



# 加密鑰匙的略解

密碼學裡的鑰匙對，加密和解密演算法



Philip Kwan

Jan 25

ooo



我在前文：

## BIP-39種子和它的安全性

解說了加密貨幣錢包和BIP-39的詞語組合和種子，也用密碼學概略地解說了加密鑰匙。

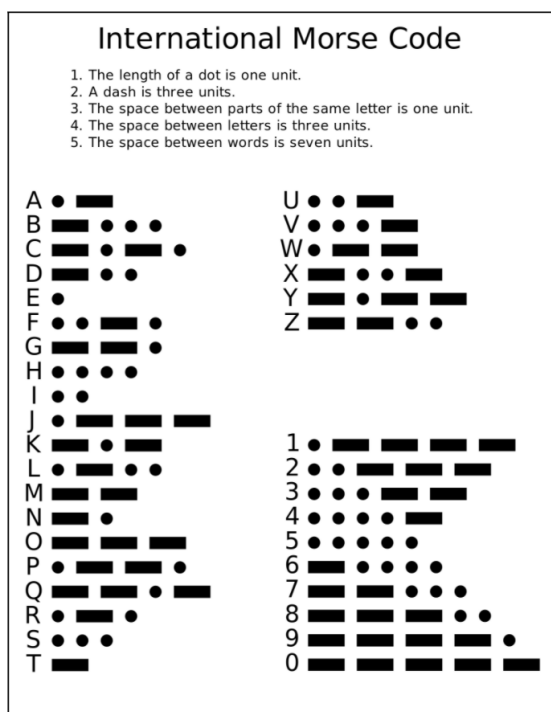
這篇文章我想再多解說一些密碼學的原理，因為我覺得了解密碼學對於了解區塊鏈，了解加密貨幣，了解互聯網，皆有很好的幫助。

這篇文章會分成三部分，第一部分比較容易了解，之後的部分會有一些技術性的解釋，可能會較難一點。

- 1) 加密鑰匙對，私鑰和公鑰的運作原理
- 2) 用符號和程式來表達加密和解密等運算
- 3) 再解說種子與加密鑰匙對的關係

## 加密鑰匙對，私鑰和公鑰的運作原理

大家可能對普通的加密和解密有基本的理解，例如摩斯密碼。



摩斯密碼是對稱密鑰加密 (symmetric key algorithm)，即加密和解密也是用同一個方法，或鑰匙 (key)。

與之相對的，非對稱密鑰加密 (asymmetric key algorithm)，或公開金鑰加密 (public key cryptography)，是利用一對鑰匙，即私鑰 (private key) 和公鑰 (public key) 來進行加密和解密。

[Wikipedia | Public Key Cryptography](#)

讓我用一些現實的例子做比喻。

我擁有一對這樣的鎖。應該說，我擁有一批這樣的鎖：

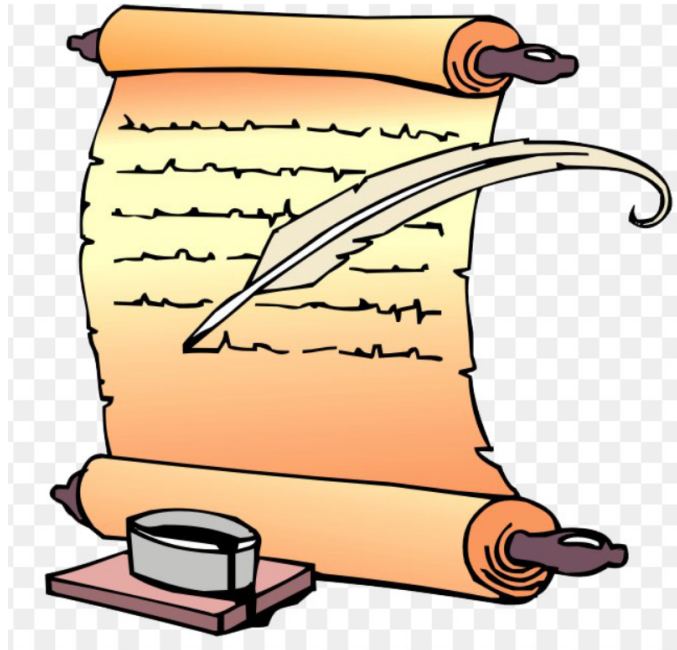


這些鎖都是用密碼來開的。全部黑色的鎖也是同一個密碼[A]，全部綠色的鎖也是同一個密碼[B]。當然，[A]跟[B]是不同的密碼。

當我要跟其他人交換信息的時候，就把信息用這種箱裝著。它可以用黑鎖或綠鎖來鎖上。



然後，我把黑鎖，跟綠鎖的密碼[B]，統統發放出去給所有的人。



發出去的公告，大意像這樣：

“我，Philip C Kwan，是這一對黑色和綠色的鎖的擁有人。  
我會用綠鎖來鎖住我的信息，收信人可以用密碼[B]來打開我的鎖。  
要發信給我的人，請用這些黑鎖來鎖住信息才給我。”

之後如公告所示，當我要發信息給他人的時候，我就用綠鎖來鎖住我的信息。對方就可以用密碼[B]來打開。

當別人要發信息給我的時候，他就用黑鎖來鎖住他的信息。我就可以用密碼[A]來打開。

我用的例子間化了一些細節，但應該能解釋到這些概念：

- 1) 當他人能夠用我發佈的密碼[B]（即公鑰）來打開我的鎖時，這確認了這個信息是從我而來的。換句話說，我不能否認（non-repudiation）信息的真實性（authenticity）。
- 2) 當他人用我發放的黑鎖來加密他要給我的信息時，只有我才能用密碼[A]（即私鑰）來解密信息。

這，就是非對稱密鑰加密的原理。

## 用符號和程式來表達加密和解密等運算

上一段的例子簡單比喻了公開金鑰加密。在其中的鎖頭和箱子，現實上（或者說，電腦世界上）是不存在的。

現實是，公開金鑰加密法只需要私鑰和公鑰，和一些演算法和函數（function）。

讓我用一些數學和函數符號來表達。

我的另外一篇文章，比較詳細的解釋了我的符號和表達方法，請看：

[我的數學和函數符號表達方法](#)

公開金鑰加密有四個主要功能，它們之間也是有關聯的：

- 1) 加密 (encrypt)
- 2) 解密 (decrypt)
- 3) 簽名 (sign, create signature from message)
- 4) 核對簽名 (verify, verify signature of message)

## 加密 (encrypt)

$\text{encrypt}(\langle \text{key} \rangle, \langle \text{message} \rangle) = \langle \text{encrypted\_message} \rangle$

加密函數，需要兩個輸入，即鑰匙 (key) 和信息 (message)，執行把信息變成密文信息 (encrypted message)，作為輸出結果。

信息可以是任何的格式或形式，即任何的二進制數據 (binary data)，例如文字 (text)，圖像 (image)，音頻和視頻 (audio and video) 等等。

輸出的密文信息也是二進制數據，基本上它就是把能解讀的信息變成不能解讀的信息 (converting the readable/processable data, to unreadable/unprocessable data)

假設信息不變，用不同的鑰匙去執行加密，那輸出的密文信息就不一樣。

## 解密 (decrypt)

$\text{decrypt}(\langle \text{key} \rangle, \langle \text{encrypted\_message} \rangle) = \langle \text{message} \rangle$

解密函數，需要鑰匙 (key) 和密文信息 (encrypted message) 兩個輸入，執行把密文信息變回信息 (message)，作為輸出結果。

因為是非對稱密鑰加密的關係，如果信息是用私鑰 (private key) 來加密的，那密文信息就必需要這個私鑰相對應的公鑰 (public key) 來解密。

要是用不相對的鑰匙來解密，那得出的解密信息就一定不會是原來的信息，它大有可能是不能被解讀的信息。

另外，要是密文信息被修改了的話，那就算用相對的鑰匙來解密，都不會成功。哪怕是一個大至 1GB 的信息中只是改了其中的一個 bit 亦然。

用符號來表達的話：

$\text{decrypt}(\langle \text{public\_key\_A} \rangle, \langle \text{encrypted\_message\_by\_private\_key\_A} \rangle) = \langle \text{message} \rangle$

$\text{decrypt}(\langle \text{private\_key\_A} \rangle, \langle \text{encrypted\_message\_by\_public\_key\_A} \rangle) = \langle \text{message} \rangle$

$\text{decrypt}(\langle \text{any\_other\_key} \rangle, \langle \text{encrypted\_message\_by\_private\_key\_A} \rangle) = \text{ERROR, or } \langle \text{garbage\_data} \rangle$

$\text{decrypt}(\langle \text{public\_key\_A} \rangle, \langle \text{encrypted\_message\_by\_private\_key\_A\_with\_some\_modifications} \rangle) = \text{ERROR, or } \langle \text{garbage\_data} \rangle$

小總結一下，加密和解密這對函數的功能是：

- 1) 私密性(privacy) – 把任何數據加密和解密，尤其是確保用公鑰加密的數據，只能給私鑰的擁有者解讀。
- 2) 真實性(authenticity) – 確保數據的製造者的真實性，尤其是用私鑰加密的數據，只能由私鑰的擁有者發佈出來。
- 3) 完整性(integrity) – 確保數據在加密，傳送，到解密的過程中，沒有被篡改。

## 簽名和核對簽名

在資訊科技和互聯網的領域裡，很多的應用（applications）是需要上文那三項功能的。但是，也有很多的應用只是需要真實性和完整性，而不需要私密性的。

例如，區塊鏈和在它上面運行的加密貨幣，它們大部份的運作方法都是把交易，數據，和信息，公開的放在區塊鏈上。

當信息不需要加密，而只需要確保真實性和完整性時，那簽名跟核對簽名的功能就適合了。

$\text{sign}(\langle \text{key} \rangle, \langle \text{message} \rangle) = \langle \text{signature} - \text{encrypted\_hash\_of\_message} \rangle$

$\text{verify\_signature}(\langle \text{key} \rangle, \langle \text{signature} \rangle, \langle \text{message} \rangle) = \langle \text{true} \mid \text{false} - \text{result of verification} \rangle$

簽名函數，跟加密函數一樣，需要鑰匙和信息。執行時把信息做一個哈希（hash），然後加密這個哈希，作為輸出結果，即簽名（signature）。

（關於哈希的細節，這裡有一些網上的補充資料，也容許我另文再述：

[Wikipedia | Hash Function](#)）

核對簽名函數，跟解密函數相似但不一樣。它需要三個輸入：相對的鑰匙，簽名，信息（即原本的，沒有加密的信息）。

執行時，像簽名函數的第一步一樣，把信息做一個哈希。

然後，用鑰匙解密簽名，得出哈希。

然後，比較這兩個哈希。

當這兩個哈希是一樣的話，就代表信息是從鑰匙的擁有者而來的，就輸出布林值（boolean）的 true，否則是 false。

（關於布林值：[Wikipedia | Boolean Data Type](#)）

舉一個簡單的例子，我想把我的投票意向表達在區塊鏈上，我只需要把我的信息：

“I choose to vote for plan B”

跟這個信息的簽名，放到區塊鏈上去。

之後，一些其他人就可以核對和處理我的投票了，而大部份人都可以直接的（i.e. 不用解密）看到我的投票。

## 再解說種子與加密鑰匙對的關係

上一篇文章還有一點沒有解釋：

2) 我的加密錢包是用18，或是24個詞語來組合BIP-39種子的，那它們是否比較安全？

客觀的從種子的長度來看，一定是24個的詞語組合/256bit的種子會比較安全，因為越長的長度代表越多的複雜性，代表越難去被暴力破解（brute-force attack）。

但是，正如我在之前的文章說過，128bit 其實是有足夠的複雜性來保障加密錢包的。

就讓我提供這些參考：

[12 vs. 24 words seed](#)

[How secure is the seed phrase \(12 words, 24 words\)](#)

再詳細去解釋的話，從詞語組合（seed phrases）到產生加密鑰匙對，中間有兩個步驟。

1) 產生種子（generate seed）

2) 產生加密鑰匙對（generate key pair）

讓我繼續用函數符號來表達。

`generateSeed(<list of seed phrases>) = <seed>`

`generateKeyPair(<seed>) = <privateKey>, <publicKey>`

`generateSeed`函數，輸入詞語組合（即12，18，或24個的詞語），執行後輸出種子。

之前的文章解說過，12個的詞語組合可產生128bit的長度的數字，而24個的詞語組合則可產生256bit的長度的數字。

`generateKeyPair`函數，輸入從`generateSeed`函數的種子，執行後輸出加密鑰匙對。

以上的表達省略了很多術語，細節和較深奧的演算，比如：

1) 熵（entropy）- 它是用來量化和表達產生種子的隨機性（randomness）

2) 分層確定性錢包（hierarchical deterministic wallet，或HD wallet）- 實際上的產生加密鑰匙對的演算法。

因篇幅所限，加上我覺得要我簡潔但準確地解釋這些題目是有點難度，讓我先提供一些參考給大家：

[learn me a bitcoin | Mnemonic Seed](#)

[learn me a bitcoin | HD Wallets](#)

目標受眾/標籤：技術學習，加密學，電腦科學

技術難度：中級

♡ Like


💬 Comment

➦ Share



Write a comment...

© 2022 Philip Kwan · [Privacy](#) · [Terms](#) · [Collection notice](#)

 **Get the Substack app**

Substack is the home for great writing