

一次搞懂密碼學中的三兄弟 — Encode、Encrypt 跟 Hash



Larry Lu

Mar 23 · 10 min read

前言

隨著資訊安全越來越受到使用者的重視，密碼學作為資訊安全的基石也逐漸變成開發人員的必修課，所以筆者我今天就要來說說密碼學中大家很容易搞混的三個東西：編碼（Encode）、加密（Encrypt）跟雜湊（Hash）

雖然他們三者的比較已經很多人寫過了，但還是有些人搞不清楚，所以這篇決定換個方式：不提太多理論，而是舉大量的例子跟實際應用，如果這些例子你都能看懂，那自然就會知道三者的差別

編碼（Encoding）

首先來看看最簡單的編碼，所謂的編碼並不會修改資料、也沒有任何加密的效果，單純就是換個方式來表達資料而已，其中最有名的例子就是摩斯密碼

A ● —
 B — ● ● ●
 C — ● — ●
 D — ● ●
 E ●
 F ● ● — ●
 G — — ●
 H ● ● ● ●
 I ● ●

J ● — — —
 K — ● —
 L ● — ● ●
 M — —
 N — ●
 O — — —
 P ● — — ●
 Q — — ● —
 R ● — ●

S ● ● ●
 T —
 U ● ● —
 V ● ● ● —
 W ● — —
 X — ● ● —
 Y — ● — —
 Z — — ● ●

在摩斯密碼的編碼系統中，每個英文字母都可以被表示成點（dot）跟劃（dash）的組合，或是表示成長音跟短音，譬如說將 Hello World 編碼（encode）成摩斯密碼的形式就是 `.... . .-.. .-.. --- .-- --- .- .-.. ---`

雖然這串東西乍看之下很難懂，但 只要知道摩斯密碼的轉換規則 就有辦法翻譯回來，所以編碼 完全沒有安全性可言，就只是 換個方式來表達資料 而已

實例

1. encodeURIComponent and decodeURIComponent

在 JavaScript 中有兩個很實用的 function 分別是 `encodeURIComponent` 跟 `decodeURIComponent`，他們是用來把網址中的特殊字元（空白、標點符號等等）編碼成符合 URL 的格式（轉完變超醜的）

譬如說 `https://www.google.com/search?q=創世神` 這段網址會被編碼成

`https://www.google.com/search?q=%E5%89%B5%E4%B8%96%E7%A5%9E`，所以即便電腦不支援顯示中文，或是某些比較舊的瀏覽器無法貼中文網址，也還是可以透過編碼後的網址連到目標網站

2. Base64

Base64 是一種可以把二進位的資料編碼成 ASCII 字元的方法，如果你想在純文字的介面上傳送圖片給朋友，可以先用指令 `cat hello.png | base64` 把圖片編碼成

Base64 格式，然後把生出來的這一大串東西傳給他

```
Larry-Pro ~/Desktop > cat Hello.png | base64
iVBORw0KGgoAAAANSUhEUgAAABoAAAANCAYAAAC3mX7tAAABwk1EQVQ4jd3UP2sqQRSG8Xw1SS5EoIY1aqGWqWaiJCIuNha6wo
EgQBAAtTxMJCFMU/hViJhTaCILKNhSbaaVhclmXGfVJc8N6QSKxS3NP00fNj3hnmi1+qq/8TchwH27axbZvj8QiAEALbthFCfD
s4nU7J5XJIKRk0hxSLxZ+h/X5PLBYjEonw+voKQKfTwe120+v1vh18f3/n5uYGIQRvb2/4fL6fIYBKpUKhUDgtWJaF1+vFNM1
Pm9u2DYCUktvbW4QQGIaB3+8/9Tm0w26306XzBXp6ekJKiZSSw+GAX+PBNE0sy0JVVRqNBtFo1P18fhZar9c8Pj5Sr9eJx+PM
ZrOv0N3dHeVymXK5TK1U4vr6GtM0qVarpNNpJpMJxWKRVCp1F1IUhcFgAMByuSQUCuE4zmXRZTIZX1Se0HUdXdfZbrdnoVAox
GKxA0Bw00ByuZBSXgY1m02SySS04wBgGMZZSNM0arUa80d13t/f/41ut9uhKAoPDw+sVisAWq0WgUCAdruNEIJsNksikSCfzz
MajRiNRgSDQfr9Pt1ul3A4zHg8xjAMNE3j+fkZVVXZbDaf7+iSk1KeTnVJ77/1az/DB4jtxbidrgwVAAAAAE1FTkSuQmCC
```

你的朋友收到 這團看起來像是亂碼的東西 後，只要用 Base64 進行 decode 就可以收到原本的圖片了：`echo {那一大串} | base64 -D > image.png`



為了讓編碼後的那一大串能顯示在一個螢幕之內，我用的圖片畫質很差，請不要介意XD

雖然這團東西看起來像亂碼，但他是沒有任何安全性的，熟悉 Base64 的人很快就能解碼回來，頂多只能騙騙你那不懂電腦的老媽XD

3. 霍夫曼編碼

霍夫曼編碼（Huffman Coding）是一種用來進行 **無失真壓縮** 的編碼演算法，說穿了牠的概念就是把常用的字記成縮寫，從而降低資料量、達到壓縮的效果

如果還是覺得有點抽象的話，可以想想你去早餐店跟阿姨點餐時：

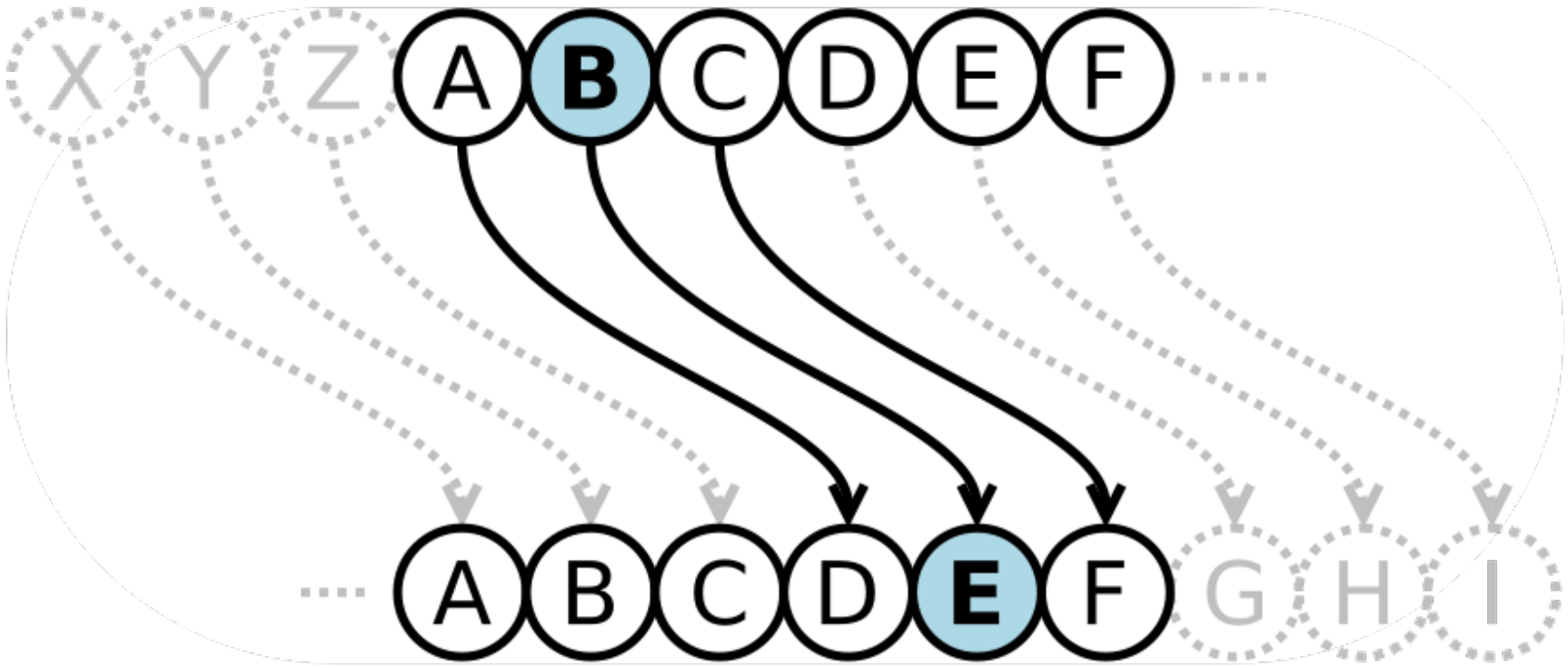
我要“一個雞腿堡加蛋”跟“大杯的冰紅茶”

因為阿姨腦中有一個早餐縮寫的對照表，所以在你講的同時阿姨就會快速在紙上記下“G堡蛋、大冰紅”，這就是壓縮技術的原理：把常出現的字用縮寫記錄下來，等日後要解壓縮時再還原回去

. . .

加密（Encrypt）

加密跟剛說到的編碼有點像，唯一不同之處是加密跟解密必須要有金鑰（Key）才能進行。以最簡單的 凱薩加密法 來說，他加密的方式就是把每個英文字母加上一個 偏移量，這個偏移量就是用來執行加解密的 Key



假如我想要用凱薩加密法對 `Hello World` 進行加密，並且設定 `key=3`，那加密完的結果就會變成 `Khoor Zruog`，只有知道 `key` 的人才有辦法把 `Khoor Zruog` 還原回去 `Hello World`

說是這麼說，但你我都知道英文字母不就那 26 個嗎？照目前電腦的運算速度，只要把偏移量從 0 到 25 都算過一遍，即便我不知道 `key` 也可以很容易就看出偏移量是 3

```
Larry-Pro ~/Desktop > python caesar.py "Khoor Zruog"
00: Khoor Zruog      01: Jgnnq Yqtnf      02: Ifmmp Xpsme      03: Hello World
04: Gdckn Vnqkc      05: Fcjjm Umpjb      06: Ebiil Tloia      07: Dahhk Sknhz
08: Czggj Rjmgj      09: Byffi Qilfx      10: Axeeh Phkew      11: Zwddg Ogjdv
12: Yvccf Nficu       13: Xubbe Mehbt      14: Wtaad Ldgas      15: Vszzc Kcfzr
```

16: Uryyb Jbeyq	17: Tqxxa Iadxp	18: Spwwz Hzcwo	19: Rovvy Gybvn
20: Qnuux Fxaum	21: Pmttw Ewztl	22: Olssv Dvysk	23: Nkrru Cuxrj
24: Mjqqt Btwqi	25: Lipps Asvph		

所以說凱薩加密法非常不安全，他最大的用處應該就是在課堂上傳紙條，但又不
想被中間的同學看懂（我就幹過這種事XD），除此之外在現代已經沒什麼用處了

實例

1. AES

AES (Advanced Encryption Standard) 是一種對稱加密演算法，所謂的對稱就是說
加密解密 **都是用同一個 key**，這點跟上面說到的凱薩加密法一樣，但 AES 不像凱
薩的 key 只有 0-25 這麼少種，而是可以有超過 10^{38} 種，所以安全性比凱薩高非
常非常多

除了安全性高不易被破解之外，AES 加密檔案的速度也非常快，所以被美國政府
用來加密機密檔案。如果你也想用 AES 來加密 D 槽裡面那些神秘檔案，可以試試
系統內建的 OpenSSL

加密：`openssl aes-256-cbc -in <input> -out <output>`，輸入密碼（Key）加密
後會變成一團醜不拉噁的亂碼，完全不知道從何破解起，不像凱薩加密完還是英
文字母，一下子就猜到了


```
Larry-Pro ~/Desktop > cat plain_text
```

	File: plain_text
1	Hello, I'm Larry.

```
Larry-Pro ~/Desktop > openssl aes-256-cbc -in plain_text -out encrypted_text
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
Larry-Pro ~/Desktop > cat encrypted_text
```

	File: encrypted_text
1	Salted__I^0^M2W?^Y%?e,Pc??t^_??W??o?^N~?^Tl[???

要解密時就同樣的指令加個 `-d`，輕輕鬆鬆



但如果你不小心忘記密碼，那就真的 gg 了，因為 AES 加密後的檔案即使用超級電腦來破解，也要超過十億年，更不用說用你的電腦了XD

對稱式加密法的缺點

AES 又快又安全，聽起來超厲害對吧？但他這類 **對稱式加密法** 有一個缺點：就是如果你想把加密後的 文字/檔案 傳給你的秘密情人 Alice，兩個人就**必須先講好密碼是什麼**，這樣你傳過去她才用你設的密碼解密

但因為網路環境是不安全的（尤其是在有 HTTPS 之前），所以除非你把 Alice 當面約出來講密碼是什麼，不然只要是透過網路把密碼傳給對方，就有機會被中間人拿走

一旦中間人拿到密碼，那你用再強的加密演算法都沒有用。而且比起完全沒加密，更恐怖的是明明密碼已經被偷走了，你還自以為檔案有加密就超安全，真的怎麼死的都不知道

2. RSA

為了解決上述的問題，於是就有了**非對稱加密法**，其中最有名的就是 RSA

RSA 這類非對稱加密法有個很特別的地方，就是他會產生一組兩個 Key 分別叫公鑰（Public Key）跟私鑰（Private Key），而且 **用公鑰加密的內容只能用私鑰解**（超神奇的！！！）

同樣的情況當你要傳檔案給 Alice，就先請 Alice 生一組 Key 然後把公鑰傳給你。你有了 Alice 產生的公鑰之後，就用公鑰幫檔案加密再傳給她，Alice 收到就可以用她自己手上的私鑰解開（這段比較難，可以多看不幾遍）

安全性方面，因為私鑰從頭到尾都在 Alice 手上，完全沒有傳出去過，所以即便中間人取得公鑰加密後的檔案也沒辦法解開，超安全 der

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

雜湊（Hashing）

接著是最後一個：雜湊，不知道大家有沒有注意過你的身分證字號，他的最後一碼其實是個驗證碼哦，他是前九碼根據 **某個公式** 計算出來的

以 M140051653 這個身分證字號來說，計算的方式是先把 M 轉換成對應的數字 21，接著根據下圖的方式進行計算，如果算出來剛好等於最後一碼，那代表這個身分證字號是合法的

M(2 1)140051653

x1987654321

10 - (2+9 + 8+28+0 + 0+20+3 + 12+5) % 10 = 3

這種把各個 欄位/字元 丟進去某個公式計算的方式就叫做**雜湊（Hash）**，而這個計算公式就稱為 **雜湊函數（Hash function）**，過程可能會做各種加減乘除，最後算出一個值或字串

因為最後一個數字 3 是經由前幾個數字計算、濃縮出來的，所以理所當然不可能由雜湊後的結果 3 回推出前幾個數字分別是什麼，所以雜湊的過程是 **不可逆的**

實例

1. 判斷檔案內容是否相同

如果你想驗證某兩個檔案的內容是否相同，可以使用 md5 來計算檔案的雜湊值，他會使用某個 hash function 根據檔案內容 計算出一個長度 128 bit 的雜湊值，若是兩個檔案內容不同就會計算出不同的 hash value

其實有很低的機率 $(1/2^{128})$ 會產生碰撞，造成不同檔案有相同的 hash value，但

因為機率太小一般情況下可以忽略

```
Larry-Pro ~/Desktop > md5 data1 data2 data3
MD5 (data1) = ba1f2511fc30423bdbb183fe33f3dd0f
MD5 (data2) = ba1f2511fc30423bdbb183fe33f3dd0f
MD5 (data3) = e7df7cd2ca07f4f1ab415d457a6e1c13
Larry-Pro ~/Desktop > cat data1 data2 data3
```

	File: data1
1	123
	File: data2
1	123
	File: data3
1	1234

如上圖，從雜湊值可以看出 data1 跟 data2 的雜湊值是相同的，也就代表他們的檔案內容有很高機率一樣，而 data2 跟 data3 的雜湊值不同，代表他們的內容一定不一樣

2. 驗證檔案是否被竄改

如果你開發過前端的話應該會發現，在你要使用 bootstrap 或 jquery 這類第三方 library 時，官方都會提供一個 integrity 屬性，那個就是 library 程式碼經過 SHA-2 演算法 hash 的結果

這樣子做有個好處，因為使用 bootstrap 的網站太多了，如果有天很不幸官方提供的 bootstrap.js 網址被駭客偷換成惡意程式碼，就會有很多網站遭殃

但有了 hash 之後，因為不同的程式碼會導致不同的 hash value，所以瀏覽器看到就不會載入怪怪的 bootstrap.js，只因為他的 hash 結果跟原本的不一樣

3. 驗證使用者密碼

大部分網站的後端資料庫都不會直接儲存使用者的密碼，因為那樣太危險了，取而代之的是儲存密碼的雜湊值

當使用者嘗試要登入時，只要將使用者輸入的密碼進行雜湊，再跟資料庫裡面的雜湊值比對就可以了。只要輸入的密碼是正確的，用同一個公式進行計算必定會得到相同的結果

安全性方面，因為雜湊值無法反推回原密碼，所以即便有一天資料庫不小心洩漏出去了，駭客還是拿不到使用者的密碼，只能拿到密碼的雜湊值

• • •

重點整理

看到這相信大家已經累了，怕大家看到後面就忘了前面，所以再條列一下他們三個的重點

編碼（Encoding）

- 只是換個方式表達資料
- 不需要 Key 即可解碼（不安全）

加密（Encrypt）

- 用 Key 來保護資料的機密性
- 加密跟解密都需要 Key

雜湊（Hashing）

- 把資料丟進一串公式計算出一個結果
- 無法反推回原字串

總結

相信大家都了解編碼、加密跟雜湊的不同了，因為他們三者各有優缺，適用的情境也不太一樣，所以實務上常常各取所長把他們混在一起用

譬如說壓縮檔加密就同時用到編碼跟加密、JWT (JSON Web Token) 用到編碼跟雜湊、HTTPS 的實現則是用到加密跟雜湊...

像這樣的例子還有很多，他們三者相輔相成才有了現在流行的各種認證機制，很多聽起來很神奇、很安全的功能其實背後都是他們在撐，所以說他們是網路安全的基石也不為過吧！

延伸閱讀

- [Hash是什麼？5分鐘帶你了解區塊鏈雜湊相關的知識](#)
- [What is SSH?](#)
- [When to base64 encode images \(and when not to\)](#)

Cryptography

Security

Medium

About Help Legal

Get the Medium app

