAC 集合是找不出有關明文的任何資訊，他唯一能做的就是去比較兩密文的 HMAC 集合之間的交集，而此交集數量就等於原先沒經 HMAC 的集合交集數量,0 集合和 0 集合、1 集合和 1 集合、0 集合和 1 集合的交集數量可以得知，而 0 集合和 1 集合的交集數量非 1 即 0(代表者明文大小關係)，所以我們專注在 0 集合和 0 集合、1 集合和 1 集合的交集數量，而此資訊其實會得到有關明文的資訊，如下例:

假設現有 3 個數字，皆為 8bit， 180、188 和 208，其經過 HYL encoding 後如下

180(二進位形式 10110100):

0 集合:11, 10111, 1011011, 10110101, 100000000, 101000000, 101100000, 101101000

1 集合:1, 101, 1011, 101101, 10, 10110, 1011010, 10110100

188(二進位形式 10111100):

0 集合:11, 1011111, 10111101, 100000000, 101000000, 101100000, 101110000, 101111000

1 集合:1, 101, 1011, 10111, 101111, 10, 1011110, 10111100

208(二進位形式 11010000):

0 集合:111, 11011, 110101, 1101001, 11010001, 100000000, 110000000, 110100000

1 集合:1, 11, 1101, 110, 11010, 110100, 1101000, 11010000

就二進位形式來看，180 和 188 前 4bit 一樣，所以無論是 180 的 0 集合和 188 的 0 集合，或 180 的 1 集合和 188 的 1 集合，其交集數皆為 4，而 180 和 208 前 1bit 一樣，所以交集數是 1，188 和 208 交集數也是 1。

而攻擊者要如何得到資訊呢，我們先把 180,188,208 稱為 a, b, c，攻擊者先看 a 和 b 來看，便可透過其 1 集合交集數得知此 2 數前四個 bit 一樣，在透過 1 集合和 0 集合的比較得知 b 為較大的一方，在此情況下，攻擊者可推得 a 的第 5 個 bit 為 0，b 的第 5bit 為 1，接者看 a 和 c，攻擊者透過交集數知道此 2 數前 1bit 一樣，再透過比大小可得知 c 較大，如此又可推得 c 的第 2bit 為 1，a 的第 2bit 為 0，而由於 a 和 b 前 4bit 一樣，所以 b 的第 2bit 也為 0。

從以上可知，攻擊者的策略便是從$c_1..c_z$中，找到一個c_i，而此c_i和其他密文交集數量種類為最多，則猜c_i的明文，成功機率最高，而我們將會去算當 M=16384(2^{14})的情況下，證明我們的演算法會比較好。

我們首先定義對數字 x 來說，與其交集數為 i 的數字為x_i，接者因為[5]這篇的優勢上限為$\frac{9Z}{\sqrt{M-Z+1}}$，當 M=16384 和 Z=1 時，此值會大於$\frac{1}{2^4}$，因此我們只對 10、11、12 和 13 交集數進行討論。

我們可以把從 M 中取 Z 個數字的結果拆成以下幾個部份:

1. 存在一數 x 在 Z 個數字中，而x_{10}、x_{11}、x_{12}和x_{13}也同樣在這 Z 個數字中。

2. 1 不發生且存在一數 x 在 Z 個數字中，而x_{10}、x_{11}、x_{12}和x_{13}中，有三個也在這 Z 個數字中。

3. 1, 2 不發生且存在一數 x 在 Z 個數字中，而x_{10}、x_{11}、x_{12}和x_{13}中，有二個也在這 Z 個數字中。

4. 1, 2, 3 不發生且存在一數 x 在 Z 個數字中，而x_{10}、x_{11}、x_{12}和x_{13}中，有一個也在這 Z 個數字中。

5. 1~4 都不發生，也就是對這 Z 個數字中的每一個數字 x，x_{10}、x_{11}、x_{12}和x_{13}都不在這 Z 個數字中。

而由於我們要取上限，若 1 發生的話，我們把攻擊者解出密文的機率定為 100%，而若 2 發生的話定為$\frac{1}{2}$，以次類推，於是我們就可以把優勢上限寫成 Pr(1 發生)*1 +Pr(2 發生)*1/2+Pr(3 發生)*1/4+Pr(4 發生)*1/8+Pr(5 發生)*1/16

而以下則是我們計算後的結果，由於優勢上限超過 1 就沒有用了，於是我們會把超過 1 的部份切掉，如下圖 6:

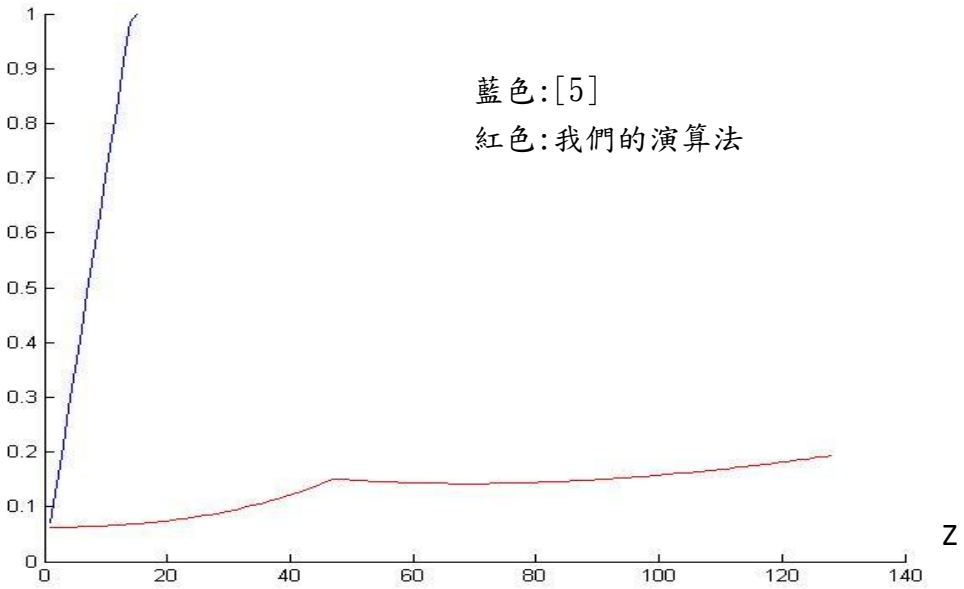


圖 6:優勢上限比較圖 1(第一等級安全遊戲)

可以看到[5]推出來的優勢上限在 Z=15 時就超過 1 了，而我們的演算法在 Z=128 時上限仍只有 0.2 左右，顯見我們的演算法在此標準下比起[1]這篇佔優。

4.1.2 第二等級安全性比較

第二個安全遊戲的參數定義和第一個一樣，詳細定義如下:

(Z,M)安全遊戲:

1.挑戰者隨機從明文範圍為 0~M-1 中，挑選 Z 個不一樣的數字$r_1..r_z$，並把其加密後成$c_1..c_z$，在從剩下的數字中隨機選擇一數 x，並加密成 c_x最後把
(r_1, c_1)、(r_2, c_2) ….(r_z, c_z), c_x傳給攻擊者

2.從這些資訊中，猜想c_x的明文為x'，並把回傳x'。

3.檢查傳過來的數字x'是否等於 x，若是，則算攻擊成功，反之攻擊失敗。

我們定義攻擊者在這個安全遊戲下所得到的優勢為攻擊成功的機率，也就是 x 等於x'的機率。

首先我們假設$r_1 < r_2 < \cdots r_z$，則 0, r_1….r_z, M-1 可以把明文切成 Z+1 個區間，我們令這些區間為$m_1 \ldots m_{Z+1}$，而因為我們的加密方法為可以對密文比較大小，所以攻擊者藉由比較c_x和$c_1..c_z$，可以把 x 定在某一區間m_i內，則我們可以把機率改寫如下:
$\Pr(x = x') = \sum_{i=1}^{z+1} \Pr\left(x = x' \middle| x在m_i區間內\right) * \Pr(x在m_i區間內)$
而因為 x 為隨機挑選,所以 x 落在哪一區間內基本上所有加密方法都會得出一樣的結果，所以我們專注在討論當 x 在m_i區間內時，x 被猜中的機率。

當 x 在m_i區間內時，攻擊者知道r_{i-1}和r_i的明文和密文，所以本來只比較c_x和c_{i-1}交集數(假設為 n)的話,只能知道此兩數前 n bit 一樣,和第 n+1 bit 為何,但現在因為r_{i-1}已知，使得在此種狀況下，攻擊者可得到 x 的前 n 個 bit，如此一來，攻擊者的策略就是取得c_x和c_{i-1}及c_x和c_i的交集數量，並選擇較大的一方做為猜測依據。

在弄懂攻擊者的策略後，接者我們討論當 x 在m_i區間內時，x 被猜中的機率最高為

何，我們將以m_i區間的大小為準做討論，從最高的觀點出發，那自然是與c_{i-1}和c_i交集數量多的最好都在m_i區間裡面，如此一來 x 被猜中機率就很高，但須考慮到一因素，若x被猜中機率為$\frac{1}{2^n}$(後 n 個 bit 不知道)，則和 x 擁有相同機率且也m_i區間內的數字共有2^n個，因為此2^n個數的開頭都一樣，差別只有後 n 個 bit，因此這2^n個數字為連在一起，r_{i-1}和r_i不可能在此2^n個數字中，而此2^n個數字平均被猜中機率為$\frac{2^n}{\text{區間大小}} * \frac{1}{2^n} = \frac{1}{\text{區間大小}}$。

所以在討論上限時，不能一味地把與c_{i-1}和c_i交集數量多的數字放進m_i區間內，而必須同時考慮此種數字所佔大小，這些大小總和必須剛好填滿整個區間，而這些數字大小分別為 1, 2, 4, ...2^n，而每個大小都有 2 塊(c_{i-1}和c_i)，舉例來說，區間大小 6 和區間大小 5，6 可以寫成 1+1+2+2，被猜中機率為$\frac{2}{3}$，但 5 只能寫成 1+2+2，被猜中機率為$\frac{3}{5}$，也就是區間大小從 5 增加到 6，被猜中機率反而上升，而以下我們計算出此種最高猜中機率，並和[1]比較，下圖 7 為結果，橫軸為區間大小，縱軸為最高被猜中機率:

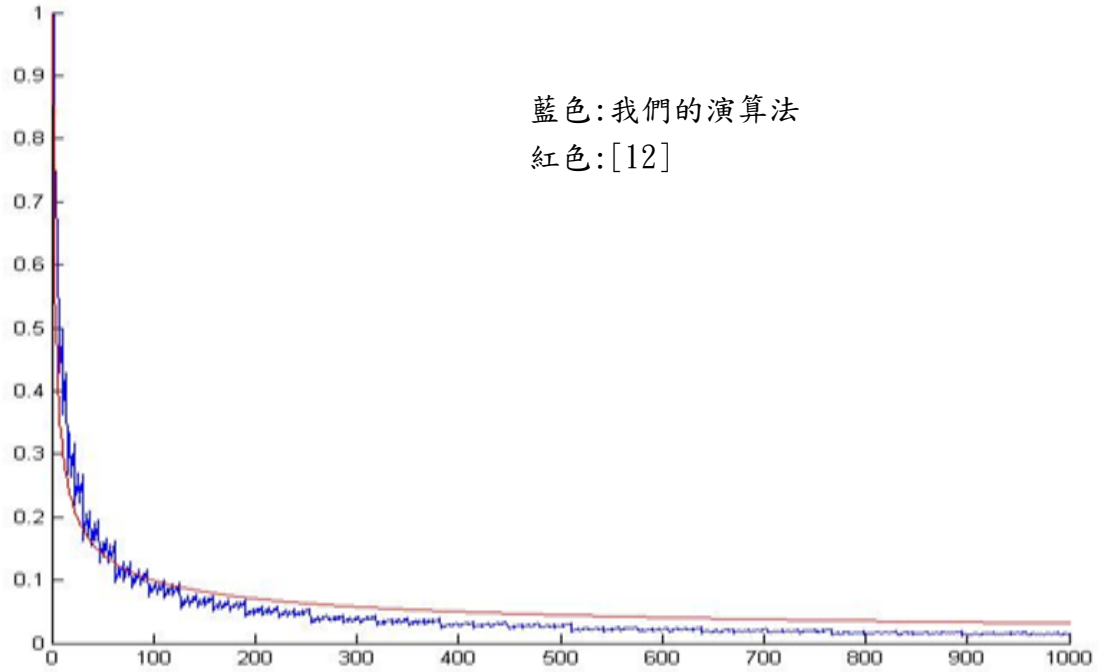


圖 7:平均被猜中機率上限比較圖

可以看到我們的演算法即使跳動很大，但整體趨勢還是往下降的，而在一開始，也

就是區間大小較小時，我們的演算法比[12]來的差，但在區間大小超過 126 後，我們的演算法就會比[12]好，而當區間大小為 1000 時，我們的演算法的機率大約為[12]的一半。

可見在區間大小較小時，我們的演算法會比較差，而區間大小跟 Z 值有關，當攻擊者知道越多的明文和密文組合，也就是 Z 值越大，區間平均大小就越小，所以現在我們想求得一個 Z 值上限 U，當 Z 值小於 U 時，我們的演算法不會比較差。

首先假設有兩函數 MY(n)和 OPE(n)，輸入為一整數 n，輸出為區間大小為 n 時，我們的演算法(MY(n))和[1](OPE(n))的最高被猜中機率，接者我們令平均區間大小為 k，k=$\frac{M-Z}{Z+1}$，則只要 k 具有下列兩性質，我們的演算法就不會比較差:

1.對於所有小於 k 的數字 i，MY(i)< $\mathrm{OPE(k)}-\frac{1}{M-Z}$

2.對於所有大於或等於 k 的數字 j，j-k>k*(M-Z)*(MY(j)-OPE(k))

證明:

首先，對於[1]來講，其算機率上限時是每塊區間都切成一樣，也就是每塊大小都為 k 時，其機率會最大，因此我們假設當 k 滿足上述 2 條件時，我們存在一切區塊大小的方法$x_1, x_2 \ldots x_{z+1}$使得我們的演算法的機率會大於[1]平均切的機率，也就是 $\sum_{j=1}^{Z+1} MY(x_j)$ >(Z+1)*OPE(k)，而再假設此切法中前 i 個區間大小小於 k，剩下的大於或等於 k。

接者因為所有小於 k 的數字都滿足性質 1，則可得 $\sum_{j=1}^{i} MY(x_j)$ < i * $\left(\mathrm{OPE(k)}-\frac{1}{M-Z}\right)$(1)，另對所有大於或等於 k 的數字滿足性質 2，則得 $\sum_{j=i+1}^{Z+1} x_j$ -k*(Z+1-i) >k*(M-Z)*($\sum_{j=i+1}^{Z+1} MY(x_j)$-(Z+1-i)*OPE(k))，移項可得 $\sum_{j=i+1}^{Z+1} MY(x_j)$ < ($\sum_{j=i+1}^{Z+1} x_j$ -k*(Z+1-i))/ k*(M-Z) +(Z+1-i)*OPE(k)(2)，(1)+(2)可得 $\sum_{j=1}^{Z+1} MY(x_j)$ < (Z+1)*OPE(k)+ ($\sum_{j=i+1}^{Z+1} x_j$ -k*(Z+1))/ k*(M-Z)，而由於區間大小總和為 k*(Z+1)，可推得$\sum_{j=i+1}^{Z+1} x_j$ -k*(Z+1)≤ 0，$\sum_{j=1}^{Z+1} MY(x_j)$ ≤(Z+1)*OPE(k)，與假設矛盾

所以只要 k 具有上述性質，就不存在切法使得我們的演算法會得到較高的機率。

而根據實驗後推算，當 k 大於 214 時，就具有上述性質，而我們的 Z 值上限就為$\frac{M-214}{215}$。

但即使如此，交集數量還是一個很大的問題，雖然沒辦法解出確切的明文，卻能把明文限縮在一定範圍內，而要怎樣才能打破此種關係呢，我們採用增加元素的方式，比如說有一對 0 交集的密文和一對 7 交集的密文，我們給 0 交集的密文加上 7 個相同元素使它變成 7 交集，而對 7 交集的密文加上 7 個不同元素，則它還是 7 交集，則 2 對密文單從交集數就無法分辨哪對是哪對，而在 HYL encoding 下，想照這個方法加元素加到所有密文交集數都一樣的話，需要的大小太大，所以有沒有加元素的方法，而此方法會讓原交集數量少的可以在相同新增元素個數時可以拿到較多的交集數，而原先交集數量多的則反之。

所以我們想到使用 RS encoding，因為 RS encoding 具有性質最小漢明距離，此性質是說，只要輸入集合不同，則可確保輸出集合必定在至少幾個位置不同，而因為輸出集合是包含輸入集合的，所以輸入集合越相似(交集數多)，經由 RS encoding 新增的元素就會受到此項限制，降低碰撞(產生交集)的機率，而使得說原本兩資料其較不相似的(交集數量較少)的，有較高的機率在 code 後得到較高交集數量，符合我們的要求，但到底這高一點的機率會造成怎樣的效果，將有賴第 5 章實驗去證明。

4.2 進階加密演算法

接者介紹加入 RS encoding 的加密演算法，由於在 HYL encoding 中，1 和 01 是表不同元素，但若要直接做 RS encoding 的話會被看成相同，所以首先我們要先產生一函數，將 HYL encoding 和補足產生的元素對應到一對一到整數，方便我們做 RS encoding，至於此函數需要一些前置工作如下:

首先檢查前面 HYL encoding 產生的元素，在資料 bit 數為 b 的情況下，產生元素長度最長為 b，而因為結尾一定是 1，所以長度為 1 的元素只有 1 這一種，長度為 2 的

元素有 01 和 11 兩種…長度為 b 的元素就有2^{b-1}個，全部加起來後約等於2^b，接者考慮補足所需的元素，0 集合補足是把 1 集合的元素結尾改 0 做出，所以所需總數等同 HYL encoding 的元素總數2^b，最後是 1 集合補足，是把 0 集合結尾加 0 加到 bit 數加 1，總數也是等同 HYL encoding 的元素總數2^b，所以此函數的值域在資料 bit 數為 b 的情況下，需要大小為 $3\cdot 2^b$。

之後我們建立高度為 b+1 的滿二元樹，並把每個節點位置依順序標號，如下圖 8:

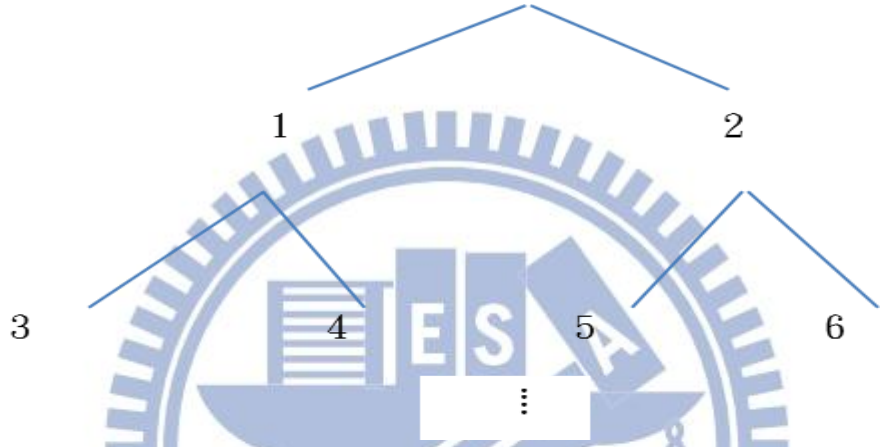


圖 8:標號過的二元樹

接者我們把 0~2^{b+2}-1 的數字隨機排列，並按照位置填入二元樹，如下圖 9 以 b 等於 8 為例，隨機排列的前 6 數為 407, 415, 437, 475, 69, 344:

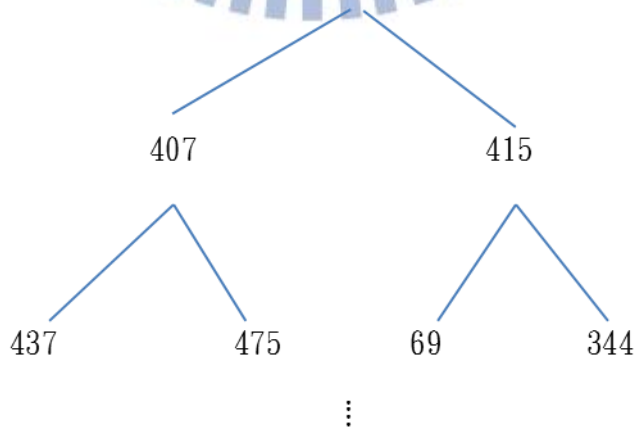


圖 9:填值過後的二元樹

建好二元樹後，函數前置工作就算完成。

函數的實際演算法如下:

從剛剛建立的二元樹的根部開始,把輸入的 HYL encoding 或補足的元素當成路徑，0 為往左走，1 為往右走去追蹤這棵樹，並回傳最後停下的節點值，舉例而言，若輸入為 10，輸出為 69，輸入為 0，輸出為 407

有了函數後，便把加密演算法改寫如下:

KeyGen(1^{γ}): 給定一安全係數γ，隨機產生γ長度的字串 4 個，sk1、sk2、sk3 和 sk4 並輸出，還有產生一函數 r，將 HYL encoding 和補足所產生的元素對應到一整數。

Encryption(資料 D , sk1 , sk2, sk3, sk4, 函數 r):

1. 先把資料 D 轉成二進位形式，並對其做 HYL encoding 和補足，產生其 0 集合D_0和 1 集合D_1

2. 使用函數 r，把D_0和D_1裡的元素都轉成數字，形成新的 0 集合D_0'和 1 集合D_1'

3. 使用 RS(n, b) code 對D_0'和D_1'進行 coding，得到兩大小為 n 的集合D_0''和D_1''如下

$$D_0'' = \{d_{01}, d_{02} \dots d_{0b}, d_{0b+1}, \dots . d_{0n}\}$$

$$D_1'' = \{d_{11}, d_{12} \dots d_{1b}, d_{1b+1}, \dots . d_{1n}\}$$

注意在D_0''和D_1''內，前 b 個元素為 HYL encoding 和補足產生，後面 n-b 個元素為 RS code 產生，為了不讓這些新加的元素影響比較大小結果，我們將用不同的 key 對他們做 HMAC，藉此讓 RS encoding 產生不會和 HYL encoding 和補足產生的元素有交集。

4. 對$d_{01}, d_{02} \dots d_{0b}$和$d_{11}, d_{12} \dots d_{1b}$這些由 HYL encoding 和補足產生的元素，我們用 sk1 對其做 HMAC,而對$d_{0b+1}, \dots . d_{0n}$這些由 RS code 產生的 0 集合元素,使用 sk3 為其 HMAC，

而對d_{1b+1},….d_{1n}這些由 RS code 產生的 1 集合元素，使用 sk4 對其進行 HMAC，如下:

$$HMAC(D_0'')=\{\ HMAC_{sk1}(d_{01}), HMAC_{sk1}(d_{02})\cdots.\ HMAC_{sk1}(d_{0b})$$

$$HMAC_{sk3}(d_{0b+1}), HMAC_{sk3}(d_{0b+2})\cdots.\ HMAC_{sk3}(d_{0n})\}$$

$$HMAC(D_1'')=\{\ HMAC_{sk1}(d_{11}), HMAC_{sk1}(d_{12})\cdots.\ HMAC_{sk1}(d_{1b})$$

$$HMAC_{sk4}(d_{1b+1}), HMAC_{sk4}(d_{1b+2})\cdots.\ HMAC_{sk4}(d_{1n})\}$$

5.把$HMAC_{sk1}(D_0'')$、$HMAC_{sk1}(D_1'')$這兩個集合，利用 k 個 hash 函數，做成兩長度為 m 的 bloom filter 陣列 BD_0和BD_1。

6.對資料 D 用 sk2 進行對稱式加密，得到C_D= SENC(sk2,D)。

7.輸出$HMAC_{sk1}(D_1'')$、C_D、陣列BD_0和BD_1。

Decryption 則不變。

Range Query Transformation 也需要做修改如下:

假設要求尋找區間為[A,B]的資料，步驟如下:

1.把 A 和 B 轉成二進位形式，並對其進行 HYL encoding 和補足。

2.使用函數 r 把 A 的 0 集合A_0和 B 的1 集合B_1裡的元素轉成數字

3.對A_0和B_1做 RS (n,b) coding，產生A_0'和B_1'如下

$$A_0' = \{a_{01}, a_{02}\ \ldots a_{0b}, a_{0b+1}, \ldots. a_{0n}\}$$

$$B_1' = \{b_{11}, b_{12}\ \ldots b_{1b}, b_{1b+1}, \ldots. b_{1n}\}$$

4. 對$a_{01}, a_{02} \ldots a_{0b}$和$b_{11}, b_{12} \ldots b_{1b}$用 sk1 進行 HMAC，對$a_{0b+1}, \ldots . a_{0n}$用 sk3 進行 HMAC，對$b_{1b+1}, \ldots . b_{1n}$用 sk4 進行 HMAC 加密，而後得出$HMAC_{sk1}(A_0')$和$HMAC_{sk1}(B_1')$如下:

$$HMAC(A_0')=\{ HMAC_{sk1}(a_{01}), HMAC_{sk1}(a_{02}) \cdots . \ HMAC_{sk1}(a_{0b})$$

$$HMAC_{sk3}(a_{0b+1}), HMAC_{sk3}(a_{0b+2}) \cdots . \ HMAC_{sk3}(a_{0n})\}$$

$$HMAC(B_1')=\{ HMAC_{sk1}(b_{11}), HMAC_{sk1}(b_{12}) \cdots . \ HMAC_{sk1}(b_{1b})$$

$$HMAC_{sk4}(b_{1b+1}), HMAC_{sk4}(b_{1b+2}) \cdots . \ HMAC_{sk4}(b_{1n})\}$$

5. 傳送$HMAC_{sk1}(A_0')$和$HMAC_{sk1}(B_1')$給雲端。

6. 雲端利用首先利用$HMAC_{sk1}(A_0')$和密文 D 中的BD_1這個 bloom filter，用 K 的 hash 函數去檢查兩者有無交集，並對有交集的密文 D 進行下一步。

7. 對剛剛暫存的密文 D，用 k 個 hash 函數去檢查其BD_0這個 bloom filter 和$HMAC_{sk1}(B_1')$有無交集，若有，則把此密文 D 加入結果中

8. 最後回傳每個密文 D 中的C_D供使用者解密。

五、 實驗

5.1 安全性標準

再做實驗之前，必須先定義安全性的標準，我們的目的在於防止攻擊者可從經過 RS encoding 後的密文交集數量去得知 RS encoding 前的密文交集數量，所以我們先假設攻擊者知道加密時所使用的 RS encoding，當然攻擊者也可做這實驗得到數據，並依這些數據去推說:當我看到此兩隨機挑選的密文的 0 集合交集數和 1 集合交集數量分別為 x 和 y 時，他們原先最有可能是 n 交集，因此若要統計攻擊者猜中交集數量的最高機率，就先對每一組經 RS code 後的 0 交集數量和 1 集合交集數量的組合(x, y)，其原先交集數量為 n 的有 z 組，找到一 n 使 z 最大，舉例說假設經 RS encoding 後的交集數量的組合為(2,3)的，其原先交集數量 0 的有 3 組、1 的有 5 組、2 的有 7 組，則我們就取 7 這個數，並對所有的(x, y)都找到這種 z 值後，加總除以全部樣本數就可得到攻擊者猜中交集數量的最高機率，注意此機率最低為 50%，因為隨機取 2 數，此 2 數經 HYL encoding 和補足後交集數量為 0 的機率就是 50%(第一個 bit 不同)，所以若混淆到最佳效果時，不管經 RS encoding 後交集數量為何，都只能得到猜原先 0 交集機率為最高的結果。

除了防止攻擊者猜中交集數量外，我們去檢測說，是否原先交集數量大的，經過 RS code 後交集數量仍然還是多的一方，因此我們也統計以下事情發生的機率:假設隨機挑選 a, b, c, d 四數，且經 HYL encoding 和補足後，a 和 b 的交集數量會大於 c 和 d，但若經過 RS encoding 後，c 和 d 的 0 集合交集數量加 1 集合交集數量反倒大於或等於 a 和 b 的 0 集合交集數量和 1 集合交集數量相加，我們定義此事發生的機率為混淆機率，統計所有此種例子後平均就可得到混淆機率，若此機率等於 50%，表示隨機取四數字 a, b, c, d 且交集數量是 a 和 b 比較大，則經過 RS code 後，a 和 b 仍然比較大的機率為 1/2，理論上混淆機率越高，低交集數越能超越高交集數的，攻擊者猜中交集數量的機率就越低。

因其交集數量與加密演算法中的 HMAC 和 bloom filter 無關，所以以下實驗都不做

HAMC 和 bloom filter。

5.2 實作方法

這次實驗使用 MATLAB 軟體套件，而以下為詳細實驗步驟，分為 3 部分

1. HYL encoding 實作

2. 實驗樣本產生

3. 結果計算

5.2.1 HYL encoding 實作

這裡主要敘述的是如何把一整數資料做成一元素個數為 bit 數的集合，且每個元素都為一整數，主要分成 3 部分，首先為產生一仿滿二元樹的數個矩陣當作加密演算法中隨機函數的值域，接者對資料作 HYL encoding 和補足，並把產生的元素轉成索引值，最後是用剛剛產生的索引值去找矩陣中的對應值，以下解說時以 bit 數為 10 為例。

1.產生矩陣:首先產生矩陣，因 10bit 的資料大小需要高度為 11 的滿二元樹，需要 4096 個數字，所以先執行下行。

Ran=randperm(4096)

此行將 1~4096 的數字隨機排列，並將結果存在 Ran 這個 1 X 4096 的矩陣裡面，並因為 RS code 的範圍在 0~4095 之間，所以把 Ran 裡所有元素都減一，接者仿造二元樹，我們以二元樹的層來存資料，比如說二元樹第一層有兩個節點，我們就取 Ran 的(1,1)和(1,2)這 2 個元素，將之存在 data1 這個 1 X 2 矩陣中，第二層有 4 個節點，就取 Ran(1,3)~Ran(1,6)這四個元素，將之存在 data2 這個 1 X 4 矩陣中，以次類推到最後

第11層，就取Ran的(1,2047)~Ran(1,4094)這2048個元素，將之存在data11這個1X2048的矩陣中，如此一來data1~data11實質上就是我們演算法裡的二元樹，如下圖 10:

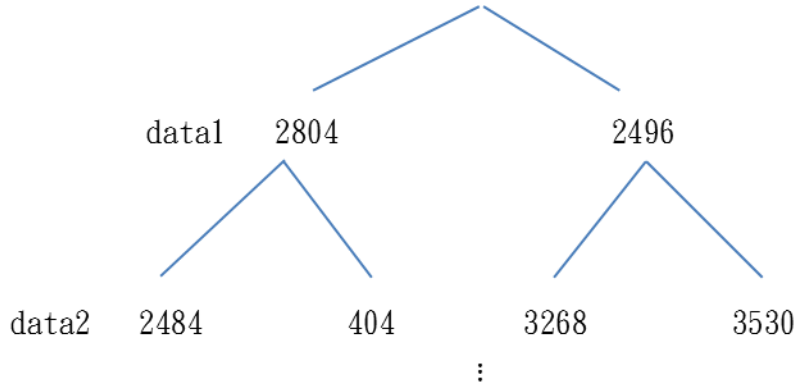


圖 10:使用隨機產生的值填滿後的二元樹

而照剛剛隨機函數演算法，若我要找11此元素的對應值，等於往右走2次，即為3530，而在此處我們必須把11換成索引值4，並在data2找第4個位置即可得3530，轉索引值的方法就把11換算成十進位並加1，在依據11長度為2，就等於在data2中找第4個位置，下面步驟會詳細說明。

2 實作HYL encoding和補足和轉化索引值

首先把資料轉化成二進位形式，此處舉例為428，其二元形式為0110101100，並存在binary這個1 X 10的矩陣。接者分成0集合和1集合部分:

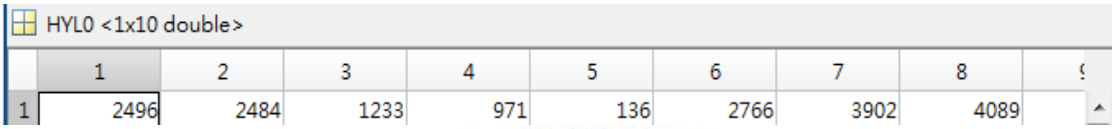
0集合:先產生一1 X 10的矩陣HYL0，接者依照HYL encoding，依序抓取binary中的前n個數字，並看其結尾，若為0，表示其本為HYL encoding產生的0集合元素，則依HYL encoding，將其尾巴改成1，並換算成索引值儲存到HYL0(換算索引值方法為換算成十進位後加1)，若結尾為1，表示其本為HYL encoding產生的1集合元素，會透過補足轉成0集合元素，則照補足方法，將其尾巴改成0後算索引值儲存到HYL0。

1集合:先產生一1 X 10的矩陣HYL1，接者依照HYL encoding，依序抓取binary中的前n個數字，並看其結尾，若為1，表示其本為HYL encoding產生的1集合元素，則依HYL encoding，直接將其換算成索引值儲存到HYL1，若為0，表示其本為HYL encoding

產生的 0 集合元素，會透過補足轉成 1 集合元素，依照演算法先將尾巴改成 1，接者後面加 11-n 個 0，之後再將其換算成索引值並儲存到 HYL1。

3. 依照索引值取得對應整數，此處也分為 0 集合和 1 集合

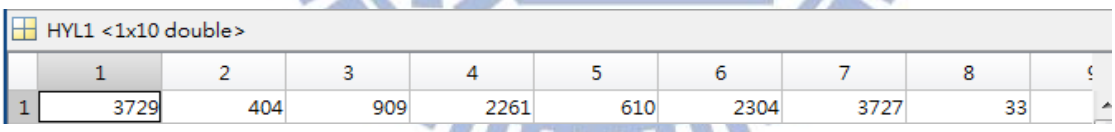
0 集合:因為 0 集合其每個元素的長度剛好依序 1~10，所以就按找 HYL0 裡的索引值，第一個去 data1 找，第二個去 data2 找，以此類推，並直接覆寫 HYL0，如下圖 11


HYL0 <1x10 double>

	1	2	3	4	5	6	7	8	9
1	2496	2484	1233	971	136	2766	3902	4089	

圖 11:428 的 0 集合

1 集合:1 集合比較不一樣的一點是其元素長度沒有依序排，所以需檢查原本 binary 矩陣中，若該位置是 0 的話，則此位置的索引值是用在 data11，而若該位置是 1，則就和 0 集合一樣的做法，若是第 2 的位置則去找 data2，以 428 為例，第一個 bit 是 0，則表示其第一個索引 1025 是用在 data11 的，而第 2 個 bit 是 1，則其第 2 個索引 2 是用在 data2 的，按照索引找到該值，覆寫到 HYL1，完成後如下圖 12:


HYL1 <1x10 double>

	1	2	3	4	5	6	7	8	9
1	3729	404	909	2261	610	2304	3727	33	

圖 12:428 的 1 集合

5.2.2 實驗樣本產生

接下來是產生樣本，我們對每個實驗都隨機取 10 組 100 個相異數字，並以平均當結果，首先是取 100 個相異數字，我們使用 Ran=randperm(1024)，將 1~1024 隨機排列，並取前 100 個數字減一當成樣本，並照 5.2.1 的方法將對這 100 個數字作 HYL encoding 和補足，並轉成 10 X 100 矩陣 data。

接者我們需要檢測每筆資料間的交集，由於經補足後，任兩數其彼此之間的 0 集合交集個數都等於 1 集合交集個數，所以在這邊只對 1 集合作檢測，使用 2 層 for 迴圈跑出結果，存在 100 X 100 矩陣中，如下圖 13:

	1	2	3	4	5	6	7	8	9
1	10	1	0	0	0	1	1	2	1
2	1	10	0	0	0	3	4	1	3
3	0	0	10	3	2	0	0	0	0
4	0	0	3	10	2	0	0	0	0
5	0	0	2	2	10	0	0	0	0
6	1	3	0	0	0	10	3	1	4
7	1	4	0	0	0	3	10	1	3
8	2	1	0	0	0	1	1	10	1
9	1	3	0	0	0	4	3	1	10
10	0	0	2	2	6	0	0	0	0
11	3	1	0	0	0	1	1	2	1
12	0	0	2	2	4	0	0	0	0
13	0	0	2	2	3	0	0	0	0
14	1	3	0	0	0	5	3	1	4
15	1	2	0	0	0	2	2	1	2
16	0	0	4	3	2	0	0	0	0
17	1	5	0	0	0	3	4	1	3
18	0	0	5	3	2	0	0	0	0
19	0	0	1	1	1	0	0	0	0
20	0	0	4	3	2	0	0	0	0

圖 13:原先交集數之矩陣

每一點(x,y)，代表第 x 筆資料和第 y 筆資料有多少個交集，如第一筆資料 506 和第 2 筆資料 57 就沒有交集(因第一個 bit 不同)，而第 2 筆資料 57 和第 7 筆資料 7 就有 4 個交集(兩者前 4bit 都是 0000)。

5.2.3 結果計算

接下來我們將對樣本進行 RS encoding，RS encoding 是引用自 MATLAB 中 Communication System Toolbox 這個擴充套件中的 comm.RSEncoder System object，以下是詳細指令:

```
hEnc = comm.RSEncoder
hEnc. MessageLength=10
```

hEnc.CodewordLength=26

hEnc.PrimitivePolynomialSource=’ Property’

hEnc.PrimitivePolynomial= [1 0 0 0 0 0 1 0 1 0 0 1 1]

首先第一行是宣告 RS encoding 物件，其他行是改裡面的參數，第一個 MessageLength 是指要做 encoding 的集合大小，因 10bit 資料作 HYL encoding 和補足後集合大小是 10，所以此設為 10，而 CodewordLength 則是 encode 後的集合大小，在這裡設為 26，也就是 RS encoding 過後增加了 16 個元素，第三個 PrimitivePolynomialSource 是 PrimitivePolynomial 是自動產生或是自己輸入，因他自動產生會以 MessageLength 為準，10 就是在 GF(1024)底下的 Primitive Polynomial$1+x^3+x^{10}$，而 RS code symbol 總數也為 1024，但因為我們是在 GF(4096)底下作 encode，所以要將此行改成 Property 讓我們自己輸入，而 GF(4096)底下的 Primitive Polynomial 為$1+x+x^4+x^6+x^{12}$，由次方高打到次方低，也就是[1 0 0 0 0 0 1 0 1 0 0 1 1]

完成後打 hEnc 會顯示屬性如下圖 14

```
>> hEnc

hEnc =

  System: comm.RSEncoder

  Properties:
                    BitInput: false
              CodewordLength: 26
               MessageLength: 10
    PrimitivePolynomialSource: 'Property'
         PrimitivePolynomial: [1 0 0 0 0 0 1 0 1 0 0 1 1]
    GeneratorPolynomialSource: 'Auto'
        PuncturePatternSource: 'None'
```

圖 14:MATLAB 中的 RS encoding

在 encoding 之前，我們必須把要 encoding 的資料切成(10*100)X1 的矩陣，使用

reshape 把 data 切成 1000*1，所以打入指令 data=reshape(data,1000,1) ，接者再對其作 encoding，打入 encodedData=step(hEnc,data)就好，而之所以要切矩陣的原因是因為他預設是吃(MessageLength*k) X 1 的矩陣，k 為正整數，而每次 encoding 就去抓去 MessageLength 個元素來 encode，比如說第一次 encoding 就抓(1,1)~(10,1)，第二次抓(11,1)~(20,1)，而 encoding 完後把資料依序存在 encodedData，第一次 encoding 的存在(1,1)~(18,1)，第二次存在(19,1)~(36,1)，以此類推，encoding 完後的矩陣 encodedData 為一 1800*1 矩陣，我們將他切成 18*100。

而接者我們要統計經 encode 完後的交集數量，在這裡要注意因為我們的加密演算法前 10 個和後 16 個是用不同 key 作 HMAC，所以在算交集個數時，前 10 個直接引用在 encoding 之前的交集數量矩陣，而在這裡我們只計算後 16 個產生的交集數量再加上先前的矩陣，結果如下圖 15 圖 16:

0 集合

	1	2	3	4	5	6	7	8	9
1	26	1	0	0	0	1	1	2	1
2	1	26	1	0	0	3	4	1	3
3	0	1	26	3	2	0	0	1	0
4	0	0	3	26	2	0	0	0	0
5	0	0	2	2	26	0	0	0	0
6	1	3	0	0	0	26	3	1	4
7	1	4	0	0	0	3	26	1	3
8	2	1	1	0	0	1	1	26	1
9	1	3	0	0	0	4	3	1	26
10	1	0	2	2	6	0	0	0	0
11	3	1	0	0	0	1	1	2	1
12	0	0	3	2	4	0	0	0	0
13	0	0	2	2	3	0	0	0	0
14	1	3	0	0	0	5	3	1	4
15	1	2	0	0	0	2	2	1	2
16	0	0	4	3	3	0	0	0	0
17	1	6	0	0	0	3	4	1	3
18	0	0	5	3	2	0	0	0	0
19	0	0	1	1	1	0	0	0	0
20	0	0	4	3	2	0	0	1	0

圖 15:RS encoding 後的 0 集合交集數矩陣

1 集合:

	1	2	3	4	5	6	7	8	9
1	26	1	0	0	0	1	1	2	1
2	1	26	0	0	0	3	4	1	3
3	0	0	26	4	2	0	0	0	0
4	0	0	4	26	2	0	0	0	1
5	0	0	2	2	26	0	0	0	0
6	1	3	0	0	0	26	3	1	4
7	1	4	0	0	0	3	26	1	3
8	2	1	0	0	0	1	1	26	1
9	1	3	0	1	0	4	3	1	26
10	0	0	2	2	6	0	0	0	0
11	3	1	0	0	0	1	1	2	1
12	0	0	2	2	4	0	0	0	0
13	0	0	2	2	3	0	0	1	0
14	1	3	0	0	0	6	3	1	4
15	1	2	0	0	0	2	2	1	2
16	0	0	4	3	2	0	0	0	0
17	1	5	0	0	0	3	4	1	3
18	0	0	5	3	2	0	0	0	0
19	0	0	1	1	1	0	0	0	0
20	0	0	4	3	2	0	0	0	0

圖 16:RS encoding 後的 1 集合交集數矩陣

再根據此交集資料，和 encode 前的交集資料，作出一三維矩陣，前 2 維度為 RS encoding 後 0 集合交集數量和 1 集合交集數量，最後一維為 RS encoding 前交集數量，而裡面的值則為符合此行情況的資料對個數，100 筆資料總共有 4950 個資料對，如下圖 17

```
val(:,:,1) =

  Columns 1 through 8

        2191         153           8           0           0           0           0           0
         125           7           0           0           0           0           0           0
           6           1           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
```

圖 17:表現交集數關係的三維矩陣

如圖，最後一個維度為 1 表示此 page 內的資料對原先交集數都為 0，點(x, y)表示經 RS encoding 後的 0 集合交集個數和集合 1 交集個數，如(1,1)表示 RS encoding 後，0 集合交集個數和 1 集合交集個數皆為 0 交集，而此種情況的資料對有 2191 組，最後是算機率，根據 5.1 的定義，我們用此資料算出 2 種機率，猜中機率和混淆機率。

5.3 結果分析

這節我們將用 5.1 的標準和 5.2 的方法來檢測 RS encoding 的混淆效果，首先定義變數如下:

1. 明文資料 bit 數 x

2. 混淆個數 r(新增加的元素個數)

3. 使用 RS(x + r, x) encoding

接者分別對9, 10, 14 bit 資料做實驗，混淆個數為16的倍數(r=16*k, k=1~10, 15)，結果如下圖 18 圖 19:

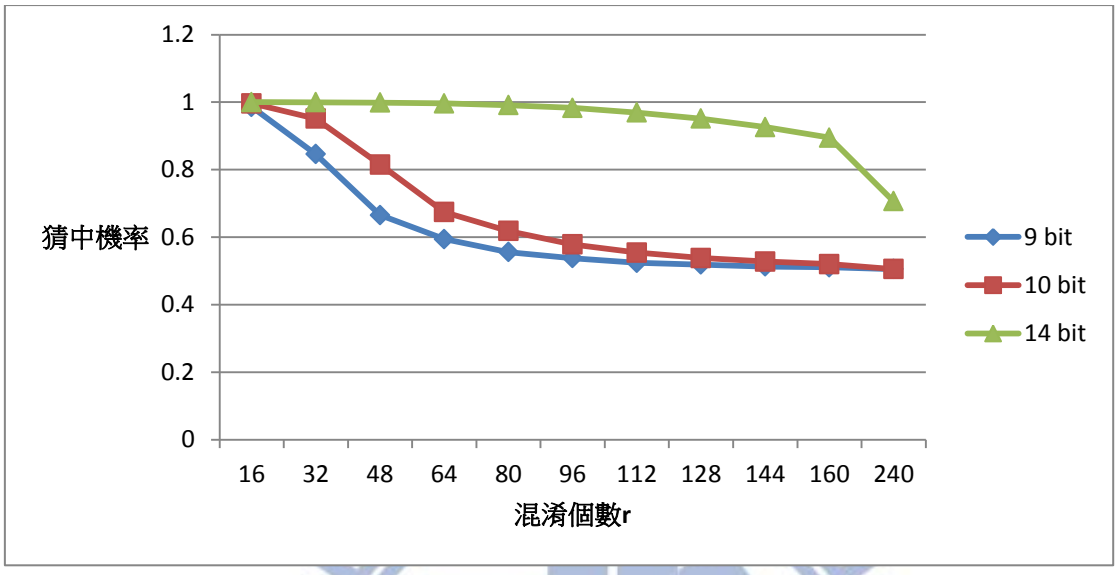


圖 18:猜中機率折線圖(no mod)

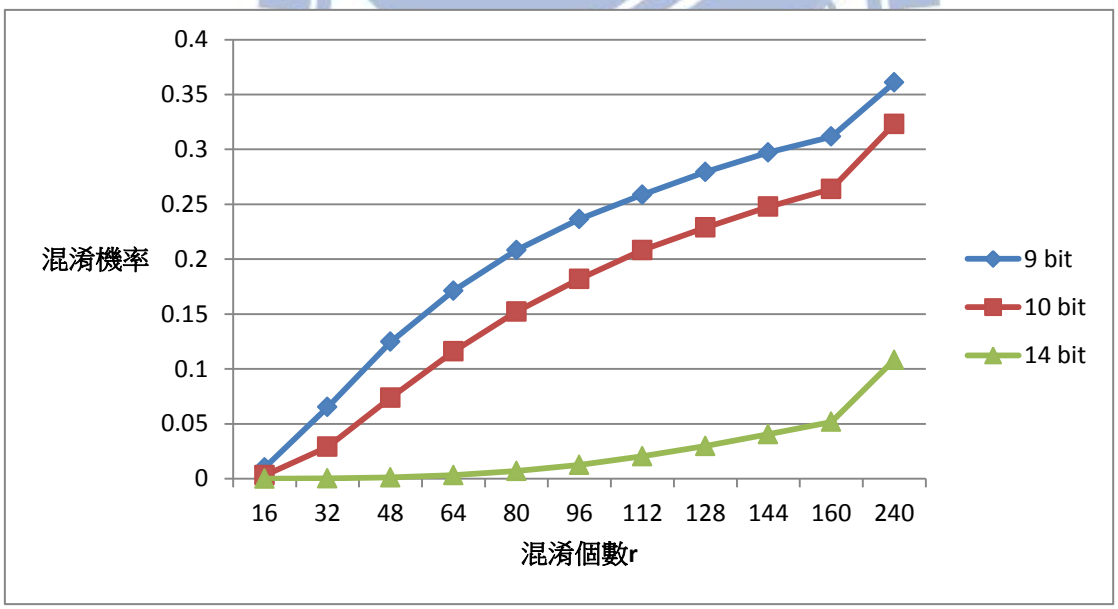


圖 19:混淆機率折線圖(no mod)

首先看到隨者混淆個數上升，效果一定會越來越好，而在相同的混淆個數下，因為明文 bit 數越多，RS code symbol bit 數也越高，則越難產生碰撞，效果就會變差，而猜中機率混淆個數為 240 的情況下，明文 9bit 和 10 bit 只有 50.5%，但明文 14bit 卻

還是有 70%。而混淆機率也是一樣，在混淆個數為 240 的情況下，9bit 和 10bit 都超過 30%，但 14bit 只有 10%左右。而此外我們也想檢查這些被猜中的資料，其到底是 0 交集被猜中還是非 0 交集被猜中，因為猜中機率在最好情況下，攻擊者只能猜 0 交集，所以我們把 0 交集視為較不重要的資訊，希望其在被猜中的資料中能佔有很高的比例，而結果如下圖 20

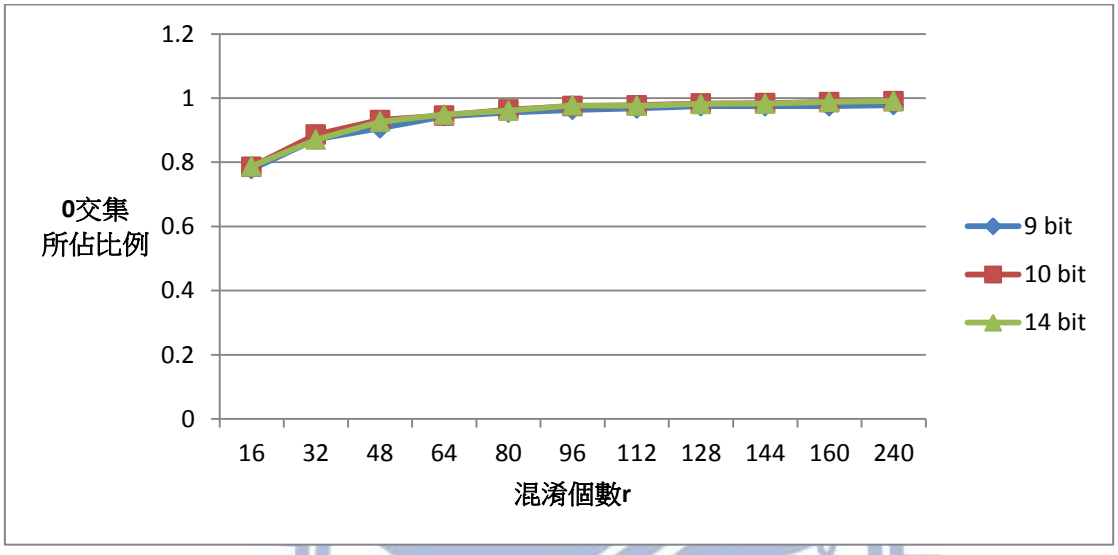


圖 20:0 交集所佔比率折線圖(no mod)

可以看到一開始不論明文大小都在 50%徘徊，而混淆個數超過一定值後便開始上升，這是因為 0 交集的資料對獲得的交集數可以壓過其他交集的資料對，而明文 bit 數越高，此現象發生越慢，明文為 14bit 的資料必須在混淆個數超過 160 後才有此現象，而 9bit 只需要 48 個混淆個數。

5.4 mod 方法

探討前面的實驗可以知道說，以明文 9bit 為例，給其 240 個混淆個數便可達到很高的效果，但若明文為 14bit，240 個混淆個數顯然不夠，而有沒有辦法加速碰撞，讓我們只要使用較少的混淆個數就能達成不錯的效果，因此我們想到 mod 方法，把經 RS encoding 新增的元素全部取 mod，增加其碰撞的機率，如下例:

RS（26,10）encoding 中明文和其 0 集合新增的元素

506:2329, 245, 2890, 3174, 1191, 795, 3804, 3035, 3699, 144, 542, 2682, 1462, 1399, 4048, 1335

57:2038, 2981, 3142, 2037, 316, 4094, 3669, 2494, 2612, 3847, 980, 2376, 2829, 3135, 2803, 2710

可以看到這 2 個明文 506, 57 新增的 16 個元素沒有交集，但要是我們對這些新增元素做 mod 32 的話就不一樣了

506: 25, 21, 10, 6, 7, 27, 28, 27, 19, 16, 30, 26, 22, 23, 16, 23

57: 22, 5, 6, 21, 28, 30, 21, 30, 20, 7, 20, 8, 13, 31, 19, 22

在扣掉重複後，共有 7 個交集如上，由此可見做 mod 會大大增加碰撞的機會，於是修改我們實驗方法如下:

1. 當做完 RS code 後，把新增元素都做 mod m，並對重複的元素做調整(第一個於數 1 為 1，第二個餘數 1 為 m+1，第三個餘數 1 為 2m+1，以此類推)。

2. 之後一樣比較交集和算機率。

首先我們針對 mod 32 去做實驗，效果如下圖 21 圖 22 圖 23

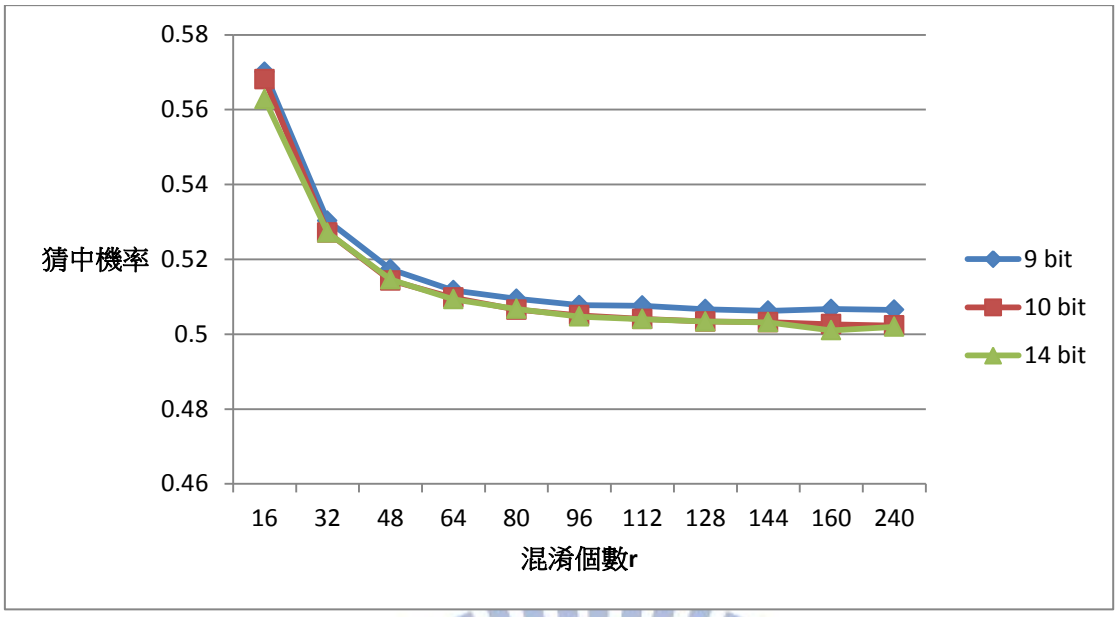


圖 21:猜中機率折線圖(mod 32)

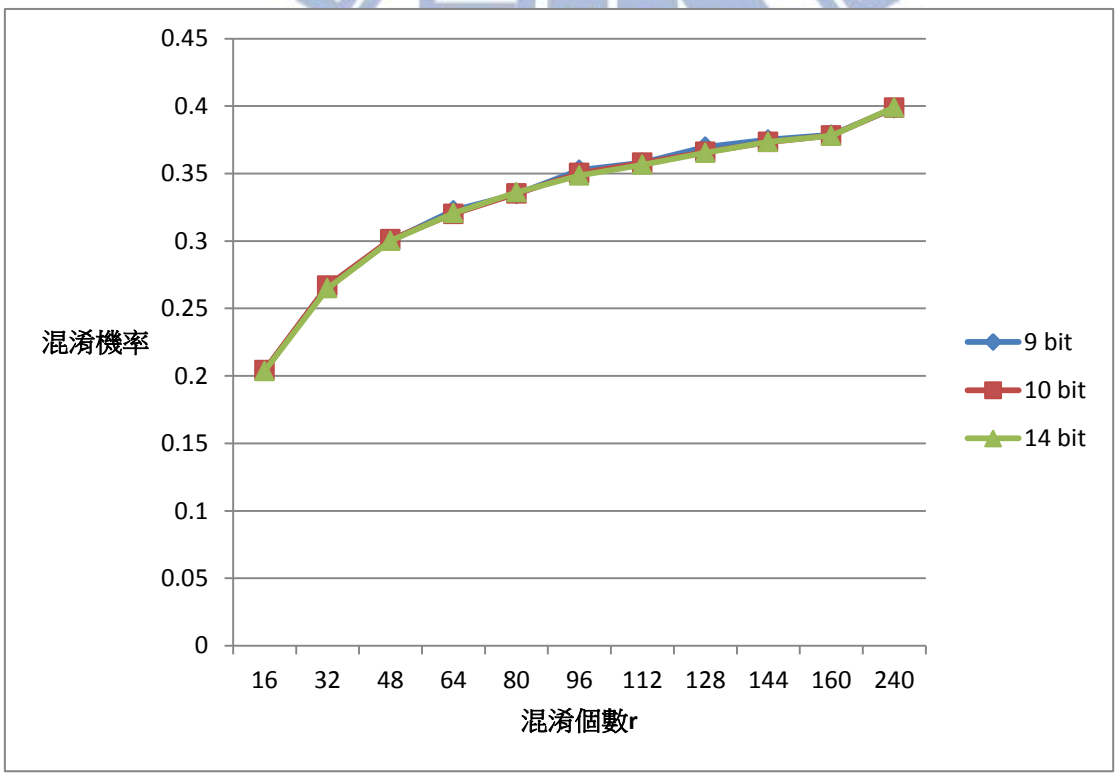


圖 22:混淆機率折線圖(mod 32)

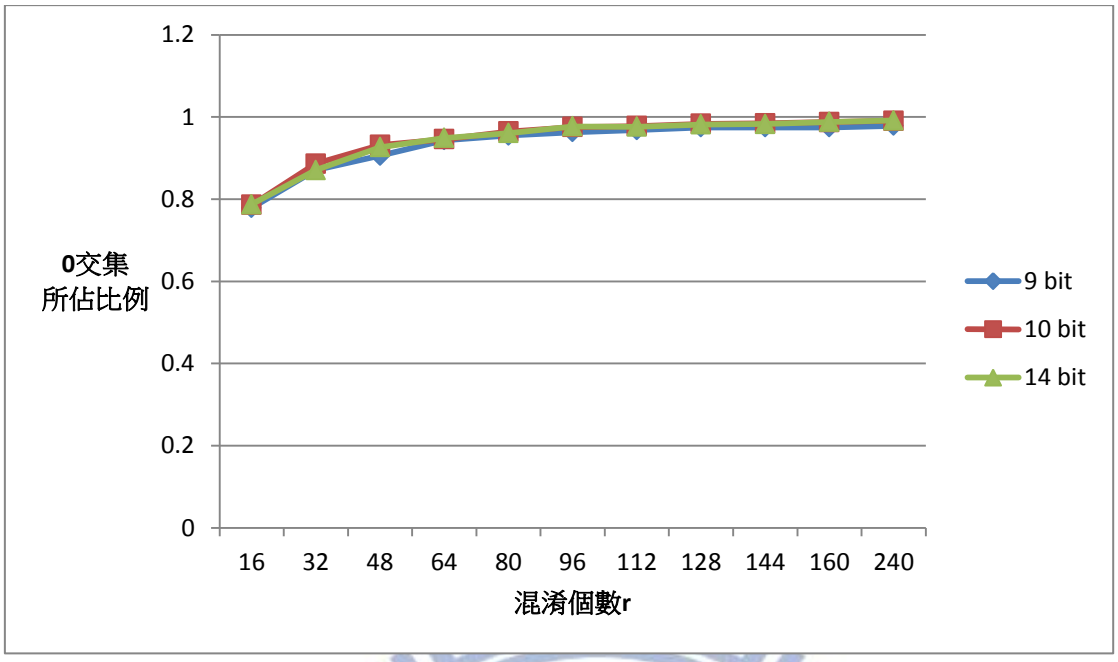


圖 23:0 交集所佔比率折線圖(mod 32)

發現效果變得非常突出，即使混淆個數只有 8，在 10 bit 的情況下都比沒有做 mod，混淆個數為 160 的一樣好，更甚者，此方法因為把 RS encoding 出來的元素都壓在 mod 32 裡面，所以讓混淆的效果不隨者資料 bit 數變多而變差，對於任何數量的資料大小，我們都可以以相同的混淆個數達到相同的混淆效果，而我們接者針對 mod 的數字進行測試，是否 mod 的數字越小，碰撞機率越高時，混淆效果也會更好，我們測試 mod 2~mod 1024，$(2^1 \sim 2^{10})$在 14bit 的情況，結果如下圖 24 圖 25 圖 26

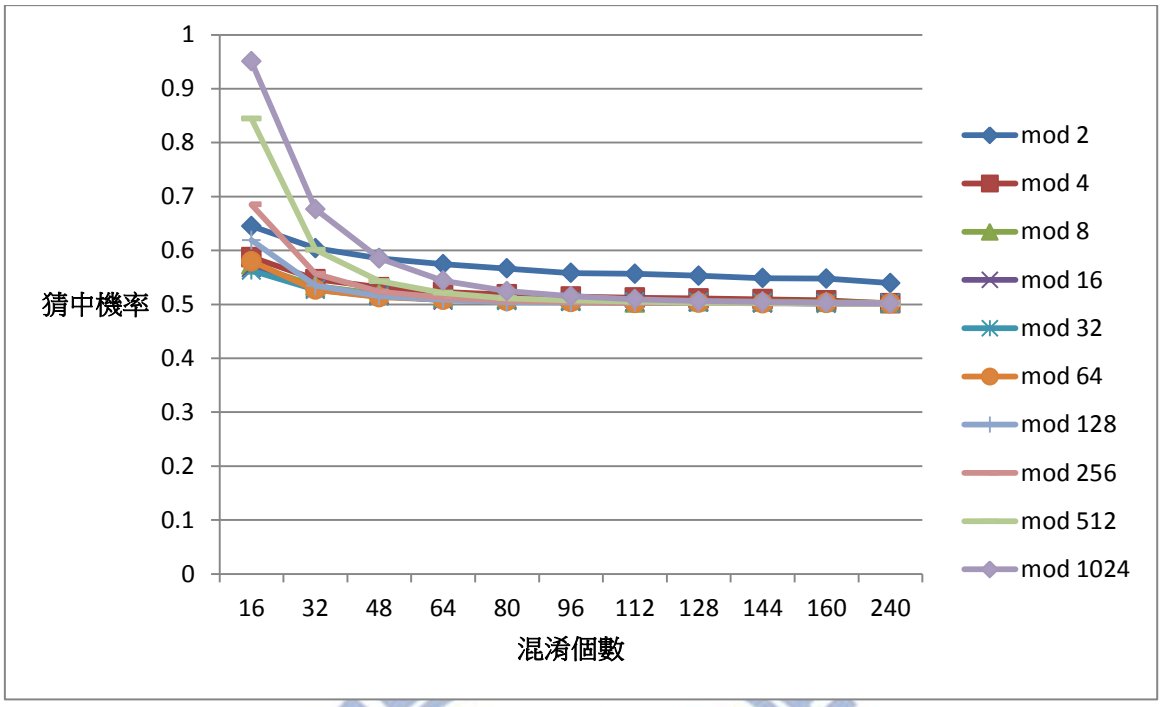


圖 24:猜中機率折線圖(mod 數字 2~1024)

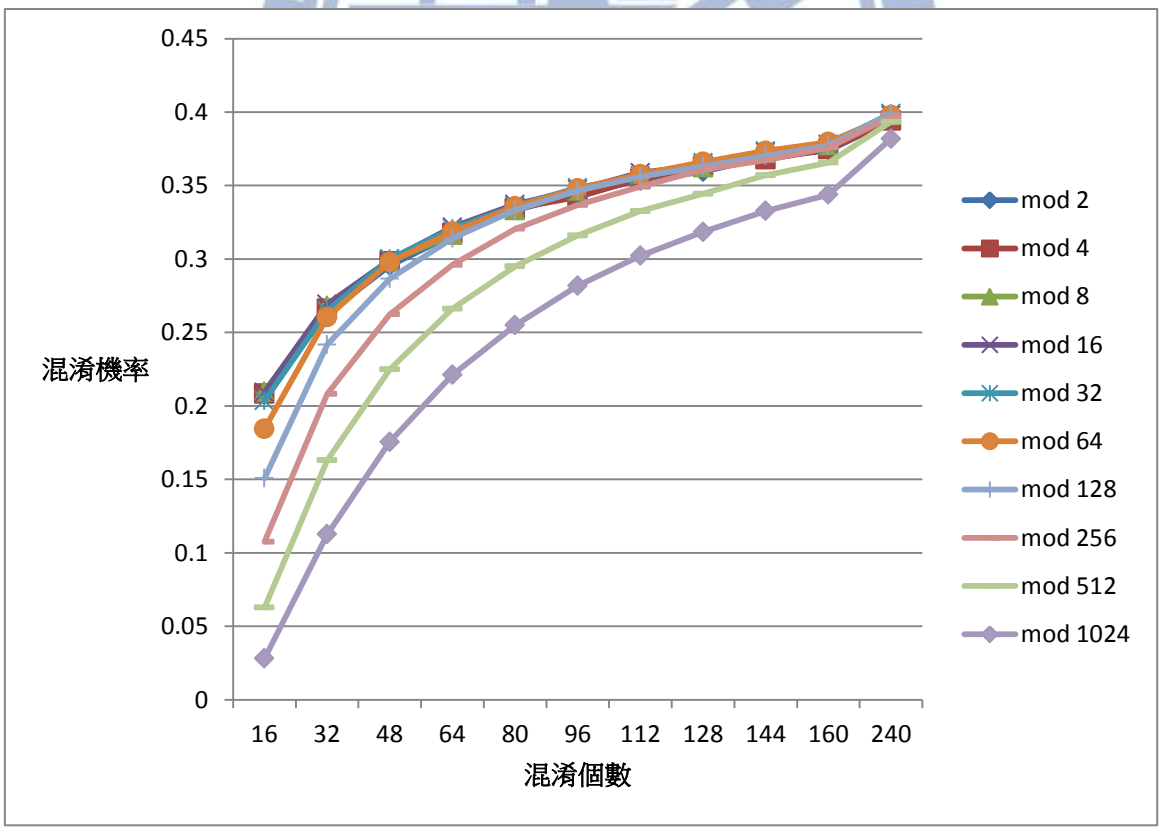


圖 25:混淆機率折線圖(mod 數字 2~1024)

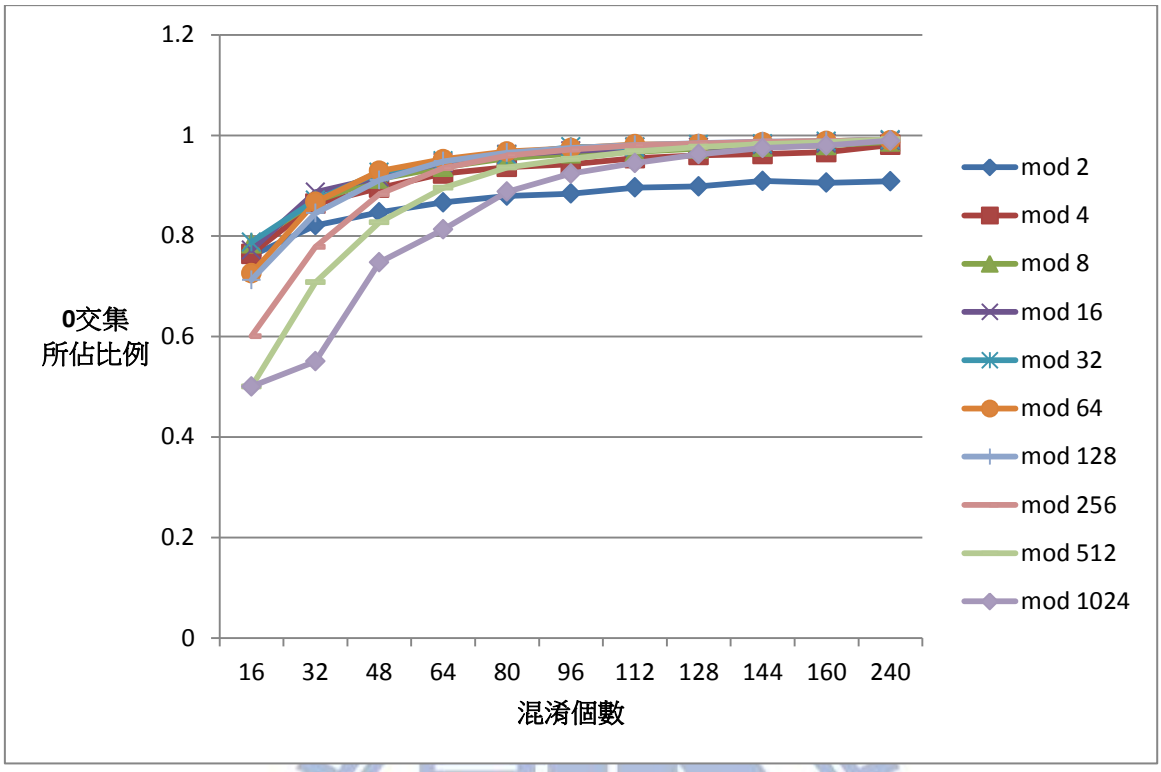


圖 26: 0 交集所佔比率折線圖(mod 數字 2~1024)

首先觀察猜中機率，發現除了 mod 2 外，其他 mod 值在混淆個數超過 80 時都會收斂成一條線，而在混淆機率方面，較大的 mod 值在混淆個數小時會比較差，但隨者混淆個數的增加，會逐漸逼近較小的 mod 值，所以在使用 mod 方法時，mod 的值在 4~32 之間的效果是最好的，而使用其他的 mod 值可能會需要更多的混淆個數來達成相同的效果。

5.5 效能

而最後我們來測試我們演算法在執行 range query 上會花的時間，我們隨機產生 1 萬的 14 bit 的資料，並使用 RS (94, 14) encoding，也就是密文集合大小為 94，而 bloom filter 長度為 5000，hash 函數個數為 40，每次測試元素是否在集合內的失誤機率為$\frac{1}{2^{40}}$，所以單次比較失誤機率約為$\frac{94}{2^{40}}$，每次 query 失誤機率約為$\frac{1}{2^{20}}$，然後針對以下 3 種情況做測試:

1. 不使用 bloom filter 和索引，也就是在比較密文大小時，直接拿兩個集合做字串比

對找交集。

2. 使用 bloom filter、不使用索引，比較大小時同上面的演算法。

3. 使用 bloom filter 和二元樹索引，先在資料上建立二元樹索引，而後在執行 query 時只需和二元樹內的節點值做比較。

而三種方法所需的計算複雜度和實際運算所需時間如下表格 4

n=16384, r=80, k=40	比較大小演算法複雜度	比較大小所花總時間
不使用 bloom filter 和索引	$O((logn+r)^2)$字串比對	180 秒
使用 bloom filter、不使用索引	$O((\log n+r)*k)$ hash 函數	1.21 秒
使用 bloom filter 和二元搜尋樹索引	$O((\log n+r)*k)$ hash 函數	0.004 秒

表格 4::效能比較表

首先可看到 bloom filter 很有效加速我們的演算法，降低了 300 倍左右，雖然也伴隨者失誤機率，而使用索引雖然單次比較大小時間一樣，但有效降低比較次數，使得整體所花的時間更短。

六、問題討論

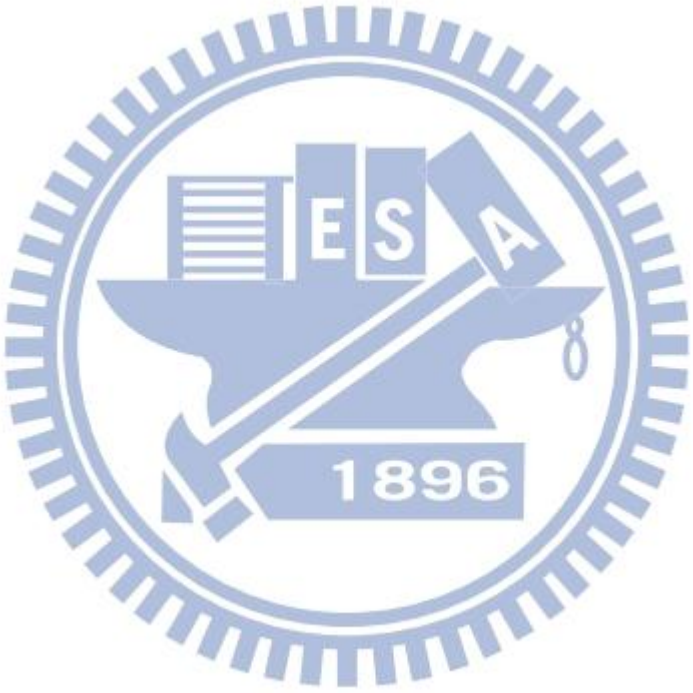
雖然以上實驗顯示我們的演算法加上 RS code 和 mod 方法會有很不錯的效果，但其實還有幾個問題待解決:

1.統計問題:這個問題是有關 mod 方法的問題，在使用 mod 方法時，若混淆個數太大，可能導致某些元素必定出現，比如說使用 mod 16 和混淆個數 80，則第一個餘數 0~15 出現機率接近 1，而 HYL encoding 中最常出現的元素，其機率也不過$\frac{1}{2}$，則攻擊者在收集足夠多資料後，便可從元素出現機率來判別這些元素皆為加進來混淆的，降低我們的混淆效果，解決方法可能要針對不同的混淆個數來決定 mod 值大小，比如說使用混淆個數 40，則我 mod 值就用 80，來讓元素出現機率降到最高$\frac{1}{2}$，或乾脆使用較大的 mod 值如 256 和 512，而這些 mod 值在混淆個數多時，效果與其他 mod 值差不多。

2.1 集合與 0 集合交集問題:這問題就比較嚴重,前面我們都忽略 1 集合和 0 集合的交集，因其代表大小關係，但其實它還會透漏一個資訊，就是如果有交集的話，則此元素必為 HYL encoding 所產生的元素,而要是攻擊者收集足夠多的資訊,就可以分出 HYL encoding 的元素和混淆產生的元素，進而使我們的混淆方法徒勞無功，而目前還未想到有效解決此問題的方法，有個想法是我們本來對 0 集合和 1 集合，透過 RS encoding 新增的元素用不同的 key 做 HMAC,而把上述的方法改成用相同的 key 做 HMAC,使其隨機產生交集，並定出一數 m，只要交集數超過 m 便算有交集，但因為 1 集合和 0 集合的原先交集數量只有可能是 0 或 1，而被 RS encoding 和 mod 方法混淆的機率很大，使得 m 值不管定多少都很難分離原先交集數量 0 和原先交集數量 1，所以這問題還有待解決。

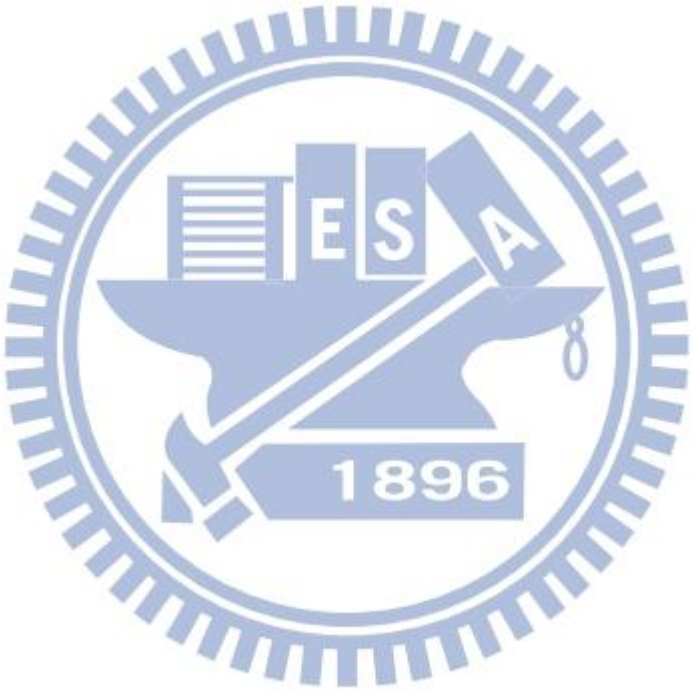
3.接下來探討因使用 bloom filter 而失誤的機率，而其每次比較大小的失誤機率為(集合元素個數)/2^k，k 為 hash function 個數，所以說集合大小越大，雖然混淆效果越好，但是失誤機率也越大，解決方法就是盡可能使用多一點的 hash function，雖然這樣做 bloom filter 會加長。而若我們假設 bloom filter 長度為 i 倍的集合元素個數，則 hash

function 個數為 0.7i，每次比較大小失誤機率為$\frac{(\text{集合元素個數})}{2^{0.7i}}$ ，因此在集合元素個數為 94 的情況下，若要保持$\frac{1}{2^{20}}$的失誤機率，則 bloom filter 的大小就約為 5000。



七、結論

我們提出一以 HYL encoding 為主的加密演算法可支援 range query，並洩漏大小關係方便伺服器建立索引加速 range query 的執行，密文為一集合大小為 O(log n)，比較所需時間為 O((log n)* k)個 hash 函數，n 為明文範圍大小，而分析且證明其安全性高於[1]，並提出 HYL encoding 會造成的問題，並嘗試使用 RS encoding 來增加密文集合，使得密文集合大小為 O(log n+ r)，比較大小時間為 O((log n+ r)* k)個 hash 函數，且以實驗秀出 RS encoding 所造成的效果，最後再以 mod 方法來使得效果只跟 r 值有關，與明文大小無關。



八、參考文獻

[1] Alexandra Boldyreva, Nathan Chenette, Younho Lee, Adam O'Neill, "Order-Preserving Symmetric Encryption", EUROCRYPT 2009, pp. 224-241

[2] MIT Faculty, "Relational Cloud: A Database-as-a-Service for the Cloud", http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper33.pdf

[3] Hasan Kadhem, Toshiyuki Amagasa, Hiroyuki Kitagawa, "MV-OPES: Multivalued-Order Preserving Encryption Scheme: A Novel Scheme for Encrypting Integer Value to Many Different Values", IEICE Transactions on Information and Systems, VOL.2010; NO.9; PAGE.2520-2533

[4] Bernard Sklar, "Reed-Solomon Codes", http://ptgmedia.pearsoncmg.com/images/art_sklar7_reed-solomon/elementLinks/art_sklar7_reed-solomon.pdf

[5] Alexandra Boldyreva, Nathan Chenette, Adam O'Neill, "Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions", CRYPTO 2011

[6] Liangliang Xiao, I-Ling Yen, "A Note for the Ideal Order-Preserving Encryption Object and Generalized Order-Preserving Encryption", IACR Cryptology ePrint Archive, Volume 2012

[7] Hsiao-Ying Lin, Wen-Guey Tzeng, "An Efficient Solution to the Millionaires' Problem Based on Homomorphic Encryption", ACNS 2005, pp. 456-466

[8] Fei Chen, Alex X. Liu," Privacy- and Integrity-Preserving Range Queries in Sensor Networks", IEEE/ACM TRANSACTIONS ON NETWORKING 2012

[9] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz, "Theory and Practice of Bloom Filters for Distributed Systems", IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 14, NO. 1, FIRST QUARTER 2012

[10] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu, ” Order Preserving Encryption for Numeric Data”, SIGMOD 2004

[11] Dongxi Liu, Shenlu Wang, “Programmable Order-Preserving Secure Index for Encrypted Database Query”, IEEE Fifth International Conference on Cloud Computing 2012

[12] Liangliang Xiao, Osbert Bastani, I-Ling Yen, “Security Analysis for Order Preserving Encryption Schemes”, CISS 2012

[13]Keke Chen, Ramakanth Kavuluru, Shumin Guo, “RASP: Efficient Multidimensional Range Query on Attack-Resilient Encrypted Databases”, CODASPY 2011