1，其中 a 與 N 互質，所以我們可知 $a \times a^{\phi(N)-1} \bmod N = 1$，眼尖的讀者不難發現 a 模 N 的乘法反元素 $x = a^{\phi(N)-1} \bmod N$。但在 RSA 密碼系統中卻無法利用這個方法去求得私密金鑰 d：

$$e \times e^{\phi(\phi(N))-1} \bmod \phi(N) = 1$$

令 $d = e^{\phi(\phi(N))-1} \bmod \phi(N)$。駭客雖然知道 N，但不知道 $\phi(N)$，因此駭客沒辦法導出使用者的私密金鑰。另一方面，即使合法使用者知道 $\phi(N)$ 的值，要得到 $\phi(\phi(N))$ 的值還是有點難度。

另一種方法稱之爲擴充的歐幾里德演算法 (Extended Euclidean Algorithm)，這種方法爲歐幾里德演算法的延伸，歐幾里德演算法是求解最大公因數最著名的方法之一，其方法說明如下。假設欲求 a 與 n 之最大公因數 $\gcd(a, N)$，首先就先令 $r_0 = N$、$r_1 = a$（設 $N > a$），然後我們可以求得：

$$\begin{aligned}
r_0 &= r_1 g_1 + r_2 \\
r_1 &= r_2 g_2 + r_3 \\
&\vdots \\
r_{j-2} &= r_{j-1} g_{j-1} + r_j \\
&\vdots \\
r_{m-3} &= r_{m-2} g_{m-2} + r_{m-1} \qquad (7.1) \\
r_{m-2} &= r_{m-1} g_{m-1} + r_m \qquad (7.2) \\
r_{m-1} &= r_m g_m \qquad (7.3)
\end{aligned}$$

最後求得的 r_m 就是 a 與 N 的最大公因數，這個方法我們又稱之爲輾轉相除法。例如，求 $\gcd(155, 496)$ 如下：

$$\begin{aligned}
496 &= 155 \times 3 + 31 \\
155 &= 31 \times 5
\end{aligned}$$

所以 $\gcd(155, 496) = 31$。

那麼如何利用擴充的歐幾里德演算法來求乘法反元素呢？因爲計算 $d = e^{-1} \bmod N$ 的首要條件就是 e 與 N 必須要互質，因此 $r_m = \gcd(e, N) = 1$，從公式 (7.1) 及 (7.2) 得到

$$\begin{aligned} r_m &= r_{m-2} - r_{m-1}g_{m-1} \\ &= r_{m-2} - (r_{m-3} - r_{m-2}g_{m-2})g_{m-1} \\ &= (1 + g_{m-1}g_{m-2})r_{m-2} - g_{m-1}r_{m-3} \end{aligned}$$

依此類推，最後一定可以求得兩個整數 d 與 t，使得 $r_m = dr_1 + tr_0$，也就是 $de + tN = 1$，其中的整數 d 其實就是我們要求的乘法反元素 $d = e^{-1} \bmod N$，因爲

$$\begin{aligned} 1 &= de + tN \bmod N \\ &= de \bmod N \end{aligned}$$

在 RSA 的密碼系統中，因爲使用者知道 $\phi(N)$，故使用者可以利用擴充的歐幾里德演算法來求得其私密金鑰 $d = e^{-1} \bmod \phi(N)$。但是其他人只知道 N 並不知道 $\phi(N)$，故他們無法從公開金鑰 (e, N) 去求得私密金鑰 d，但是若他們有辦法將 N 分解成兩個大質數 p 跟 q，那麼 $\phi(N) = (p-1)(q-1)$ 就可以被求得。一旦 $\phi(N)$ 被求得，私密金鑰 d 就可以很容易被得知，RSA 密碼系統也就不安全了。

所幸，要因數分解 N 是個很困難的問題，根據 Rivest 等人的記載，要分解一個 200 位的十進位數約需 40 億年的計算，而分解一個 500 位數則約 10^{25} 年，而這兩種估計均以最佳演算法，且每指令執行的平均時間爲 1 微秒計算，即使電腦速度往後每十年就增加數倍；如此這樣的計算仍約需數百年的時間才能造出真正可分解 500 位數的電腦。一旦有人宣稱他可以在很短的時間（如一年內）將一個 512 位元之大數因數分解出二個很大質數之相乘積，那麼 RSA 再也不安全了，因此我們常說 RSA 密碼系統之安全是植基於因數分解的困難度。