以格式保留演算法 FF1 的身分證字號加密實作
(Encrypting Identification Card Number by Format-preserving Encryption FF1 Algorithm)

楊慶隆　黃擴增　曾誌嶽　林泓毅
國立東華大學資訊工程學系
E-mail: cnyang@gms.ndhu.edu.tw

摘要

格式保留加密(Format-preserving Encryption; FPE)是密文仍保留明文格式的加密方法。一般加密演算法並無此特性。例如AES加密的輸出是隨機雜亂的 128 個位元。雖然確保了機密性，但是格式、或長度變化後對某些舊系統可能會出現問題。這些系統的數據模式可能無法更改，或者是軟硬體更新、改造成本過高，為了解決這樣的困擾FPE加密技術應運而生。FPE主要應用於保護常被使用和儲存的敏感資料。所謂的敏感但未分類 (Sensitive But Unclassified; SBU)資料、與機密資料都是敏感的均要加以保護，兩者之間的差異僅在敏感程度。FPE經常用於保護身分認證或其他日常生活應用的SBU資料。例如: 信用卡號、銀行卡號、社會安全碼、身份證字號、護照號碼、甚至是個人的姓名、出生年月日、PIN 號碼、電子郵件帳號等。在本文中，我們以 NIST 的格式保留加密法 FF1 來加密國民身分證字號，其加密後也是另一個符合規則的隨機身份證字號。我們的設計是依據中華民國國民身份證字號的發放現況、並採用類似AES的XTS-AES模態利用儲存位置當Tweak參數。根據儲存索引位置的不同，相同的身分證字號也會有不同的加密身分證字號。

關鍵詞：格式保留加密、敏感資料、FF1、AES-CBC模態、身份證字號。

Abstract

Format-preserving Encryption (FPE) is an encryption method where the ciphertext still retains the plaintext format. In general, most ciphers do not have the feature. For example, the AES block cipher has a random 128-bit output. However, some legacy systems may have problems after changing the format or length. Maybe, the data mode of these legacy systems cannot be able to be changed, or the cost of software and hardware updates are too high. For solving such problems, FPE encryption technology came into being. FPE is used to protect sensitive data. The sensitive but unclassified (SBU) data and classified data are both sensitive and must be protected, their difference is only in the degree of sensitivity. FPE protects SBU data used for identity verification or daily life applications, e.g., credit card number, bank card number, social security number, social security number, passport number, even personal name, date of birth, PIN number, email account number, etc. In this work, we use NIST FF1 to encrypt the national ID number. Our design is based on the status of issuing national ID numbers in Republic of China, and meantime adopts the XTS-AES mode using the storage location as the tweak parameter. Because of using the location of the storage index as a tweak, the same ID number in different locations will have different encrypted ID numbers.

Keywords：Format-preserving Encryption (FPE), Sensitive data, FF1, AES-CBC mode, Identification card number.

1. 前言

格式保留加密((Format-preserving Encryption; FPE)的特性就是加密後的密文保留原來明文的格式，例如 16 位數字卡號可加密為 16 位數字的信用卡號。FPE 加密技術主要是保護常被使用和儲存的敏感資料。所謂的敏感但未分類 (Sensitive But Unclassified; SBU) 資料、與機密資料都是敏感的均要加以保護，兩者之間的差異僅在敏感程度。一般 FPE 保護的 SBU 資料，例如: 零售、醫療保健、和金融資料庫中常用的信用卡號、社會安全碼、身份證字號、護照號碼、甚至是個人的姓名、出生年月日、PIN 號碼、電子郵件帳號等。上述信息、及個資長久以來都使用於身分認證、或其他日常生活應用。例如參考文獻 [1] 提出了一種可以用於信用卡號、電子郵件、日期的FPE加密方法。參考文獻 [2-5]，分別對中文名字 [2]、信用卡號 [3, 4]、及 GPS 訊息 [5] 使用 FPE 的加密方式。在不那麼注重個資的年代，許多儲存 SBU 資料的資料庫，並沒有對這些資料做適當的保護。FPE 好處是保留原始數據的格式和長度，這兩種特性可使用於資料格式特定的舊系統 [6]，所以不需要常更新代碼與系統。這也是為什麼FPE加密技術應運而生，常用於零售、醫療、金融等環境的原因。與傳統加密相比，FPE 的另外一個優點是我們可以對 FPE 加密後的數據進行大量處理 (因為它格式和長度都與加密前一樣)，僅在需要使用時才轉成明文。這功能有全同態加密 (Fully Homomorphic Encryption; FHE) 的特性，但是相較於 FPE，FHE 複雜多了。使用 FPE 加密 SBU 資料的另一個好處是我們不希望因為加密導致這些 SBU 資料受到額外的關注。

傳統加密產生的隨機雜亂資料很容易察覺，但 FPE 加密並不容易發現資料是否被加密可減少對資料惡意破壞。

本文是依據中華民國國民身份證字號的發放現況，以 NIST 的格式保留加密演算法 FF1 完成身分證字號加密。本文結構如下，第二部分為文獻探討，介紹了身分證字號產生規則、及格式保留加密演算法 FPE。第三部介紹使 FF1 演算法的身分證字號加密，包含了設計概念、及加/解密實作。第四部分為實作結果，最後則為結論。我們也把實作的程式碼放於附錄中。

2. 文獻探討

2.1 國民身分證字號產生規則

中華民國國民身份證字號由 10 碼組成，第一碼為大寫英文字母，其餘九碼為阿拉伯數字。第一碼的大寫英文字母代表國民第一次所報的戶籍地區，而二十六個英文字母中有些字母因為時代的演變目前已經沒有再使用。在 1974 年以後陽明山管理局虛位化，所以原先使用的Y已停止發放。而 2010 年以後台中縣的 L、台南縣的 R 與高雄縣的 S 因為縣市合併為直轄市的緣故也都停止發放。所以現今還有繼續在發放的字母只有 22 個。第二碼為首位的數字碼，用來區分性別 1 代表男性 2 代表女性，在 110 年後增加了 8 跟 9 分別代境外移入的男性與女性。第三碼原本是流水號的一部份但在 2003 年後，分別使用 6、7、8、9 來代表在台灣取得戶籍的外國人。第四到第九碼為流水號。第十碼則為檢查碼。下表為國民身分證字號英文代碼與縣市及轉換對應的數值對照表。

表 1 國民身分證字號英文碼與縣市及轉換數值表

使用中的字母						
台北市	臺中市	基隆市	臺南市	高雄市	新北市	宜蘭縣
A	B	C	D	E	F	G
10	11	12	13	14	15	16
桃園市	嘉義市	新竹縣	苗栗縣	南投縣	彰化縣	新竹市
H	I	J	K	M	N	O
17	34	18	19	21	22	35
雲林縣	嘉義縣	屏東縣	花蓮縣	臺東縣	金門縣	澎湖縣
P	Q	T	U	V	W	X
23	24	27	28	29	32	30
連江縣	已停發的字母	台中縣	臺南縣	高雄縣	陽明山管理局	
Z		L	R	S	Y	
33		20	25	26	31	

國民身份證字號有其特定的編碼原則。檢查碼的產生規則如下，將第一個英文字母依照表 1 轉換為數字： 轉換後的身份證字號共 11 位、其權重從左到右分別為 (1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1)。將每位數字與其對應的權重相乘，相加的結果若是 10 的倍數則身份證字號正確。Eq. (1)是用身分證字號產生器所生成的隨機身分證字號 K114719629，呈現了符合上述的檢查碼規則。

$$\begin{cases} \overbrace{\text{K}}^{\text{轉換對應的數值=19}}\ \overbrace{1}^{\text{男性}}\ \overbrace{1471962}^{\text{流水號}}\ \overbrace{9}^{\text{檢查碼}} \Rightarrow 1\ 9\ 8\ 7 \\ \overset{6}{4}\ \overset{5}{7}\ \overset{4}{1}\ \overset{3}{9}\ \overset{2}{6}\ \overset{1}{2}\ \overset{1}{9} \Rightarrow (1\cdot 1)+(9\cdot 9)+(1\cdot 8)+(1\cdot 7)+(4\cdot 6)+(7\cdot 5)+(1\cdot 4)+(9\cdot 3)+(6\cdot 2)+(2\cdot 1) \\ +(9\cdot 1)=210 \pmod{10}=0 \Rightarrow \text{身份證字號正確} \end{cases} \quad (1)$$

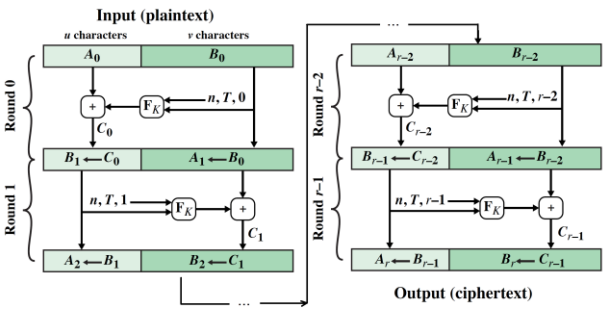
2.2 格式保留加密演算法 FPE

FPE格式保留加密顧名思義就是密文與明文有相同格式的加密技術 [7]。一般而言，FPE 需符合 (i)明文與密文具有相同的格式與長度、(ii) 適用於各種的數字與符號、(iii) 適用於多種不同的明文長度、(iv) 明文長度短依然安全。總結來說就是要符合密文與明文有相同的格式、長度，並且可以有效的解回明文。另外就是明文長度短時仍能保持安全的強度。所以FPE不直接使用區塊狀加密來進行加密，而是使用區塊加密技術來產生出隨機字串，並使用這個字串來與明文進行運算以完成加密。

FPE 採用 Feistel 結構，先將明文分為 A_0 與 B_0 兩部分。對第 i 個 Round，將 B_i 經過 $F_K(\cdot)$ 函式後產生的字串 $F_K(B_i)$ 與 A_i 做運算得到 B_{i+1} 、而 B_i 指定給 A_{i+1} ，其中 $0 \le i \le (r-1)$ ，如 Eq. (2) 所示。完成 r 個 Round (Round 0 ~ Round r) 操作後產生同樣長度的密文。圖 1 是 FPE 加密演算法的 Feistel 結構。FPE 解密只是使用反向的運算就能解回明文。

$$\begin{cases} A_{i+1} = B_i \\ B_{i+1} = A_i + F_K(B_i), \text{ 其中 } 0 \le i \le (r-1) \end{cases} \quad (2)$$

為了保持 FPE 加密的安全強度，使用了 Tweak 參數，例如我們可以明文與 Tweak 參數相加加密後再加上相同的 Tweak 參數。另外 Tweak 參數是可變的，這樣不只可以加強加密的安全性。相同的明文也因為使用不同的 Tweak 會有不同的密文。



(取材自 W. Stallings: Cryptography and Network Security: Principles and Practice)

圖 1 FPE 加密演算法的 Feistel 結構

NIST 設計了三種 FPE 加密方法: FF1、FF2 和 FF3，廣義來說他們都可以歸類成 AES (Advanced

Encryption Standard) 的加密模態。FF1 和 FF2 兩種演算法有 10 個 Round，而 FF3 只使用 8 個 Round。另外，FF2 的每個 Round 使用主密鑰所產生的子密鑰加密、而 FF1 和 FF3 的每個 Round 則直接使用主密鑰加密。還有，FF1 的每個 Round 使用 CBC 加密模態、而 FF2 和 FF3 的每個 Round 是使用簡單的 ECB 加密模態。

3. 使用格式保留 FF1 演算法的身分證字號加密方法

3.1 設計概念

我們提出的國民身分證字號格式保留加密方法是植基於 FF1。最好不要使用 FF3 技術，因為最近 FF3 已遭到密碼分析攻擊[8-10]。 NIST 已得出結論，FF3 不再適合作為通用 FPE 方法。NIST 可能會修改 SP 800-38G 變更 FF3 規範，或者是撤回 FF3 的批准。

本文是依據中華民國國民身份證字號的發放現況，以 NIST 的格式保留加密演算法 FF1 完成實作。我們的國民身分證分別有三種不同的基數，分別是代表縣市的第一碼英文字母(基數為 26)，代表性別的第二碼 (基數為 2)，流水號的第三到九碼(基數為 10)。因為他們不同的基數，所以使用 FF1 加密的時候無法一次性加密 (註: 由於 FF1 的輸入長度是 2^1 至 2^{32}、且輸入的明文需有相同的基數)。但是第一碼“英文字母”、以及第二碼“性別”各只有一個碼字，並不符合 FF1 所規範的最小長度需為 n=2。為了解決上述問題，我們將身分證字號分成兩個部分。第一部分是結合第一碼及第二碼映對至 7^2 (radix 為 7 的 2 個碼字)、第二部分是第三碼到九碼流水號 10^7 (radix 為 10 的 7 個碼字)，對於這兩部份我們分別使用兩個 FF1 來處理。另外，檢查碼不處理，加密的身份證字號檢查碼是由加密後的第一、二部分依據國民身份證字號的規則產生。

第一部分 7^2 的映對設計說明如下。因為代表陽明山管理局的字母 Y，自從 1974 年後就已經停止發放，現在陽明山管理局的現行行政區是台北市。考慮 1974 年至已有 50 年，所以我們合理地將字母 Y 去掉，這樣我們會得到基數為 25 的第一碼和基數為 2 的第二碼共有 50 種的組合。為了滿足 FF1 規範的最小長度為二，我們將第一碼及第二碼組合起來映對至 7^2 (radix 為 7 的 2 個碼字)。選擇使用小於 50 的 7^2 (49 種組合)來代表第一部份，就是我們使用基數為 7 的兩個字元來進行 FF1 的加密。此時，我們要對多出來的一個組合進行處理。台中縣、台南縣與高雄縣在 2010 年因為縣市合併的關係所以原先所使用的字母 L、R、S 也就跟著停止發放，而台中縣、台南縣、高雄縣中，又屬台南縣的人數最少，考慮到可能要做處理的數量問題，我們將代表台南縣的 R1 和 R2 做合併處理，以達到只有 49 種組合。Eq. (3)呈現了這個設計概念。

$$\begin{cases} \overbrace{\text{K}}^{\text{radix 25(除去Y)}} \quad \overbrace{1}^{\text{radix 2}} \Rightarrow \overbrace{\text{K1}}^{\text{50種組合}} \Rightarrow 7^2\text{(radix 7為基底的2個碼)} \\ \text{(註: 將R1/ R2視為一種組合、另外處理)} \\ \overbrace{1471962}^{\text{radix 10}} \Rightarrow 10^7\text{(radix 10為基底的7個碼)} \\ \text{K114719629} \Rightarrow \overbrace{\boxed{\text{K1}}}^{\text{第一部分: } 7^2} \quad \overbrace{\boxed{1471962}}^{\text{第二部分: } 10^7} \quad \overbrace{\boxed{9}}^{\text{檢查碼不處理}} \end{cases} \tag{3}$$

我們格式保留的身分證字號加密方法的身分證字號前兩碼轉換至 7^2 基底的映對如表 2 所示。舉例來說 K1 開頭的身分證字號會 (K1) 轉換為(2, 3)、而(4, 3)則代表了是 R1、或是 R2。

表 2 國民身分證字號前兩碼映對至 7^2 基底

	0	1	2	3	4	5	6
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	H1	I1	J1	K1	L1	M1	N1
3	H2	I2	J2	K2	L2	M2	N2
4	O1	P1	Q1	R1/ R2	S1	T1	U1
5	O2	P2	Q2	X2	S2	T2	U2
6	V1	W1	X1	Z1	Z2	V2	W2

R1/R2 的編碼均是 (4, 3)，接著說明如何分辨 R1/R2。假設我們今天有一個 R 開頭的身分證字號，我們依據它儲存的索引位置是奇數、或偶數來分別儲存 R1、或 R2。註: 在我們格式保留身分證字號加密方法中，此索引位置也被當作對不同身分證字號的 Tweak 參數。然後將它對應到(4, 3)後進行加密再與第二部分的加密結果結合產生一個格式相同的密文。在解回階段，我們得到(4, 3)後再根據他的索引位置得知是 R1 或 R2。在密文部份如果得到的加密結果為(4, 3)，則依據索引位置是奇 /偶數用 R1/R2 代表。我們用圖 2 身分證字號 R103764088 加密例子，來說明如何處理 R1/R。

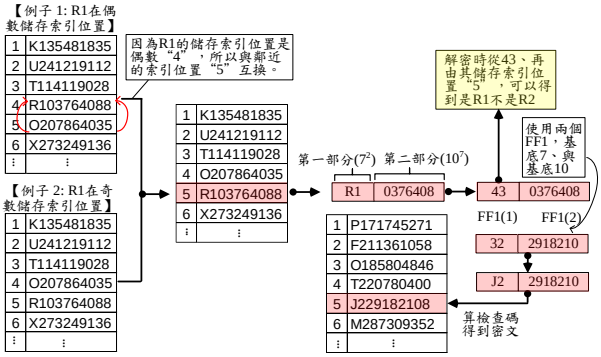


圖 2 格式保留身分證字號加密處理 R1/R2 方式

【例子 1】的 R1 是在偶數儲存索引位置，而【例子 2】的 R1 則是在奇數儲存索引位置。因為【例子 1】的 R1 儲存索引位置是偶數 “4” ，所以與鄰近的索引位置 “5” 互換。最後不正確的【例子 1】也轉換成【例子 2】的儲存方式。然後，使用兩個 FF1，分別對基底 7 的第一部分明文 $(43)_7$、與基底 10 的第二部分 $(0376408)_{10}$ 加密得到密文

$(32)_7$、與基底 10 的第二部分 $(2918210)_{10}$。以表 2 轉換(32)為(J2)，再由 J22918210 依規則計算得檢查碼 8，最後加密的身分證字號會 J229182108。解密時從 43、再由其儲存索引位置“5”，可以得到前兩碼是 R1 而不是 R2。

格式保留身分證字號加密方法中的儲存索引位置不僅用來處理、分辨R1/R2，它也是Tweak參數。這個使用方式類似AES的XTS-AES模態，利用儲存位置當Tweak參數。根據儲存索引位置的不同，就算相同的身分證字號輸入也會有不同的加密後身分證字號結果。圖3為一個簡單的例子相同的身分證字號A140228592，當分別置放在儲存索引位置“1”、和“2”。因為在FF1的Tweak參數不同，所以會產生不同的加密結果G179942203、和C223385841。

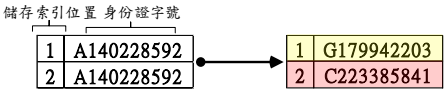


圖 3 儲存索引位置(Tweak參數)影響加密範例

3.2 FF1 演算法的身分證字號加/解密實作

描述演算法前，我們先定義一些符號。

表 3 符號定義

符號	說明
X[1…10]	原本的身分證字號
Y[1…10]	加密後的身分證字號
$\text{NUM}_{radix}(\cdot)^b$	依據定義之規則將碼字轉換成基底為 *radix* 長度為 *b* 的碼字。例如，$\text{NUM}_7(\cdot)^2$ 是表 2 身分證字號前兩碼的轉換、而 $\text{NUM}_{10}(\cdot)^7$ 表示直接擷取身分證字號第 3 碼~第 9 碼的 7 個數字
P_1 / P_2	P_1、P_2 是原身分證號轉換的第一、二部分
Q_1 / Q_2	Q_1、Q_2 是加密身分證號轉換的第一、二部分
T	儲存索引位置是 Tweak 參數用於 FF1 和 FF1^{-1}
$\text{FF1}(\cdot)$ / $\text{FF1}^{-1}(\cdot)$	$\text{FF1}(\cdot)$、$\text{FF1}^{-1}(\cdot)$是 FPE 加、解密程式
Rule(·)	計算檢查碼的函式例如 X[10]=Rule(X[1…9])

圖4是演算法 1: FF1身分證字號加密演算法。如 Eq. (3)所描述，我們先將輸入的身分證號X[1...10]的前2碼X[1, 2]轉換成第一部分$P_1=\text{NUM}_7(X[1, 2])^2$、然後第3碼 ~ 第9碼為第二部分$P_2=\text{NUM}_{10}(X[3... 9])^7$。若$P_1$是R1、或R2，則此身分證字號X[1...10]需確保儲存於奇數、或偶數索引值的儲存位置。使用FF1加密程式及Tweak 參數 T 分別加密P_1及P_2得到$Q_1= \text{FF1}(P_1, T)$、和$Q_2= \text{FF1}(P_2, T)$。FF1是 Feistel結構，對P_1而言會將基底7長度n=2的字串分成左、右各為u=1、v=1的長度處理。對P_2則是將基底10長度n=7的字串分成左、右各為u=3、v=4的長度處理。

依據表2的映對，由Q_1得到加密前兩碼Y[1, 2]。若是Q_1="43"則Y[1, 2]= R1(當T是奇數)或Y[1, 2]= R2 (當T是偶數)。雖然這加密的Y[1, 2]設為是R1、或R2都不會影響解密(註: 依照R1、R2儲存於奇數、或偶數索引值的儲存位置，這樣才不會洩漏出這個檔案是有加密或無加密)。另外，由Q_2得Y[3…9]以檢查碼函式計算檢查碼 Y[10] =Rule(Y[1...9])。最後將加密後的身分證字號Y[1...10] 儲存於索引值為 T 的儲存位置。

【演算法1】: FF1身分證字號加密演算法

輸入：身分證字號X[1...10]、Tweak參數 T
/* 參數 T 是儲存索引位置 */
輸出：加密後的身分證字號Y[1...10]
/* Y[1...10]需符合台灣身分證字號規則 */
步驟：
(1-1) if ((X[1, 2]=="R1") and (T is even)) then
{move X[1...10] to the position with an odd index T'; $T \leftarrow T'$};
/* 將此身分證字號X[1...10]移至有奇數索引值的儲存位置 */
(1-2) if ((X[1, 2]=="R2") and (T is odd)) then
{move X[1...10] to the position with an even index T'; $T \leftarrow T'$};
/* 將此身分證字號X[1...10]移至有偶數索引值的儲存位置 */
(2) $P_1 \leftarrow \text{NUM}_7(X[1, 2])^2$;
/* 依據表2的映對表 */
(3) $P_2 \leftarrow \text{NUM}_{10}(X[3... 9])^7$;
(4-1) $Q_1 \leftarrow \text{FF1}(P_1, T)$;
/* 基底 7、長度 2、Tweak 參數 T 的FF1加密程式;
(4-2) if ((Q_1=="43") and (T is odd)) then Y[1, 2] ←"R1";
go to step **(5)**
(4-3) if ((Q_1=="43") and (T is even)) then Y[1, 2] ←"R2";
go to step **(5)**
(4-4) $Y[1, 2] \leftarrow Q_1$; /* $Y[1, 2] \leftarrow Q_1$ 是依據表2的映對表 */
(5) $Q_2 \leftarrow \text{FF1}(P_2, T)$; $Y[3...9] \leftarrow Q_2$;
/* 基底 10、長度 7、Tweak 參數 T 的FF1加密程式 */
(6) Y[10]←Rule(Y[1...9]); /* 計算檢查碼 */
(7) 儲存Y[1...10]於索引值為 T 的儲存位置;

圖 4 加密演算法

圖5是演算法 2: FF1身分證字號解密演算法，過程與加密演算法基本類似。

【演算法2】: FF1身分證字號解密演算法

輸入：加密後的Y[1...10]、Tweak參數 T
/* 參數 t 是儲存索引位置 */
輸出：解密後的身分證字號X[1...10]
步驟：
(1) $Q_1 \leftarrow \text{NUM}_7(Y[1, 2])^2$;
/* 依據表2的映對表 */
(2) $Q_2 \leftarrow \text{NUM}_{10}(Y[3... 9])^7$;
(3-1) $P_1 \leftarrow \text{FF1}^{-1}(Q_1, T)$;
/* 基底 7、長度 2、Tweak 參數 T 的FF1解密程式 */
(3-2) if (P_1=="43") and (T is odd)) then X[1, 2] ←"R1";
go to step **(4)**
(3-3) if (P_1=="43") and (T is even)) then X[1, 2] ←"R2";
go to step **(4)**
(3-4) $X[1, 2] \leftarrow P_1$; /* 依據表2的映對表 */
(4) $P_2 \leftarrow \text{FF1}^{-1}(Q_2, T)$; $X[3...9] \leftarrow P_2$;
/* 基底 10、長度 7、Tweak 參數 T 的FF1解密程式 */
(5) X[10]←Rule(X[1...9]); /* 計算檢查碼 */
(6) 還原X[1...10]於索引值為 T 的儲存位置;

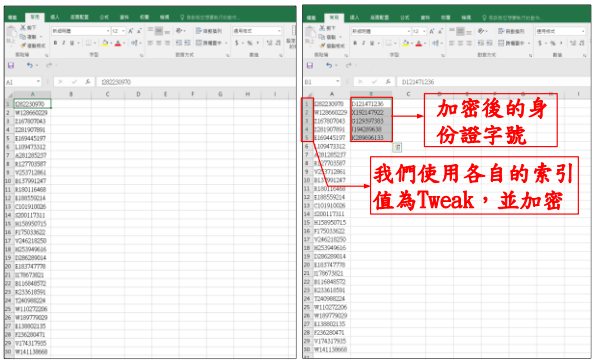
圖 5 解密演算法

輸入加密 Y[1...10]及 Tweak 參數 T，轉換得 $Q_1= \text{NUM}_7(Y[1, 2])^2$ 和 $Q_2= \text{NUM}_{10}(Y[3... 9])^7$。用 FF1^{-1} 解密程式解回 P_1、P_2。依據表 2 的映對表由 P_1得回 X[1, 2] 。若是 P_1="43"，則由 Tweak 參數 T 決定 X[1,

2] 是 R1 或 R2。由 P_2 可以得到 X[3...9]、並計算檢查碼 X[10]=Rule(X[1...9])，還原儲存位置的原身分證字號 X[1...10]。

4. 實作結果

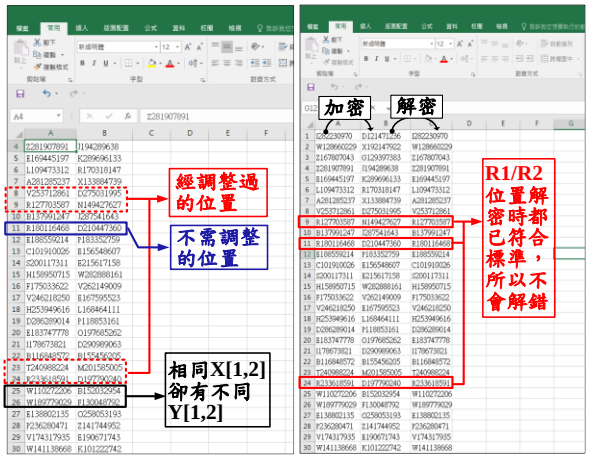
本次實驗的資料庫我們使用 Excel，其中 Tweak 參數值就使用 Excel 儲存格的索引值。測試資料如圖 6(a)所示，有 30 筆身分證資料。圖 6(b)是以各自儲存位置的索引值當 Tweak 參數的前 5 筆加密後的身分證字號。以第一筆資料 I282230970 為例，我們將兩碼 I2 根據表 2 轉換為 "31"，然後分別對前兩碼 (31) 和後 7 碼 (8223097)，使用 FF1 加密得到 (03) 和 (2147123)，將(03)映對得到 D1 後，就可以獲得前九碼 Y[1…9]=D12147123，通過公式可以得到檢查碼 Y[10]= 6，最後加密的身分證字號是 Y[1…10]=D121471236。



(a) 30 筆測試資料　(b) 前 5 筆加密後的身分證字號

圖 6 測試資料與部份加密結果

我們可以發現在測試資料中，R 開頭的特例還有其他三處，在索引儲存位置 “8” 是一組 R1 開頭的身分證字號 R127703587。由於索引值 8 是偶數，我們將這筆資料與下一筆資料 V253712861 交換位置。所以身分證字號 R127703587 的索引位置變為 9。一個不需調整的例子是身分證字號 R180116468 在奇數索引位置 “11”。另外，身分證字號 R233618591 是在奇數索引位置 “23”，它也需要與下一筆資料 T240988224 更換儲存位置，最後 R23361859 存於位置 “24”。除此之外，儲存位置 “25”和“26”的身分證字號是 W110272206 和 W189779029，他們的 X[1, 2]都是“W1”。由於 Tweak 參數值不同，加密後的身分證字號分別是 B152032954、F130048792。他們的 Y[1, 2] 分別是 “B1”和“F1”。最後我們可以得到所有 30 筆測試資料的加密結果。圖 7(a)呈現了 R1/R2 需調整、或不需調整的狀況，以及雖然有相同的明文開頭 W1 卻因為使用不同 Tweak 參數有不同的加密結果。解密階段是對個別欄位的資料進行解密，和加密階段類似先根據表 2 將前面兩碼對應到兩個基數為 7 的值，在與後面 7 碼分別進行解密。最後計算檢查碼再組合，就可解回原本的身分證字號。因為在加密時，我們已經對前兩碼 R1/R2 進行處理 (確保 R1 在奇數索引位置、R2 在偶數索引位置)，所以可確保解密的正確性。圖 7(b)展示了 30 筆加、密前後的身分證字號。



(a) R1/R2 處理及 Tweak 參數效果　(b) 30 筆加/解密前後的身分證字號

圖 7 所有測試資料加密結果

5. 結論

我們以 NIST 的 FF1 完成了格式保留的身份證字號加密，加密過程是依據目前中華民國國民身份證字號的發放現況而設計。對於 FPE 加密，秘密數據域值的大小很重要。所謂的域值大小，就是可能的輸入字串的數量。小的秘密數據域值當然會對 FPE 加密方法有安全威脅，例如參考文獻[8]就是對較小域值的 FPE 加密法攻擊。信用卡 16 個數字的 FPE 域值約 $10^{16}\approx2^{53}$、而 4 個數字 PIN 號碼的 FPE 域值約 $10^{4}\approx2^{14}$。本文實做的國民身分證字號第一部份的域值約為 $(25\times2)\approx2^{5.7}$、第二部份的域值為 $10^{7}\approx2^{23.3}$。2019 年 NIST 公布了 SP 800-38G 的修訂版 [11] 建議 FF1 加密的最小域值需為 100 萬。為了提高格式保留身份證字號加密的安全性，未來我們的工作著重於提高域值的研究。

致謝

本研究部分成果由國科會計畫編號 NSTC 112-2221-E-259-007-MY2 補助，特此致謝。

附錄

我們在附錄中，包含了實作程式碼及更詳細的實驗模擬。其中的 FF1 中用到的 AES-CBC 模態，是以 PyCryptodome 程式庫完成。圖 8 是實作 FF1 完成格式保留的身份證字號加密程式碼。圖 9 則是更詳細的實驗模擬，Tweak 參數值是由 1013~1053、並標示了更動的 R1/R2、P_1、P_2、Q_1、Q_2、Y[1, 2]、及 Y[10]等相關細項。

```
import math
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Random import get_random_bytes
key = get_random_bytes(16) # 隨機產生 16 bytes 長度的密鑰
def CIPH(key, X):
    """
    inputs:
        key 的資料型態須為 bytes, 其長度須為 16-bytes.
         X  的資料型態須為 bytes, 其長度須為 16-bytes.
    """
    cipher = AES.new(key, AES.MODE_CBC) #使用 PyCtyptodome 程式庫實作
    #選擇 Cipher-block chaining (CBC)模態
    iv = cipher.iv # 初始向量(initialization vector, IV)
    ciphertext = cipher.encrypt(X) # 加密 X
    return (iv, ciphertext)
def PRF(Key, X):
    """
    inputs:
        Key 的資料型態須為 bytes, 其長度須為 16-bytes.
         X  的資料型態須為 bytes, 其長度須為 16 的倍數-bytes.
    """
    m = int(len(X) / 16) #判斷 X 可分為幾個 block, 1 個 block 為 16-bytes
    zero = 0
    Y_0 = zero.to_bytes(16, 'big') # Y_0 為長度 16-bytes 的零
    X_list = []
    for i in range(m):
        X_list.append(X[0+16*i : 16*(i+1)])
    # X_list[0] 為第一個 block; X_list[1] 為第二個 block、共有 m 個 block
    for i in range(m):
        if(i==0):
            Y_0 = int.from_bytes(Y_0, byteorder='big') # bytes -> decimal
            X = int.from_bytes(X_list[i], byteorder='big') # bytes -> decimal
            data = Y_0 ^ X # 進行 xor 運算
            data = data.to_bytes(16, 'big') # decimal -> bytes
            Y = CIPH(Key, data) # CIPF(.) 會回傳 (iv, ciphertext)
        Y = int.from_bytes(Y[1], byteorder='big') # bytes -> decimal
        X = int.from_bytes(X_list[i], byteorder='big') # bytes -> decimal
        data = Y ^ X # 前一個密文塊進行 xor 運算
        data = data.to_bytes(16, 'big') # decimal -> bytes
        Y = CIPH(Key, data)
    return Y # (iv, ciphertext)
def FF1(plaintext, radix, key, tweak):
    """
    inputs:
        plaintext 的資料型態須為字串.
        radix 的資料型態須為整數.
        key 的資料型態須為 bytes, 其長度須為 16-bytes.
        tweak 的資料型態須為 bytes.
    """
    n = len(plaintext) # n 為 plaintext 字串 有多少 字元
    u = math.floor(n / 2) # math.floor(x) 會回傳小於等於 x 的最大整數
    v = n - u
    A, B = plaintext[:u], plaintext[u:]
    b = math.ceil(math.ceil(v*math.log2(radix)) / 8) # math.ceil(x)回傳≤ x 最大整數
    d = 4 * math.ceil(b/4) + 4
    tweak = tweak.to_bytes(2, 'big')
    t = len(tweak)
    # || 表示 concatenation
    # P = [1]^1 || [2]^1 || [1]^1 || [radix]^3 || [10]^1 || [u mod 256]^1 || [n]^4 || [t]^4
    P = bytes([1,2,1]) + radix.to_bytes(3, 'big') + bytes([10, u % 256]) +
        n.to_bytes(4, 'big') + t.to_bytes(4, 'big')
    for i in range(10):
        # Q = tweak || [0]^(-t-b-1) mod 16 || [i]^1 || [NUM_radix(B)]^b
        Q = tweak + (0).to_bytes((-t - b - 1) % 16, 'big') + i.to_bytes(1, 'big') +
        NUM_radix(int(B), radix).to_bytes(b, 'big')
        R = PRF(key, P + Q) # P+Q = P || Q
        y = int.from_bytes(R[1], 'big') # bytes -> decimal
        y = NUM_radix(y, 2)
        if i % 2 == 0:
            m = u
        else:
            m = v
        c = (NUM_radix(int(A), radix) + y) % radix**m
        C = str_radix_m(c, m, radix) # 將整數 c 轉為 字串, 其長度為 m 個字元
        A = B
        B = C
    return str(A) + str(B)
def NUM_radix(n, radix):
    a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'A', 'B', 'C', 'D', 'E', 'F']
    b = []
    while True:
        s = n // radix
        y = n % radix
        b = b + [y]
        if s == 0: break
        n = s
    b.reverse()
    string = ""
    for i in b:   #     print(a[i], end='')
        string += str(a[i])
    return int(string)
def str_radix_m(x, m, radix):
    X = [0] * m
    for i in range(m - 1, -1, -1):
        X[i] = x % radix
        x = x // radix
    tmp = ""
    for _ in range(len(X)):
        tmp += str(X[_])
    return tmp
```

圖 8 實作 FF1 加密身份證字號程式碼

	T	X[1...10]	modified R1/R2	alp2num	P1	P2	Q1	Q2	num2alp	Y[10]	Y[1...10]
1013	S212937770	S212937770	541293777	54	1293777	23	3526008	K1	7	K135260087
1014	A179507404	A179507404	007950740	00	7950740	32	0183943	J2	5	J201839435
1015	E141993252	E141993252	044199325	04	4199325	42	6331647	Q1	2	Q163316472
1016	X198224231	X198224231	629822423	62	9822423	42	3135536	Q1	5	Q131355365
1017	Q218683644	Q218683644	521868364	52	1868364	05	3209578	F1	0	F132095780
1018	C154574885	C154574885	025457488	02	5457488	26	7689730	N1	4	N176897304
1019	B249370590	B249370590	114937059	11	4937059	35	6268170	M2	0	M262681700
1020	I199419452	I199419452	219941945	21	9941945	20	2364368	H1	1	H123643681
1021	G278204215	G278204215	167820421	16	7820421	16	0251544	G2	1	G202515441
1022	K157158764	K157158764	235715876	23	5715876	30	7949097	H2	6	H279490976
1023	P132640068	P132640068	413264006	41	3264006	20	6508218	H1	8	H165082188
1024	Q107810955	Q107810955	420781095	42	0781095	24	2112306	L1	2	L121123062
1025	T296462760	T296462760	559646276	55	9646276	26	3168577	N1	7	N131685777
1026	U221717199	U221717199	562171719	56	2171719	52	5517041	Q2	9	Q255170419
1027	W148021924	W148021924	614802192	61	4802192	53	8344233	X2	6	X283442336
1028	D203889947	D203889947	130388994	13	0388994	46	2481328	U1	5	U124813285
1029	E218396521	E218396521	141839652	14	1839652	64	3060886	Z2	7	Z230608867
1030	N293728088	N293728088	369372808	36	9372808	02	2516930	C1	7	C125169307
1031	G175140981	G175140981	067514098	06	7514098	25	9957321	M1	7	M199573217
1032	A205802392	A205802392	100580239	10	0580239	13	1621792	D2	8	D216217928
1033	G224296856	G224296856	162429685	16	2429685	65	2863008	V2	9	V228630089
1034	S209913033	S209913033	540991303	54	0991303	16	4103536	G2	6	G241035366
1035	R180751681	R180751681	438075168	43	8075168	06	9298692	G1	7	G192986927
1036	V104586395	V104586395	600458639	60	0458639	05	1780973	F1	3	F117809733
1037	R248748275	T148636270	454863627	45	4863627	30	4985959	H2	2	H249859592
1038	T148636270	R248748275	434874827	43	4874827	54	8007160	S2	9	S280071609
1039	T136116469	T136116469	453611646	45	3611646	44	8022767	S1	2	S180227672
1040	M149728735	M149728735	254972873	25	4972873	06	7297317	G1	5	G172973175
1041	V174317935	V174317935	607431793	60	7431793	44	1672924	S1	5	S116729245
1042	M285307510	M285307510	358530751	35	8530751	03	9873183	D1	4	D198731834
1043	E231078320	E231078320	143107832	14	3107832	30	5321165	H2	3	H253211653
1044	M171349688	M171349688	257134968	25	7134968	03	9368399	D1	5	D193683995
1045	F251265893	F251265893	155126589	15	5126589	52	7468832	Q2	9	Q274688329
1046	A277246580	A277246580	107724658	10	7724658	34	1965891	L2	8	L219658918
1047	M275765902	M275765902	357576590	35	7576590	06	9799810	G1	5	G197998105
1048	D233570326	D233570326	133357032	13	3357032	06	7670154	G1	0	G176701540
1049	O184398027	O184398027	408439802	40	8439802	43	3632133	R1	3	R136321333
1050	N282916750	N282916750	368291675	36	8291675	40	0614829	O1	0	O106148290
1051	D147823392	D147823392	034782339	03	4782339	41	8113672	P1	0	P181136720
1052	K111788882	K111788882	231178888	23	1178888	53	5501312	X2	9	X255013129
1053	Q288255570	Q288255570	528825557	52	8825557	00	1236880	A1	3	A112368803

圖 9 更詳細的實驗模擬

參考文獻

[1] B. Cui et al., "A Data Masking Scheme for Sensitive Big Data based on Format-Preserving Encryption," IEEE ICSE and IEEE EUC, pp. 518-524, 2017.

[2] J. Zou et al., "Improved Prefix Based Format-Preserving Encryption for Chinese Names," *China Comm.*, pp. 78-90, 2018.

[3] S. Gupta, "Ensuring Data Security in Databases Using Format Preserving Encrypt ion," Confluence-2018, pp. 214-217, 2018.

[4] D. Prakash, "FPRC5: Improving the Data Security using Format PreservedRC5 Block Cipher Algorithm in Credit Card Application," ICCES 2021, pp. 775-785, 2021.

[5] C. Lee et al., "Novel Encryption Method of GPS Information in Image File Using Format-Preserving Encryption," AISC, vol. 994, Springer, Cham, 2020.

[6] N. Chaudhari, "A Cloud Security Approach for Data at Rest Using FPE," IJCCSA 2015, vol. 5, pp. 11-16, 2015.

[7] W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson Education, 2020.

[8] F.B. Durak and S. Vaudenay, "Breaking the FF3 Format-Preserving Encryption Standard over Small Domains," https://eprint.iacr.org/2017/521, 2017.

[9] V.T. Hoang et al., "Attacks Only Get Better: How to Break FF3 on Large Domains," https://eprint.iacr.org/2019/244, 2019.

[10] O. Amon et al.' "Three Third Generation Attacks on the Format Preserving Encryption Scheme FF3," EUROCRYPT 2021, LNCS vol. 12697, 2021.

[11] M. Dworkin, "NIST SP 800-38G Rev. 1, Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption," doi.org/10.6028/NIST.SP.800-38Gr1-draft, 2019.