¿Son correctas las claves RSA que generamos con Win64 OpenSSL 1.1.0g? Intentando cifrar con RSA cuando p y q no son primos (*ver nota en pág. 12*)

Autor: Jorge Ramió Aguirre (jramio@etsisi.upm.es, www.criptored.com)

Resumen: ¿Qué sucedería si al generar una clave RSA no usamos p y q como números primos? ¿Sigue funcionando el algoritmo? Son preguntas frecuentes cuando se estudia por primera vez este popular algoritmo. La respuesta es no. RSA sólo funciona correctamente si generamos la clave con p y q primos. En este trabajo demostraremos de una manera simple y práctica por qué no funciona RSA si p y q no son primos. Veremos, además, algunos ejemplos en los que Win64 OpenSSL en su última versión 1.1.0g de 2 de noviembre de 2017, no genera claves RSA correctas, entregando valores p y q no primos o calculando incorrectamente el cuerpo de cifra n y la clave privada d. Es lo que se ha podido observar sólo para claves pequeñas de hasta 128 bits, pero que el autor no había apreciado en versiones anteriores ni en Win32 OpenSSL 1.1.0g.

1. El algoritmo RSA en pocas palabras

Es de todos conocido cómo funciona el algoritmo propuesto hace ya 40 años por Rivest, Shamir y Adleman, que a fecha de hoy (enero de 2018) sigue siendo un estándar. Por lo tanto, en este documento no se entrará a demostrar las características de sus claves, operaciones de cifra que realiza o ataques factibles contra su seguridad. Presentaremos, no obstante, un breve resumen de cómo se generan las claves y cómo se cifra y se descifra con este algoritmo. El lector interesado en ello, puede refrescar estas ideas accediendo a las 10 lecciones del curso El algoritmo RSA[1] del MOOC Crypt4you de CriptoRed, publicado por este mismo autor en 2012.

Generación de una clave RSA:

- Se eligen dos primos p y q diferentes, grandes, de igual o similar tamaño.
- El producto pxq de esos dos primos nos entrega el módulo n o cuerpo de cifra.
- El dueño de la clave calcula el Indicador de Euler $\phi(n) = (p-1)(q-1)$ o cuerpo trampa.
- Se busca un valor de clave pública e de forma que mcd $(e, \phi(n)) = 1$, si bien en la práctica se fuerza a que todos usemos el número 4 de Fermat, es decir $2^{2^4}+1 = 2^{16}+1 = 65.537$.
- Con el Algoritmo Extendido de Euclides AEE, se calcula la clave privada d = inv $(e, \phi(n))$.
- Se hacen públicos los valores de n y e.
- Se guardan en secreto los valores de p, q y d. Los valores de p y q servirán al dueño de la clave para descifrar de manera óptima un mensaje confidencial, usando el Teorema Chino de los Restos TCR, dado que esta operación por lo general requiere de mucho cómputo.

Operaciones de cifra y firma con RSA:

- La cifra de un número secreto N para lograr confidencialidad será: N^e mod n.
- Esta operación la pueden hacer todos aquellos que conozcan la clave pública del sujeto.
- La firma sobre un hash h(M) para lograr integridad y autenticidad será: $h(M)^d$ mod n.
- Esta operación la puede hacer solamente el dueño de la clave, que conoce d.

[1] MOOC Crypt4you, curso El algoritmo RSA, Jorge Ramió Aguirre, marzo de 2012. http://www.criptored.upm.es/crypt4you/temas/RSA/leccion0/leccion00.html

2. Ejemplo de una clave RSA no válida de 80 bits generada con Win64 OpenSSL

Web official de OpenSSL[2], acceso el 10/01/18: "*Note: The latest stable version is the 1.1.0 series. The 1.0.2 series is our Long Term Support (LTS) release, supported until 31st December 2019. The 0.9.8, 1.0.0 and 1.0.1 versions are now out of support and should not be used.*"

Usando OpenSSL-Win64, versión 1.1.0g[3] del 2 de noviembre de 2017, descargado desde Shining Light Productions, y el software genRSA v2.1[4] de octubre de 2017, vamos a generar una clave, cifrar y descifrar, mostrando como resultado simplemente las capturas de pantalla.

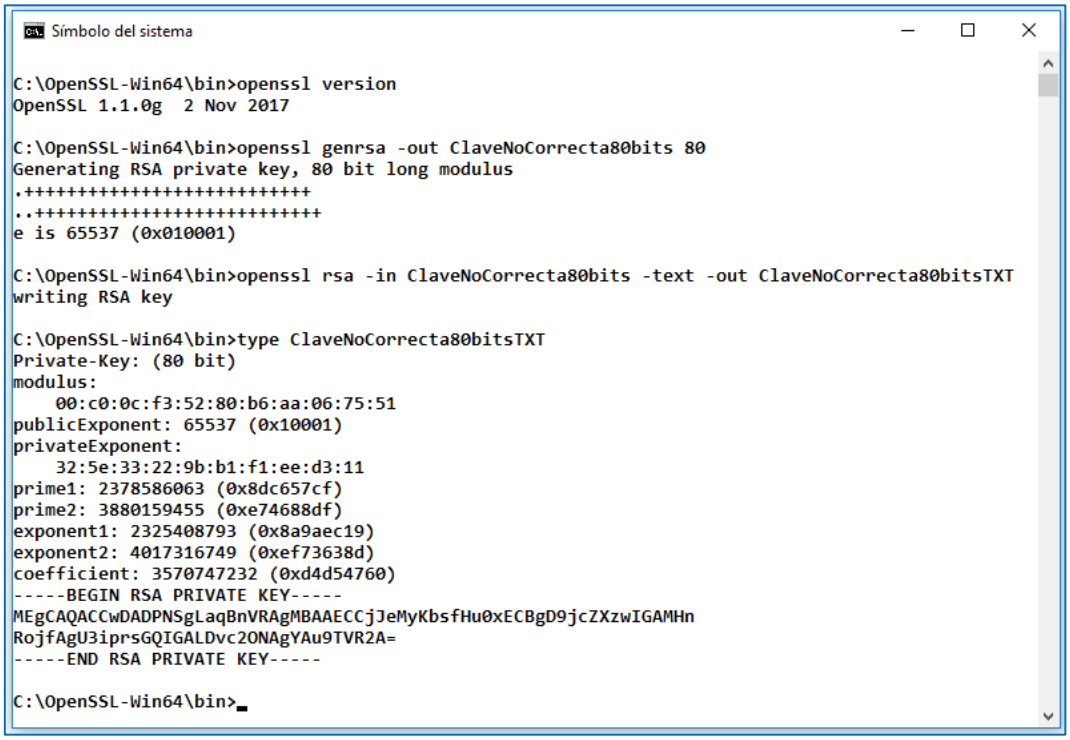

Figura 1. Generación de una clave RSA de 80 bits con Win64 OpenSSL.

Comentarios sobre la clave generada (datos en hexadecimal).

- prime1: 2378586063 (0x8dc657cf). No es primo, tiene factores 3, 7, 113266003.
- prime2: 3880159455 (0xe74688df). No es primo, tiene factores 3, 5, 23^2, 488993.
- modulus: 00:c0:0c:f3:52:80:b6:aa:06:75:51.
  Es incorrecto porque: 0x8dc657cf x 0xe74688df = 0x80150944AA067551 (64 bits)
- publicExponent: 65537 (0x10001). Es correcto.
- privateExponent: 32:5e:33:22:9b:b1:f1:ee:d3:11. Es incorrecto pues la clave privada debe ser d = 7224CD2564BF8BB5
- Se pide que genere una clave de 80 bits y OpenSSL nos ha generado una clave de 64 bits.

Aunque p y q no sean números primos, se procede ahora a generar esa misma clave con el software de prácticas genRSA v2.1 y se obtiene la clave que se muestra en la figura 2.

[2] OpenSSL Downloads. https://www.openssl.org/source/

[3] OpenSSL Shining Light Productions. https://slproweb.com/products/Win32OpenSSL.html

[4] genRSA v2.1: Software para la Generación de Claves RSA, Cifra, Firma y Ataques. Rodrigo Díaz Arroyo, Jorge Ramió Aguirre. http://www.criptored.upm.es/software/sw_m001d.htm

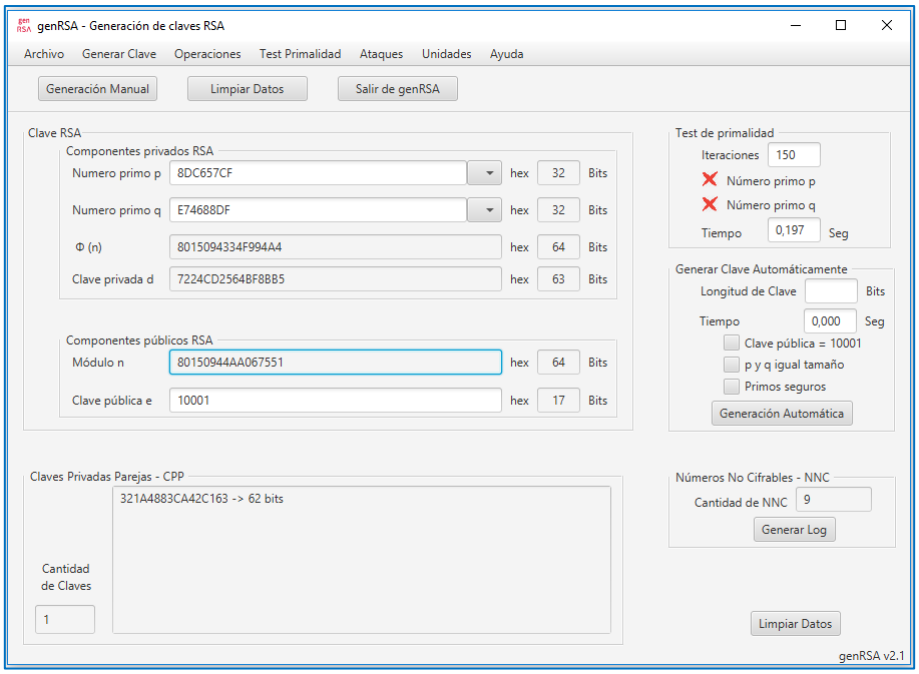

Figura 2. Clave generada con genRSA v2.1 con los valores p, q y e, entregados por OpenSSL.

Nota. *El software de prácticas genRSA v2.1 permite que el usuario introduzca cualquier número p y q durante la Generación Manual de una clave, siempre que no sean pares. Esto es así precisamente para poder comprobar qué sucedería si no se usan números primos para p y q. Lógicamente, en la generación automática de claves, el programa sólo usa números primos para los parámetros p y q, es decir, comprueba dicha primalidad antes de generar la clave*.

Como se observa en la figura 2, la clave se ha generado en hexadecimal siguiendo los pasos básicos del algoritmo RSA, pero no supera el test de primalidad. Si lo desea, puede comprobar los valores del módulo n, del indicador de Euler $\phi(n)$, de la clave privada d = inv (e, $\phi(n)$), así como el cifrado y descifrado haciendo uso, por ejemplo, del software online Mobilefish[5].

Todos los datos están en hexadecimal.

- n = pxq = 8DC657CF x E74688DF = 80150944AA067551
- $\phi(n)$ = (p-1)(q-1) = 8DC657CE x E74688DE = 8015094334F994A4
- d = inv (e, $\phi(n)$) = inv (10001, 8015094334F994A4) = 7224CD2564BF8BB5
- Supongamos que queremos cifrar el número secreto N = F80643D7A1
- En el cifrado con e = 10001, la operación será N^e mod n = C
- $F80643D7A1^{10001}$ mod 80150944AA067551 = 2AA2459065304A75
- En el descifrado con d = 7224CD2564BF8BB5, la operación será C^d mod n = N
  $2AA2459065304A75^{7224CD2564BF8BB5}$ mod 80150944AA067551 = 666B6A2E14840973
- Pero 666B6A2E14840973 ≠ F80643D7A1
- No se recupera el texto en claro. La clave no es válida para cifrar.

En la figura 3 se muestra la captura de pantalla de genRSA v2.1 para estas operaciones.

[5] Mobilefish Big number equation calculation, software online:
https://www.mobilefish.com/services/big_number_equation/big_number_equation.php#equation_output

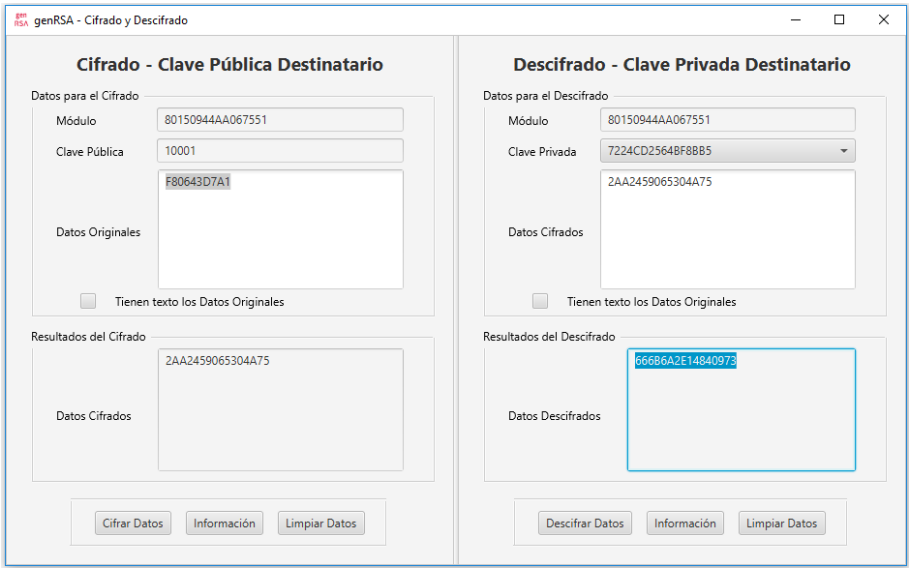

Figura 3. No se recupera el texto el claro porque la clave generada por OpenSSL no es válida.

3. Comportamiento del algoritmo RSA con claves verdaderas y falsas

Veamos qué sucede cuando se cifran todos los valores o restos del cuerpo de cifra o módulo. Lo haremos mediante un ejemplo gráfico con un par de claves (la primera correcta y la segunda incorrecta) con números muy pequeños, para que sea más fácil apreciar el efecto. Obviamente, a similar resultado se llega con claves reales y actuales de 2.048 bits o más.

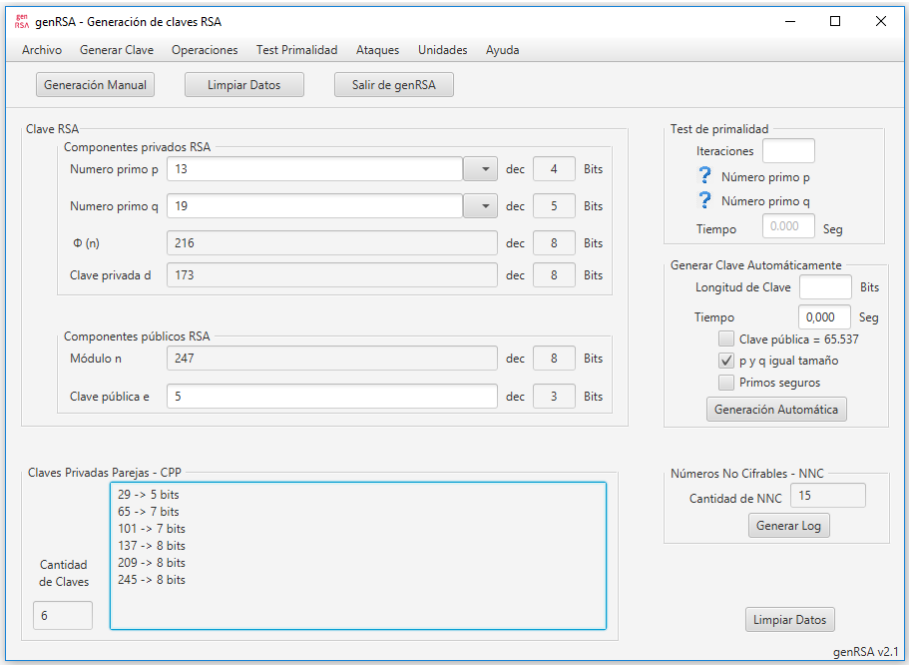

Figura 4. Clave de 8 bits válida en RSA usada para el primer ejemplo.

Con p = 13, q = 19, n = 247, $\phi(n) = 216$, e = 5 y d = 173, vamos a cifrar los primeros 100 restos del cuerpo n, desde N = 0 hasta N = 99. La cifra $C = N^e \bmod n$, es decir, $C = N^5 \bmod 247$ nos entrega la siguiente tabla, donde todos los resultados son diferentes. Marcados en color azul aparecen los números no cifrables o que se transmiten en claro.

N=0 C=0	N=1 C=1	N=2 C=32	N=3 C=243	N=4 C=36	N=5 C=161	N=6 C=119	N=7 C=11	N=8 C=164	N=9 C=16
N=10 C=212	N=11 C=7	N=12 C=103	N=13 C=52	N=14 C=105	N=15 C=97	N=16 C=61	N=17 C=101	N=18 C=18	N=19 C=171
N=20 C=115	N=21 C=203	N=22 C=224	N=23 C=17	N=24 C=85	N=25 C=233	N=26 C=182	N=27 C=183	N=28 C=149	N=29 C=22
N=30 C=140	N=31 C=122	N=32 C=223	N=33 C=219	N=34 C=21	N=35 C=42	N=36 C=82	N=37 C=189	N=38 C=38	N=39 C=39
N=40 C=222	N=41 C=110	N=42 C=74	N=43 C=218	N=44 C=5	N=45 C=106	N=46 C=50	N=47 C=73	N=48 C=3	N=49 C=121
N=50 C=46	N=51 C=90	N=52 C=143	N=53 C=40	N=54 C=175	N=55 C=139	N=56 C=75	N=57 C=57	N=58 C=210	N=59 C=89
N=60 C=34	N=61 C=55	N=62 C=199	N=63 C=176	N=64 C=220	N=65 C=221	N=66 C=92	N=67 C=136	N=68 C=178	N=69 C=179
N=70 C=109	N=71 C=67	N=72 C=154	N=73 C=99	N=74 C=120	N=75 C=56	N=76 C=228	N=77 C=77	N=78 C=13	N=79 C=53
N=80 C=188	N=81 C=9	N=82 C=62	N=83 C=239	N=84 C=145	N=85 C=206	N=86 C=60	N=87 C=159	N=88 C=160	N=89 C=33
N=90 C=181	N=91 C=78	N=92 C=118	N=93 C=6	N=94 C=113	N=95 C=114	N=96 C=96	N=97 C=184	N=98 C=167	N=99 C=112
No es necesario mostrar más resultados. Al realizar las operaciones de cifra desde N = 0 hasta N = 246, se obtienen en C todos los restos del cuerpo n = 247.									

Tabla 1. Cifra de los restos del cuerpo n, obteniendo siempre valores distintos dentro de n.

Como se observa en la Tabla 1, a medida que se van cifrando los diferentes valores de N, se van obteniendo todos los restos del cuerpo de cifra n. Esto es lo correcto.

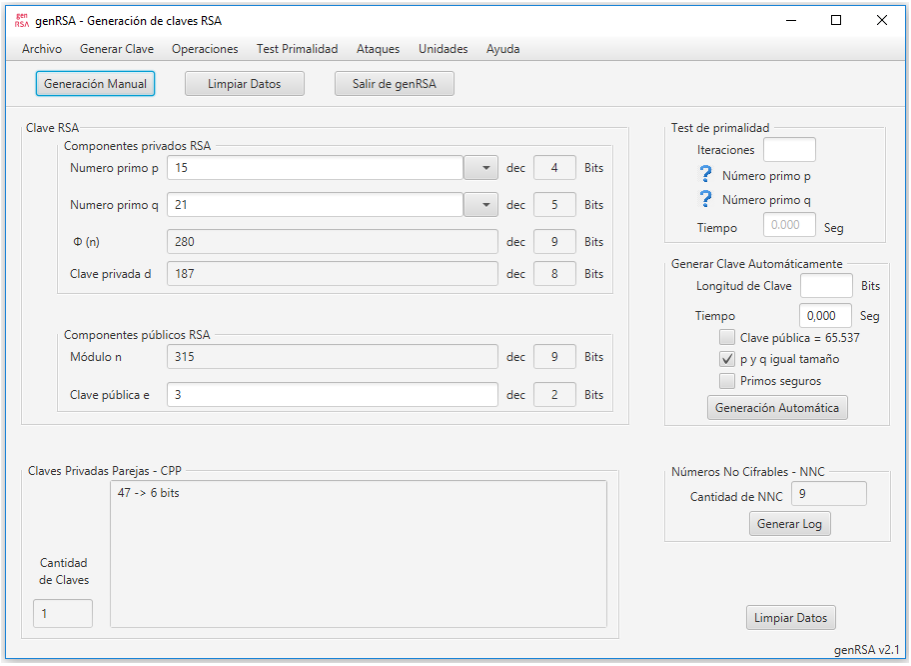

Figura 5. Clave de 9 bits no válida en RSA usada para el segundo ejemplo.

Pero si generamos una clave de tamaño similar en donde p y q no sean primos, además de lo que ya hemos visto, es decir que no se recupera el texto en claro usando la clave privada y, por lo tanto, no tiene sentido el sistema de cifra así planteado, nos encontraremos con la anomalía en la cifra que se muestra en la Tabla 2.

Ahora con p = 15, q = 21, ambos no primos; n = 315, $\phi(n) = 280$, e = 3 y d = 187, vamos a cifrar nuevamente los 100 primeros restos del cuerpo n, desde N = 0 hasta N = 99. En este caso, la cifra $C = N^e \bmod n = N^3 \bmod 315$ nos entrega los resultados que se muestran en la Tabla 2. Para hacer más fácil la lectura de la tabla, no se marcarán en color los números no cifrables, solamente se marcarán aquellos números en que el resultado de la cifra se repite.

Como hay muchos valores del criptograma C repetidos en la tabla 2, se marcarán todas esas repeticiones en un solo color, el amarillo, sin diferenciar unos de otros.

N=0 C=0	N=1 C=1	N=2 C=8	N=3 C=27	N=4 C=64	N=5 C=125	N=6 C=216	N=7 C=28	N=8 C=197	N=9 C=99
N=10 C=55	N=11 C=71	N=12 C=153	N=13 C=307	N=14 C=224	N=15 C=225	N=16 C=1	N=17 C=188	N=18 C=162	N=19 C=244
N=20 C=125	N=21 C=126	N=22 C=253	N=23 C=197	N=24 C=279	N=25 C=190	N=26 C=251	N=27 C=153	N=28 C=217	N=29 C=134
N=30 C=225	N=31 C=181	N=32 C=8	N=33 C=27	N=34 C=244	N=35 C=35	N=36 C=36	N=37 C=253	N=38 C=62	N=39 C=99
N=40 C=55	N=41 C=251	N=42 C=63	N=43 C=127	N=44 C=134	N=45 C=90	N=46 C=1	N=47 C=188	N=48 C=27	N=49 C=154
N=50 C=260	N=51 C=36	N=52 C=118	N=53 C=197	N=54 C=279	N=55 C=55	N=56 C=161	N=57 C=288	N=58 C=127	N=59 C=314
N=60 C=225	N=61 C=181	N=62 C=188	N=63 C=252	N=64 C=64	N=65 C=260	N=66 C=216	N=67 C=253	N=68 C=62	N=69 C=279
N=70 C=280	N=71 C=71	N=72 C=288	N=73 C=307	N=74 C=134	N=75 C=90	N=76 C=181	N=77 C=98	N=78 C=162	N=79 C=64
N=80 C=125	N=81 C=36	N=82 C=118	N=83 C=62	N=84 C=189	N=85 C=190	N=86 C=71	N=87 C=153	N=88 C=127	N=89 C=314
N=90 C=90	N=91 C=91	N=92 C=8	N=93 C=162	N=94 C=244	N=95 C=260	N=96 C=216	N=97 C=118	N=98 C=287	N=99 C=99
No es necesario continuar las cifras hasta N = n-1 = 314. El comportamiento es claro, se observan muchos resultados de cifra C que se repiten y esto no es aceptable.									

Tabla 2. Cifra de los restos del cuerpo n, obteniendo muchos valores repetidos.

A continuación se muestran algunos ejemplos de criptogramas C iguales obtenidos cifrando números N diferentes:

- $1^3 \bmod 315 = 16^3 \bmod 315 = 46^3 \bmod 315$ C = 1
- $2^3 \bmod 315 = 32^3 \bmod 315 = 92^3 \bmod 315$ C = 8
- $3^3 \bmod 315 = 33^3 \bmod 315 = 48^3 \bmod 315$ C = 27
- $4^3 \bmod 315 = 64^3 \bmod 315 = 79^3 \bmod 315$ C = 64
- $5^3 \bmod 315 = 20^3 \bmod 315 = 80^3 \bmod 315$ C = 125
- $6^3 \bmod 315 = 66^3 \bmod 315 = 96^3 \bmod 315$ C = 216
- $8^3 \bmod 315 = 23^3 \bmod 315 = 53^3 \bmod 315$ C = 197
- $9^3 \bmod 315 = 39^3 \bmod 315 = 99^3 \bmod 315$ C = 99
- $10^3 \bmod 315 = 40^3 \bmod 315 = 55^3 \bmod 315$ C = 55
- $11^3 \bmod 315 = 71^3 \bmod 315 = 86^3 \bmod 315$ C = 71
- $12^3 \bmod 315 = 27^3 \bmod 315 = 87^3 \bmod 315$ C = 153
- $15^3 \bmod 315 = 30^3 \bmod 315 = 60^3 \bmod 315$ C = 225
- $17^3 \bmod 315 = 47^3 \bmod 315 = 62^3 \bmod 315$ C = 188
- Etcétera.

A la vista de los resultados y las distancias observadas, con repeticiones de 30 y 60 entre los diferentes valores de N, parece que existe un cierto comportamiento matemático detrás de este fenómeno, pero no es lo que estamos buscando en este trabajo.

Es más que suficiente observar que en criptografía no es aceptable que diferentes valores N secretos se cifren como un mismo criptograma C, o lo que es lo mismo, que al descifrar un criptograma C podamos obtener diferentes textos en claro. Es decir, se debe cumplir que:

- $E_K(N) = C$ debe ser único
- $D_K(C) = N$ debe ser único

Lógicamente, en este ejemplo tampoco se cumple que al descifrar el criptograma usando la clave privada se recupere el texto en claro, como se muestra en la figura 6.

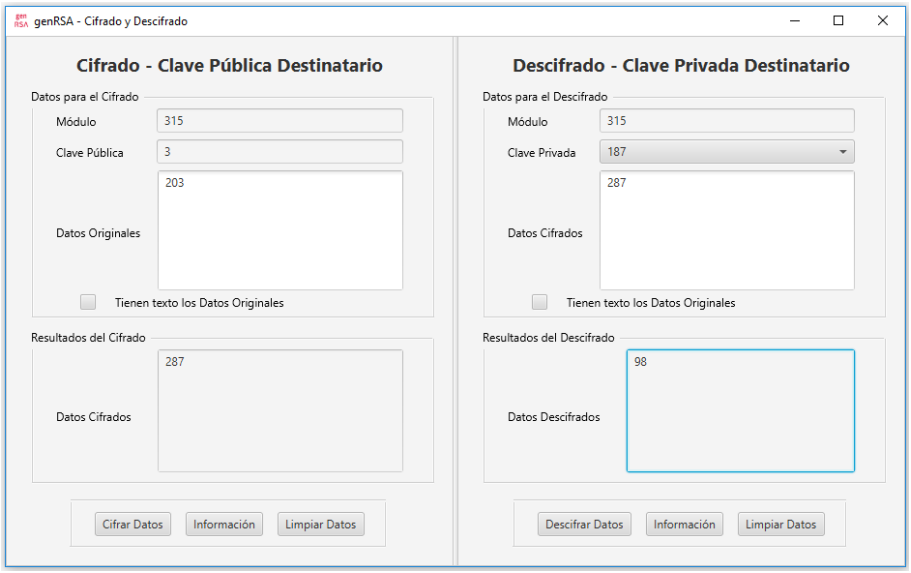

Figura 6. Con una clave RSA no válida no se recupera el texto en claro.

Lo que ha sucedido en este simple ejemplo nos indica que para poder cifrar con RSA y además descifrar, y esto último es lo más importante porque no vale de nada cifrar si después no puede recuperarse el texto en claro, los parámetros p y q deben ser primos.

Como ha podido comprobar, nada nos ha impedido generar una clave siguiendo los pasos del algoritmo RSA pero sin usar primos en los parámetros p y q. Ni mucho menos se nos ha impedido realizar una cifra sobre un número secreto N. Pero, por un lado, hemos incumplido la regla de que el resultado de la cifra debe ser único y, por otro, hemos sido incapaces de descifrar el criptograma C. Por lo tanto, carece de sentido el sistema de cifra creado.

Es claro que existe una explicación matemática de todo esto, pero no es el objetivo de este artículo ver este tema desde un punto de vista formal y matemático. Si lo desea, encontrará una explicación gráfica más ortodoxa del porqué es necesario usar primos en la generación de claves RSA, leyendo las páginas 626, 627 y 628 del capítulo 14, Cifrado Asimétrico Exponencial, de mi Libro Electrónico de Seguridad Informática y Criptografía[6].

[6] Libro Electrónico de Seguridad Informática y Criptografía, Versión 4.1 de fecha 1 de marzo de 2006, Jorge Ramió A. http://www.criptored.upm.es/guiateoria/gt_m001a.htm

4. Otros ejemplos de claves RSA generadas con OpenSSL que no son válidas

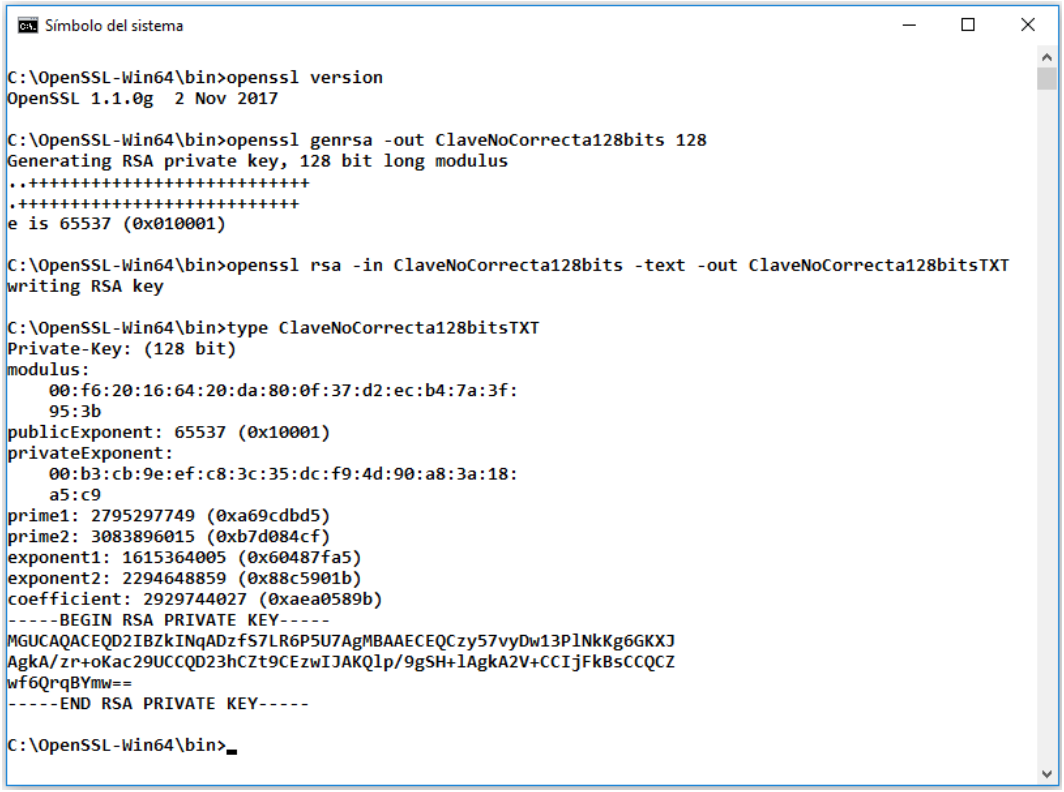

Figura 7. Clave RSA de 128 bits generada con OpenSSL donde *prime1* y *prime2* no son primos.

Comprobamos con msieve153[7] que *prime1* y *prime2* no son números primos.

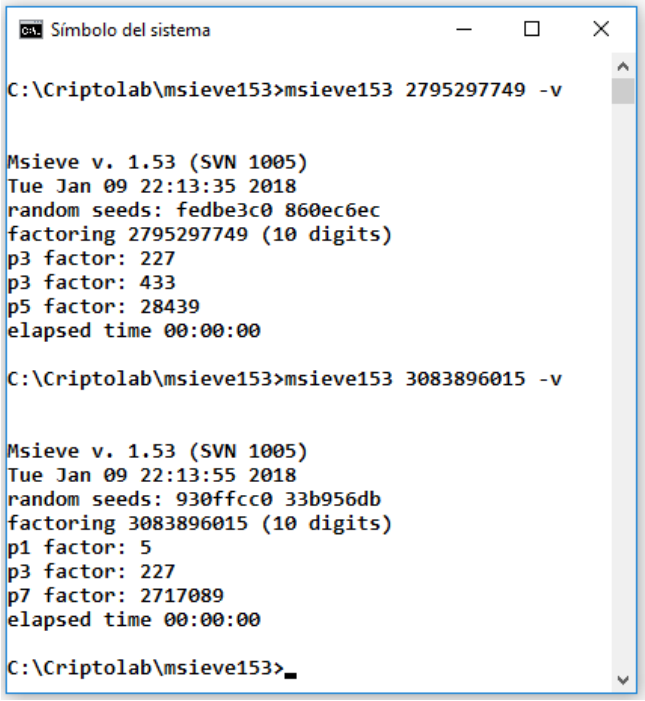

Figura 8. Los valores p y q entregados por OpenSSL no son primos.

[7] Msieve153, Sourceforge. https://sourceforge.net/projects/msieve/

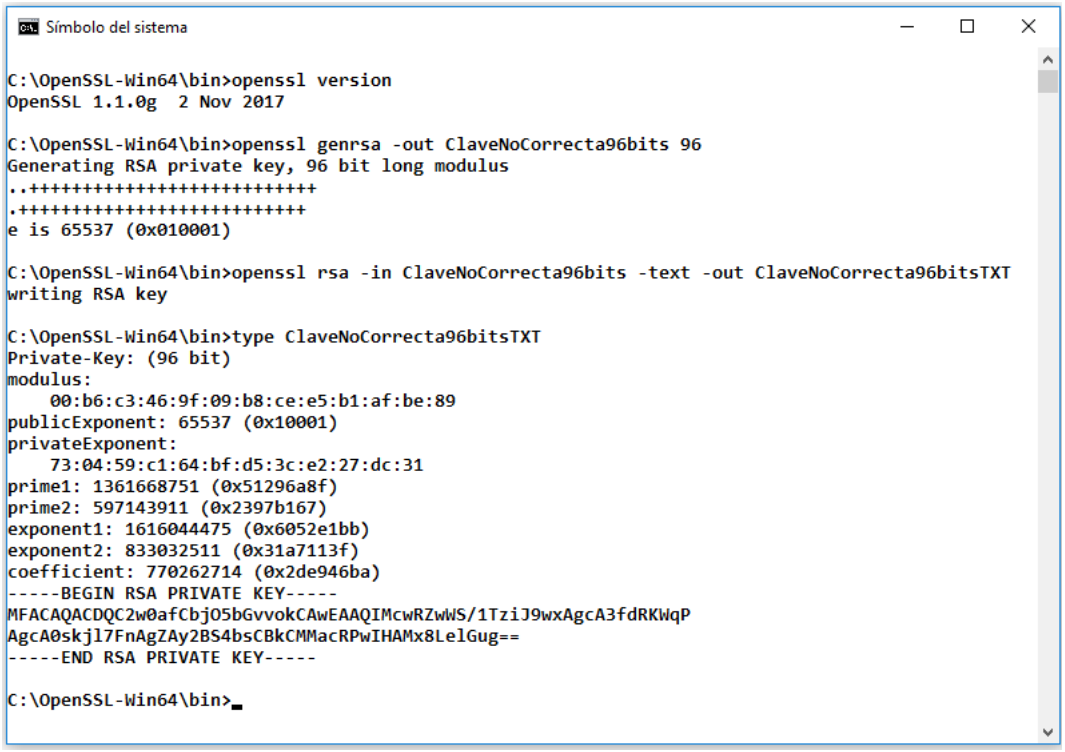

Figura 9. Clave de 96 bits generada con OpenSSL, con p primo, pero donde q no es primo y el módulo n y clave privada d son incorrectas.

Generamos con genRSA v2.1 la clave anterior, supuestamente de 96 bits, con los valores p y q que nos ha entregado OpenSSL, obteniendo la clave que se muestra en la figura 10.

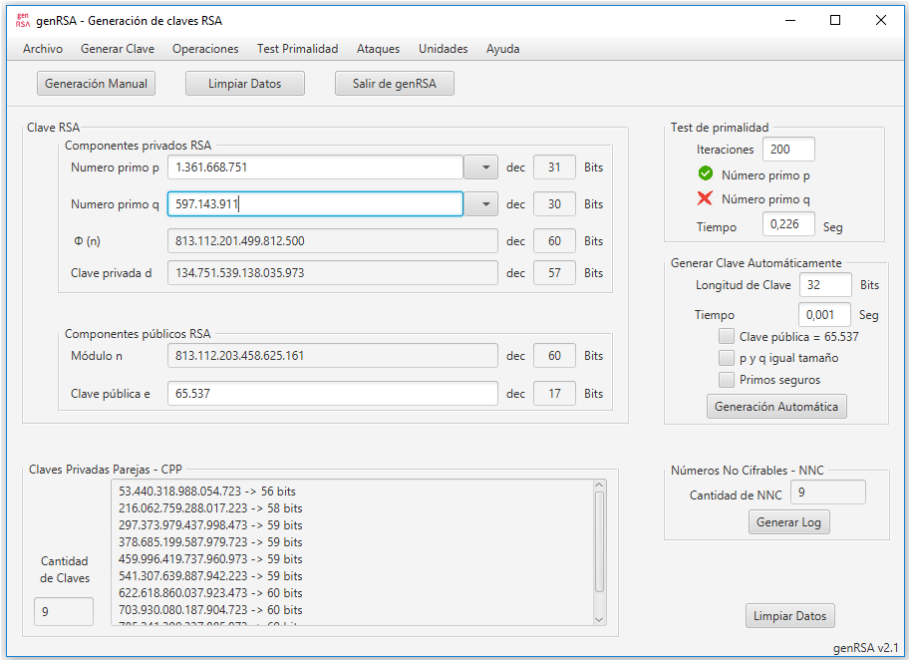

Figura 10. La clave anterior generada por OpenSSL, pero ahora través de genRSA v2.1.

La clave solicitada a OpenSSL era de 96 bits, pero nos ha generado un par de valores p y q, de 31 y 30 bits, que al multiplicarlos nos entrega una clave con un módulo de 60 bits.

En la clave generada por OpenSSL, comparada con la que genera genRSA v2.1, podemos apreciar los siguientes datos no correctos:

- p = 1.361.668.751. Es un primo de 31 bits, pero debería tener 48 bits.
- q = 597.143.911. No es primo (7x13x 6562021), tiene de 30 bits y debería tener 48 bits.
- pxq = 1.361.668.751 x 597.143.911 = 813.112.203.458.625.161
  Pero OpenSSL nos indica que n = 0xb6c3469f09b8cee5b1afbe89 = 156.562.345.821.696.301.019.941.551.753 (no es correcto)
- $\phi(n)$ = 1.361.668.750 x 597.143.910 = 813.112.201.499.812.500
- d = inv (e, $\phi(n)$) = inv (65.537, 813.112.201.499.812.500) = 134.751.539.138.035.973
  Pero OpenSSL nos indica que d = 0x730459c164bfd53ce227dc31 = 35.596.035.690.831.478.626.229.673.009 (no es correcto).

No obstante, en la generación de claves reales de 1.024, 2.048 o más bits, con esta última versión de OpenSSL para Windows 64 no se ha logrado encontrar estas anomalías, observando valores de p, q, n, e y d totalmente correctos, como se aprecia en las figuras 11a y 11b y se comprueba en la figura 12 con genRSA v2.1 para una clave de 2.048 bits.

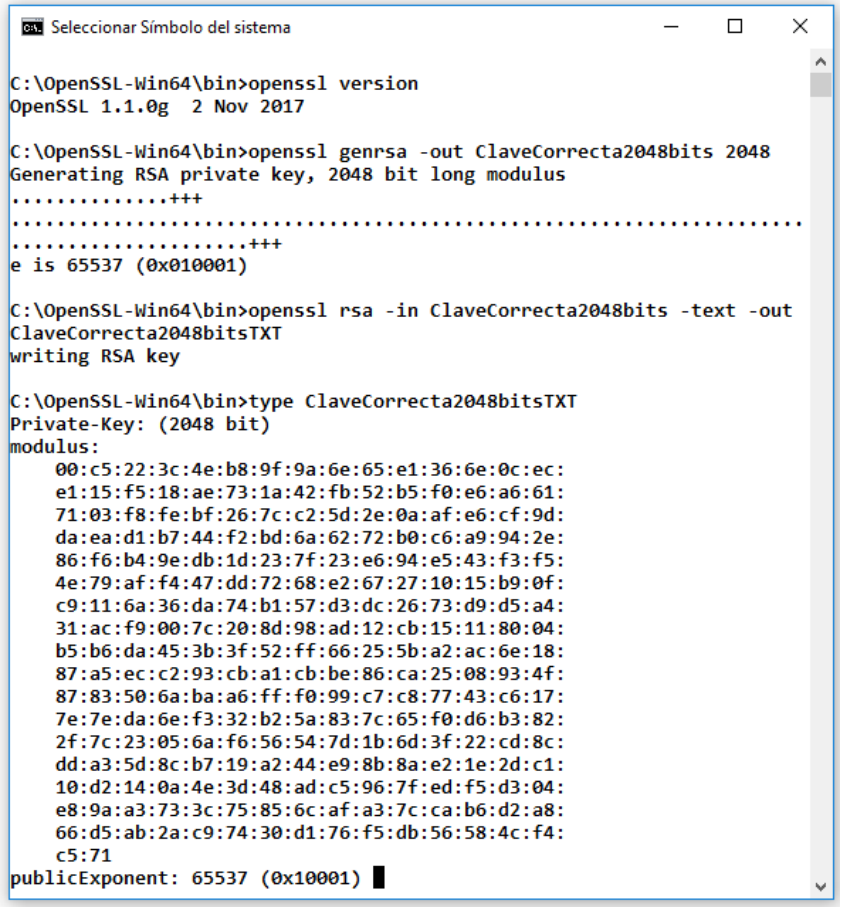

Figura 11a. Clave de 2.048 bits generada con OpenSSL: módulo y clave pública.

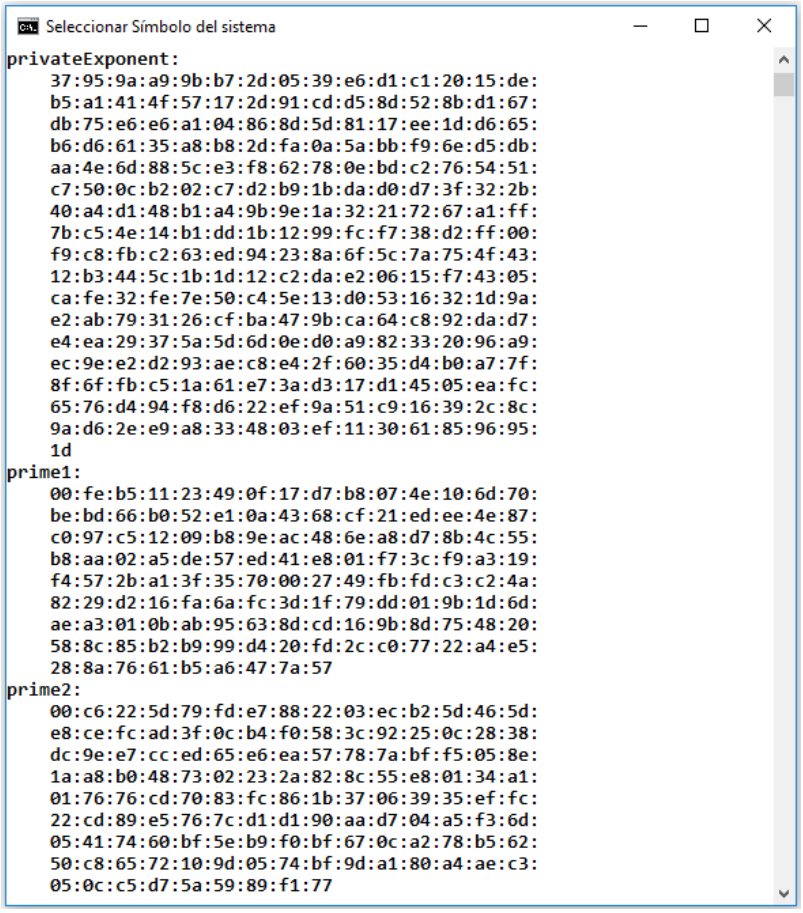

Figura 11a. Clave de 2.048 bits generada con OpenSSL: clave privada, primos p y q.

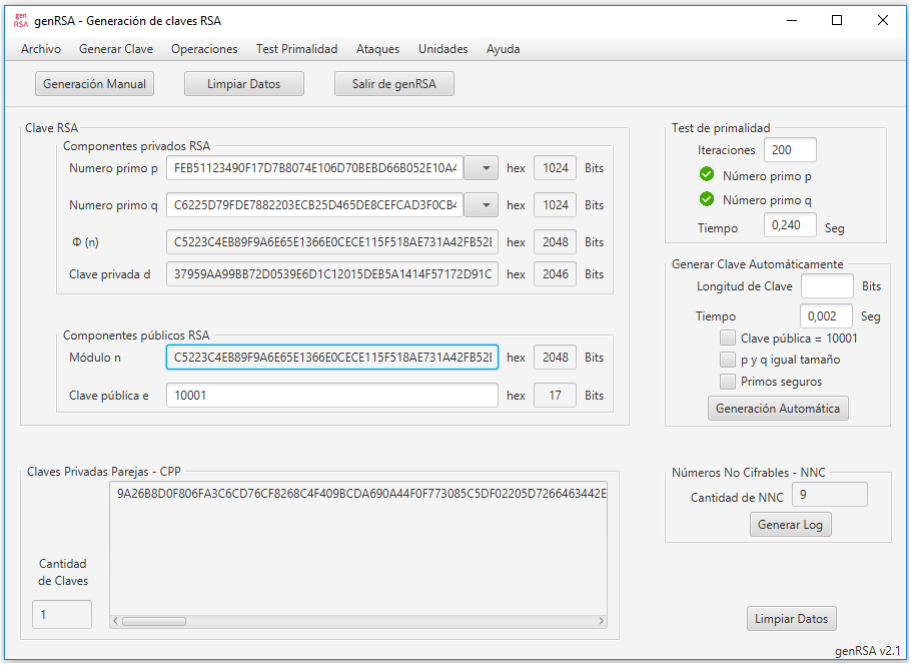

Figura 12. Clave de 2.048 bits generada con genRSA v2.1 con los datos de OpenSSL.

Se observa que con genRSA v2.1 se obtienen los mismos valores del cuerpo de cifra n y de la clave privada d y, obviamente, los valores p y q entregados por OpenSSL son primos.

5. Breves comentarios

Como se ha podido comprobar, al menos para claves RSA de hasta 128 bits, Win64 OpenSSL v 1.1.0g no las genera correctamente. En algunos casos, nos entrega como primos p y q dos números que no lo son. En otros, los valores de p y/o q sí son primos, aunque no del tamaño solicitado en el comando, pero tanto el módulo n como la clave privada d no son los valores correctos para los primos generados por ese programa.

Se supone que estas anomalías de Win64 OpenSSL v 1.1.0g en la generación de claves RSA para tamaños pequeños del módulo no se manifiestan en la generación de claves reales de 1.024, 2.048 o más bits, si bien esto último no se ha podido comprobar.

Es interesante destacar que también se han generado diferentes claves con Win32 OpenSSL v 1.1.0g, pero en este caso sólo se han obtenido claves válidas.

No se había apreciado este comportamiento en versiones anteriores de OpenSSL.

Aunque se permita generar una clave RSA sin usar números primos en los parámetros p y q, resulta claro que la clave generada no es válida y que el sistema de cifra no funciona al ser imposible recuperar el texto en claro que se ha cifrado e incumplir el principio de que el resultado la cifra debe ser único.

Si bien sería interesante que se usasen primos seguros en la generación de claves RSA, es decir, aquellos en los que se cumple que q = 2p+1, siendo p y q primos (e.g. p = 23, q = 47), en realidad OpenSSL no lo hace. ¿Por qué no lo hace? La respuesta a esta pregunta, así como la importancia o no del uso de primos seguros en la criptografía asimétrica, se dejará para un próximo artículo.

****

Artículo publicado el 11 de enero de 2018

Nota al 18/01/2018

D. Daniel Franganillo Corrales (https://www.linkedin.com/in/dfranganillo/) aporta en la fecha indicada, mediante post enviado a la nota que el autor publica en LinkedIn sobre este artículo, el siguiente enlace, que puede ser de interés en el tema que aquí se trata.
Se agradece su colaboración.
https://github.com/openssl/openssl/commit/cac19d19e7d6f252ff9aea60d85e0c0fd71a117f