WIKIPEDIA

# RSA加密演算法

維基百科，自由的百科全書

**RSA加密演算法**是一種非對稱加密演算法。在公開金鑰加密和電子商業中RSA被廣泛使用。RSA是1977年由羅納德·李維斯特（Ron Rivest）、阿迪·薩莫爾（Adi Shamir）和倫納德·阿德曼（Leonard Adleman）一起提出的。當時他們三人都在麻省理工學院工作。RSA就是他們三人姓氏開頭字母拼在一起組成的。

RSA的作者之一：阿迪·薩莫爾（Adi Shamir）

1973年，在英國政府通訊總部工作的數學家克利福德·柯克斯（Clifford Cocks）在一個內部檔案中提出了一個相同的演算法，但他的發現被列入機密，一直到1997年才被發表。

對極大整數做因數分解的難度決定了RSA演算法的可靠性。換言之，對一極大整數做因數分解愈困難，RSA演算法愈可靠。假如有人找到一種快速因數分解的演算法的話，那麼用RSA加密的訊息的可靠性就肯定會極度下降。但找到這樣的演算法的可能性是非常小的。今天只有短的RSA鑰匙才可能被強力方式破解。到目前為止，世界上還沒有任何可靠的攻擊RSA演算法的方式。只要其鑰匙的長度足夠長，用RSA加密的訊息實際上是不能被破解的。

1983年麻省理工學院在美國為RSA演算法申請了專利。這個專利2000年9月21日失效。由於該演算法在申請專利前就已經被發表了，在世界上大多數其它地區這個專利權不被承認。

## 目錄


操作
- 公鑰與私鑰的產生
- 加密訊息
- 解密訊息
- 簽名訊息

安全

實現細節
- 金鑰生成
- 速度
- 金鑰分配
- 時間攻擊

典型金鑰長度

已公開的或已知的攻擊方法

相關條目

參考文獻

外部連結


## 操作

### 公鑰與私鑰的產生

假設Alice想要通過一個不可靠的媒體接收Bob的一條私人訊息。她可以用以下的方式來產生一個**公鑰**和一個**私鑰**：

1. 隨意選擇兩個大的質數$p$和$q$，$p$不等於$q$，計算$N = pq$。
2. 根據歐拉函式，求得$r = \varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$
3. 選擇一個小於$r$的整數$e$，使$e$與$r$互質。並求得$e$關於$r$的模反元素，命名為$d$（求$d$令$ed \equiv 1 \pmod{r}$）。（模反元素存在，若且唯若$e$與$r$互質）
4. 將$p$和$q$的記錄銷毀。

$(N, e)$是公鑰，$(N, d)$是私鑰。Alice將她的公鑰$(N, e)$傳給Bob，而將她的私鑰$(N, d)$藏起來。

### 加密訊息

假設Bob想給Alice送一個訊息$m$，他知道Alice產生的$N$和$e$。他使用起先與Alice約好的格式將$m$轉換為一個小於$N$，且與$N$互質的整數$n$，比如他可以將每一個字轉換為這個字的Unicode碼，然後將這些數字連在一起組成一個數字。假如他的資訊非常長的話，他可以將這個資訊分為幾段，然後將每一段轉換為$n$。用下面這個公式他可以將$n$加密為$c$：

$$c \equiv n^e \pmod{N}$$

計算$c$並不複雜。Bob算出$c$後就可以將它傳遞給Alice。

### 解密訊息

Alice得到Bob的訊息$c$後就可以利用她的金鑰$d$來解碼。她可以用以下這個公式來將$c$轉換為$n$：

$$c^d \equiv n \pmod{N}$$

得到$n$後，她可以將原來的資訊$m$重新復原。

解碼的原理是

$$c^d \equiv n^{e \cdot d} \pmod{N}$$

已知$ed \equiv 1 \pmod{r}$，即 $ed = 1 + h\varphi(N)$。 由歐拉定理得：

$$n^{ed} = n^{1+h\varphi(N)} = n\left(n^{\varphi(N)}\right)^h \equiv n(1)^h \pmod{N} \equiv n \pmod{N}$$

### 簽名訊息

RSA也可以用來為一個訊息署名。假如Alice想給Bob傳遞一個署名的訊息的話，那麼她可以為她的訊息計算一個雜湊值（Message digest），然後用她的私鑰「加密」（如同前面「加密訊息」的步驟）這個雜湊值並將這個「署名」加在訊息的後面。這個訊息只有用她的公鑰才能被解密。Bob獲得這個訊息後可以用Alice的公鑰「解密」（如同前面「解密訊息」的步驟）這個雜湊值，然後將這個資料與他自己為這個訊息計算的雜湊值相比較。假如兩者相符的話，那麼Bob就可以知道發信人持有Alice的私鑰，以及這個訊息在傳播路徑上沒有被篡改過。

## 安全

假設偷聽者乙獲得了甲的公鑰$N$和$e$以及丙的加密訊息$c$，但她無法直接獲得甲的金鑰$d$。要獲得$d$，最簡單的方法是將$N$分解為$p$和$q$，這樣她可以得到同餘方程$de = 1(\mathrm{mod}(p-1)(q-1))$並解出$d$，然後代入解密公式

$$c^d \equiv n \pmod{N}$$

匯出$n$（破密）。但至今為止還沒有人找到一個多項式時間的演算法來分解一個大的整數的因子，同時也還沒有人能夠證明這種演算法不存在（見因數分解）。

至今為止也沒有人能夠證明對$N$進行因數分解是唯一的從$c$匯出$n$的方法，但今天還沒有找到比它更簡單的方法。（至少沒有公開的方法。）

因此今天一般認為只要$N$足夠大，那麼駭客就沒有辦法了。

假如$N$的長度小於或等於256位，那麼用一台個人電腦在幾個小時內就可以分解它的因子了。1999年，數百台電腦合作分解了一個512位元長的$N$。今天對$N$的要求是它至少要1024位元長。

1994年彼得·秀爾（Peter Shor）證明一台量子電腦可以在多項式時間內進行因數分解。假如量子電腦有朝一日可以成為一種可行的技術的話，那麼秀爾的演算法可以淘汰RSA和相關的衍生演算法。（即依賴於分解大整數困難性的加密演算法）

假如有人能夠找到一種有效的分解大整數的演算法的話，或者假如量子電腦可行的話，那麼在解密和製造更長的鑰匙之間就會展開一場競爭。但從原理上來說RSA在這種情況下是不可靠的。

# 實現細節

## 金鑰生成

首先要使用機率演算法來驗證隨機產生的大的整數是否質數，這樣的演算法比較快而且可以消除掉大多數非質數。假如有一個數通過了這個測試的話，那麼要使用一個精確的測試來保證它的確是一個質數。

除此之外這樣找到的$p$和$q$還要滿足一定的要求，首先它們不能太靠近，此外$p-1$或$q-1$的因子不能太小，否則的話$N$也可以被很快地分解。

此外尋找質數的演算法不能給攻擊者任何資訊，這些質數是怎樣找到的，尤其產生亂數的軟體必須非常好。要求是隨機和不可預測。這兩個要求並不相同。一個隨機過程可能可以產生一個不相關的數的系列，但假如有人能夠預測出（或部分地預測出）這個系列的話，那麼它就已經不可靠了。比如有一些非常好的亂數演算法，但它們都已經被發表，因此它們不能被使用，因為假如一個攻擊者可以猜出$p$和$q$一半的位的話，那麼他們就已經可以輕而易舉地推算出另一半。

此外金鑰$d$必須足夠大，1990年有人證明假如$p$大於$q$而小於$2q$（這是一個很經常的情況）而$d < \frac{1}{3} \times N^{\frac{1}{4}}$，那麼從$N$和$e$可以很有效地推算出$d$。此外$e=2$永遠不應該被使用。

## 速度

比起DES和其它對稱演算法來說，RSA要慢得多。實際上Bob一般使用一種對稱演算法來加密他的資訊，然後用RSA來加密他的比較短的對稱密碼，然後將用RSA加密的對稱密碼和用對稱演算法加密的訊息送給Alice。

## 金鑰分配

和其它加密過程一樣，對RSA來說分配公鑰的過程是非常重要的。分配公鑰的過程必須能夠抵擋中間人攻擊。假設Eve交給Bob一個公鑰，並使Bob相信這是Alice的公鑰，並且她可以截下Alice和Bob之間的資訊傳遞，那麼她可以將她自己的公鑰傳給Bob，Bob以為這是Alice的公鑰。Eve可以將所有Bob傳遞給Alice的訊息截下來，將這個訊息用她自己的金鑰解密，讀這個訊息，然後將這個訊息再用Alice的公鑰加密後傳給Alice。理論上Alice和Bob都不會發現Eve在偷聽他們的訊息。今天人們一般用可靠的第三方機構簽發憑證來防止這樣的攻擊。

## 時間攻擊

1995年有人提出了一種非常意想不到的攻擊方式：假如Eve對Alice的硬體有充分的了解，而且知道它對一些特定的訊息加密時所需要的時間的話，那麼她可以很快地推匯出$d$。這種攻擊方式之所以會成立，主要是因為在進行加密時所進行的模指數運算是一個位元一個位元進行的，而位元為1所花的運算比位元為0的運算要多很多，因此若能得到多組訊息與其加密時間，就會有機會可以反推出私鑰的內容。

# 典型金鑰長度

1997年後開發的系統，用戶應使用1024位元金鑰，憑證認證機構應用2048位元或以上。

# 已公開的或已知的攻擊方法

針對RSA最流行的攻擊一般是基於大數因數分解。1999年，RSA-155 (512 bits)被成功分解，花了五個月時間（約8000 MIPS年）和224 CPU hours在一台有3.2G中央記憶體的Cray C916電腦上完成。

RSA-158表示如下：

```
395058745832651445264197678006144819960207764603049364541393760515793556265294506836 09
727842468219535093544305870490251995655335710209799226484977949442955603

= 338849583746672139436839320467218152281583036860499304808492584055528 1177×
  116588234066712599031483765583832708181310122581463926004395209941313443341629245361 39
```

2009年12月12日，編號為RSA-768（768 bits, 232 digits）數也被成功分解[1]。這一事件威脅了現通行的1024-bit金鑰的安全性，普遍認為用戶應儘快升級到2048-bit或以上。

RSA-768表示如下：

```
123018668453011775513049495838496272077285356959533479219732245215172640050726
365751874520219978646938995647494277406384592519255732630345373154826850791702
612214291346167042921431160222124047927473779408066535141959745985690214341 3

= 334780716989568987860441698482126908177047949837137685689 1
  243138898288379387800228761471165253174308773781446799948 9×
  367460436667995904282446337996279526322791581643430876426 7
  603228381573966651127923337341714339681027009279873630891 7
```

# 相關條目

- 公開金鑰加密
- 量子電腦
- 秀爾演算法

# 參考文獻

1. Factorization of a 768-bit RSA modulus (PDF). 2010年1月7日 [2010年1月10日].

# 外部連結

- RSA, The Security Division of EMC (http://www.rsasecurity.com)
- RSA演算法詳解 (http://www.guideep.com/read?guide=5676830073815040)