Factorización de enteros grandes para ataque a RSA

José Antonio González Cervera

Pablo Sánchez Robles

Defiende tu derecho a pensar, porque incluso pensar de manera errónea es mejor que no pensar.

Hipatia de Alejandría.

Índice de contenido

Introducción:

En este trabajo vamos a tratar el tema de la factorización de enteros grandes para romper claves RSA. Como hemos visto en clase RSA es un algoritmo criptográfico asimétrico descrito por Ron Rivest, Adi Shamir y Leonard Adleman en 1977, no fue hasta 1983 cuando fue patentado por el instituto tecnológico de *Massachusetts.*

[1]La idea básica del algoritmo es la generación de dos claves, una privada y una pública por cada "usuario" de una comunicación, es decir, el emisor tendrá un par de llaves y el receptor igual. Para establecer una comunicación segura entre ambos ha de cifrarse el paquete utilizando dichas claves, en el caso de que A quiera enviarle un mensaje a B, A debe cifrarlo con la clave pública de B para que así sólo este pueda descifrarlo, o eso es lo que se pretende, como en todo sistema de seguridad siempre hay posibilidades de "romper" las claves y es precisamente de esto de lo que va a tratar este trabajo.

Ejemplo de ataque por factorización:

[1]Si suponemos un numero primo p = 61, un numero primo q = 53, un n = pxq, un exponente público e = 17 y un exponente privado d = 2753, para cifrar un mensaje llamémosle "m" solo tenemos que elevar dicho mensaje a "e" y hacerle el módulo con "n" y para descifrar un texto cifrado "c" sólo tendríamos que elevarlo a "d" y de nuevo hacer el módulo con "n". Con estos datos vamos a cifrar un mensaje m = 123:

Cifrar(m) = (123^17) % 3233 = 855.
Descifrar(cifrar(m)) = (855^2753) % 3233 = 123.

Como vemos el proceso es sencillo e incluso fácil de romper con números pequeños como los del ejemplo pero si los factores primos "p" y "q" son números de 100 dígitos decimales la cosa se pone másdifícil.

El método que se utiliza para romper RSA es la factorización de enteros grandes puesto que teniendo una clave pública podemos conocer el valor de N y para poder descifrar un mensaje debemos conocer los valores "p" y "q" que son sus factores primos y a través de ellos calcular el exponente "e" y el exponente "d".

1ª parte: Comparativa de algoritmos de factorización

En esta primera parte de nuestro trabajo vamos a realizar una batería de pruebas de distintos software de factorización para poder seleccionar de entre ellos el más eficaz con el objetivo de poder llegar a factorizar números más grandes en las siguientes fases de éste. Todas las pruebas las vamos a llevar a cabo en la misma máquina con el fin de que los resultados sean objetivos para la comparación.

Los programas que probaremos en esta fase son:

1. **Dismat**: Son una series de aplicaciones integradas en el mismo software, desarrolladas en Visual C++ y Visual Basic para cálculos de matemática discreta. Es usado en criptografía e incluye un apartado dedicado para RSA, está programado para Windows. [2]
2. **ExpoCrip**: Este software de laboratorio corresponde a un proyecto docente de la asignatura de **Seguridad Informática** del Departamento de Lenguajes, Proyectos y Sistemas Informáticos de la Escuela Universitaria de Informática de la Universidad Politécnica de Madrid (España) que lleva por nombre **Criptolab** y contempla algoritmos de cifra clásica, de cifra moderna, teoría de la información, aritmética modular, libros electrónicos, cuaderno de prácticas, tutoriales, protocolos, interfaces, etc.[3]
3. **Factor:** Desarrollado por Shamus Software en Dublín y escrito en C, este programa sólo realiza tareas de factorización de enteros, a diferencia de los dos anteriores se trata de un software de propósito algo más específico. Utiliza la librería MIRACL.[4]
4. **Msieve**:Es una biblioteca escrita en C con un conjunto de algoritmos para factorizar enteros grandes. Contiene una implementación de los algoritmos SIQS y GNFS, este último ha ayudado a completar algunas de las factorizaciones de números enteros más grandes que se han logrado.[5]
5. **Yafu:** Utiliza los algoritmos modernos más potentes para factorización de enteros de forma automatizada. Combina los algoritmos de factorización en una metodología inteligente y adaptativa que reduce al mínimo el tiempo para encontrar la solución al problema. Utiliza programación paralela para reducir este tiempo al mínimo. Incluye los algoritmos SNFS, GNFS, SIQS y ECM. Esta aplicación es más que un simple software de factorización, al igual que matlab o máxima es un entorno completo para el desarrollo de operaciones matemáticas. Se puede utilizar por línea de comandos o por un entorno gráfico similar a matlab. Escrito en C y ensamblador.[6]

Claves utilizadas

Los números a factorizar junto con sus factores primos son los siguientes:

Claves 40 bits
p = 863363
q = 863633
N = 745628777779

Claves 60 bits
p = 1019608241
q = 1019608567
N = 1039601297507400647

Claves 80 bits
p = 635894165447
q = 735894165691
N = 467950806349394785078877

Claves 100 bits
p = 706743837976733
q = 906743837976803
N = 640835620113478698311707724599

Claves 120 bits
p = 717061215212703167
q = 873002668389159199
N = 625996354279063020429191735094483233

Claves 140 bits
p = 747349372219033939501
q = 914465747033887393087
N = 683425401961585641053843852818264563629587

Claves 160 bits
p = 1032801399915106717648397
q = 1137217992048983267245057
N = 1174520334196836618892931660701885349279762223629

Claves 180 bits
p = 68204740314432502994914497 1
q = 76202333014907712686866063 3
N = 5197360334635686971731842490802114593074015743176266 43

Claves 200 bits
p = 99931989675439498895599715095 1
q = 82546612760950432889588142986 9
N = 82490472541698010478530377840961763685321100704467211315541 9

Claves 220 bits
p = 1166284760175921892620787274935733
q = 943833492723449343710263660377773
N = 1100778618706970838560079625932945475197558309191731656577076662609

Claves 240 bits
p = 779610602693034312947320956725626279
q = 118517497462350554326042149816815 4131
N = 923974976262932804192153961009898107223408424387084071821632234376008549

Claves 260 bits
p = 7118078223261204824909960253846656106 79
q = 7118078223261204824909960253846656111 53
N = 5066703759246539048093645999795911265669746357290399599998728500213794983028 87

Claves 270 bits

p =
3151463562847730009890180697625583199357 9
q =
4151463562847730009890180697625583199512 9
N =
1308318615080463832777443504761951980680992966879001086023834309636343145087276691

Claves 280 bits
p =
8463204278252073841050203168005725587512 87
q =
8463204278252073841050203168005725587534 47
N =
7162582665542420610395948414981222213343043821627647745989767180032561120556269362 89

Claves 290 bits
p =
3846027270057472244776028248249384518639780 1
q =
3192966548227994783046789356134765985419523 9
N =
1228023641686614506803711487058229863798799021407859427962409043437460639121928174269439

Claves 300 bits
p =
1272666058816221798652784238430023761483936087
q =
1039700555253300692961657301597922362025117917
N =
1323191608003255641669328315371653554665448894663133517665984261783314424345058797466570779

Claves 310 bits
p =
3664265039568238396621089064739914852090221571 3
q =
3318594868835830822195272636938706887412037477 3
N =
1216021115836567853959100863167537312195082359382252769474551560231654117740732907133649408149

Claves 320 bits
p =
8916374780707142295904992381461021673555642704 21
q =
8767131735639624163288722025970535770981111445 29
N =
7817103230679438172226788748257863558705285159396120575306169425679866353096752577846231706767 09

Claves 330 bits
p =
4162339038286915626983763242831101522619114825 1059
q =
3775629520559720010552160720055262370014304328 2777
N =
1571545014753423332142082797842317513426613552450591605587659937621587949010052766534230561526710843

Claves 340 bits
p =
1360662457580417336026305487538917635213330438324301
q =
1308651722472256920918852797531330690350959586207768 59
N =
1780633268815947362896284362635864222071593792506739372794804348218646360752672392410922647410798150559

Claves 350 bits
p =
2817383523690149458330910214996966124382206785374734 7
q =
3961161018561798388593101699141773554111417308350112 9

N = 1116010978837970106859209762030802459827585936963943981574593380446834594132781660436112563751541355254763

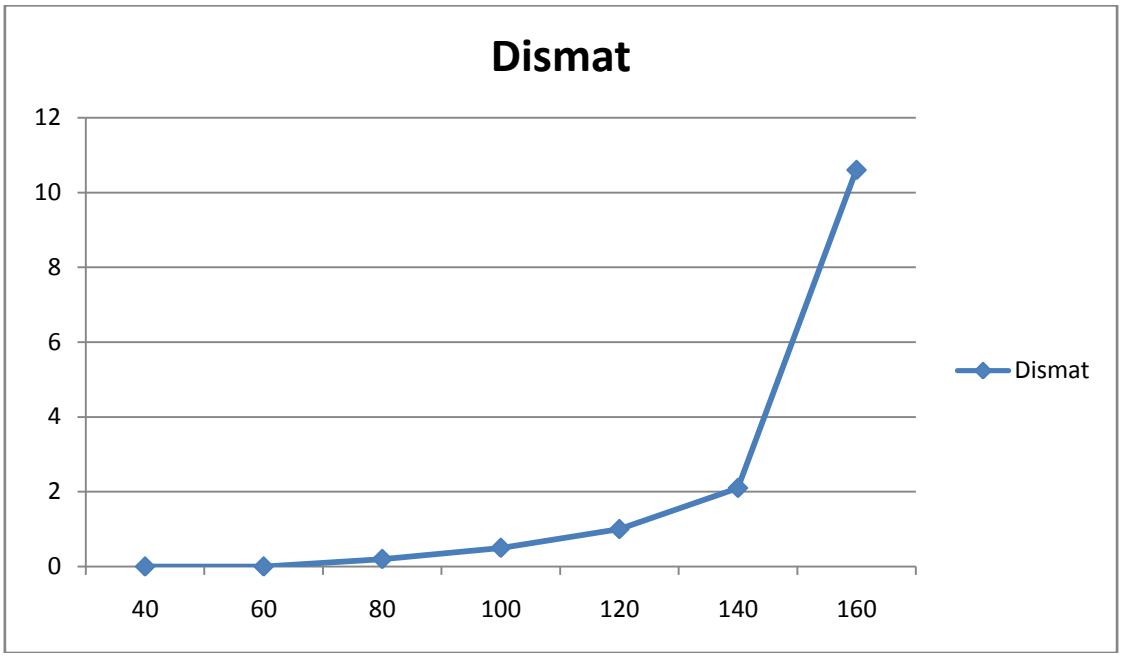
Los valores de las claves aquí mostrados son los que vamos a utilizar para todos los experimentos del trabajo, para este primer apartado haremos una prueba con claves de hasta 240 bits, el resto las utilizaremos en el siguiente apartado.

Prueba de distintos algoritmos

Los datos de salida de los programas anteriormente mencionados para un procesador QuadCore Q6600 son:

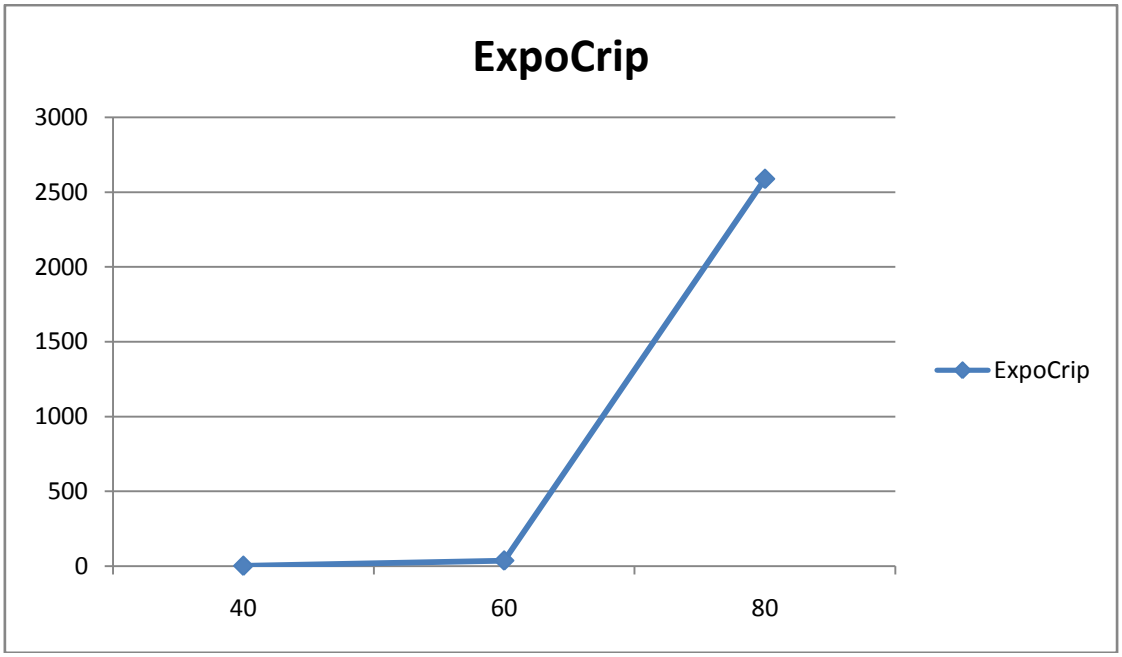
Dismat:

Tamaño de la clave(bits)	Tiempo(segundos)
40	0
60	0
80	0,2
100	0,5
120	1
140	2,1
160	10,6
180	NULL



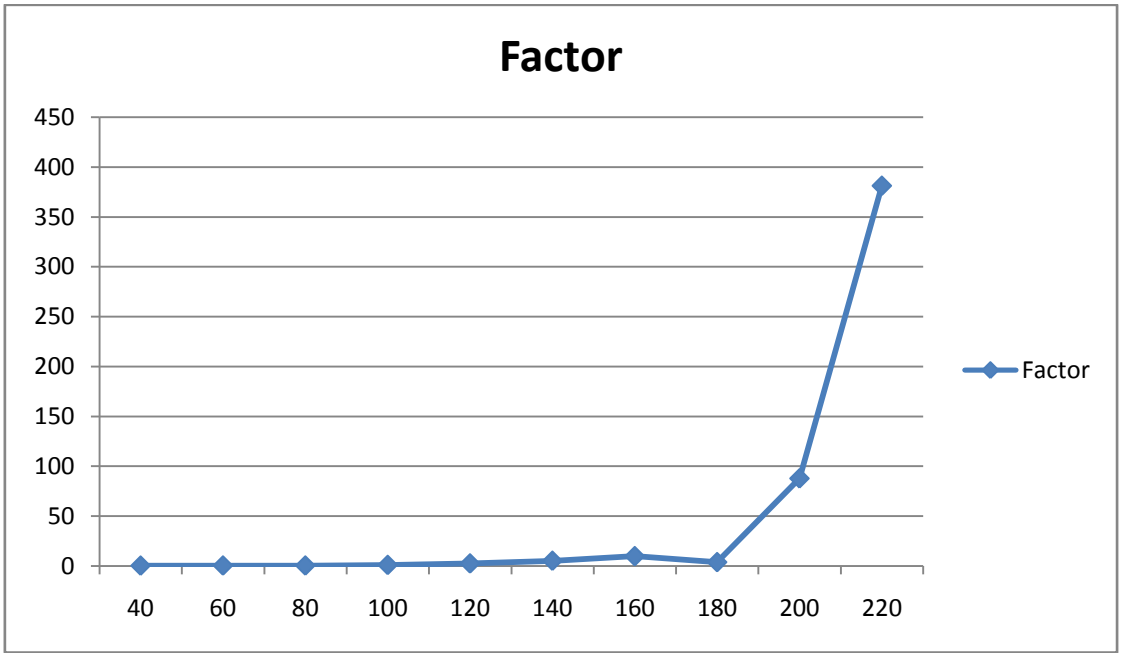
ExpoCrip

Tamaño de la clave(bits)	Tiempo(segundos)
40	2
60	37
80	2589
100	NULL



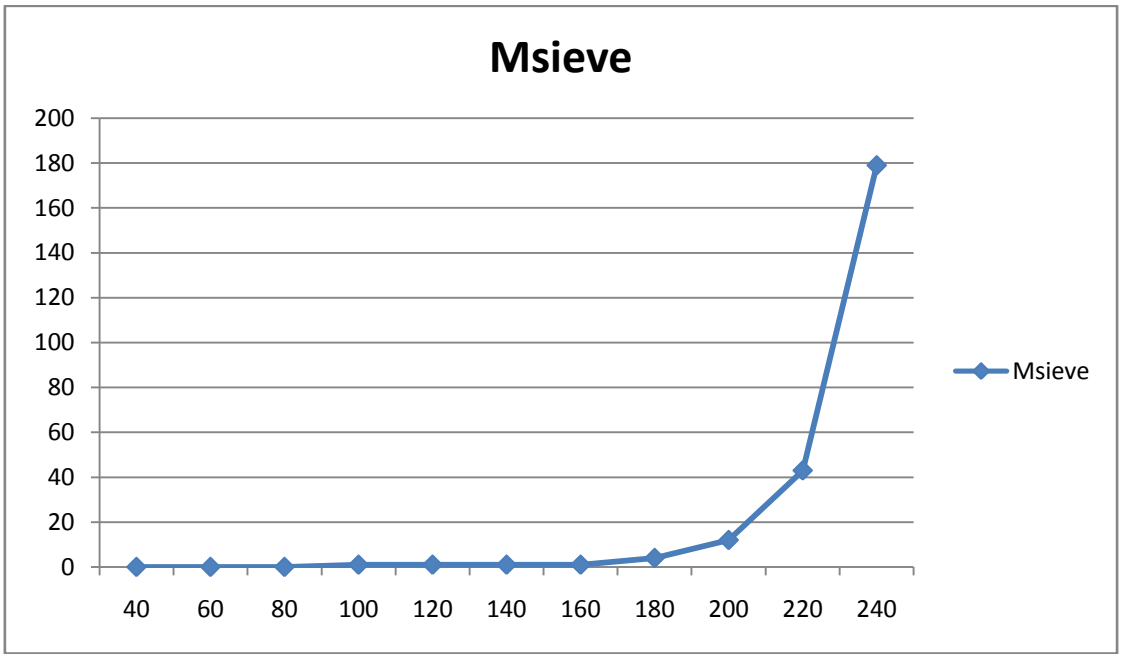
Factor:

Tamaño de la clave(bits)	Tiempo(segundos)
40	0,2
60	0,3
80	0,3
100	0,9
120	2,4
140	5,1
160	9,7
180	3,7
200	87,7
220	381,3



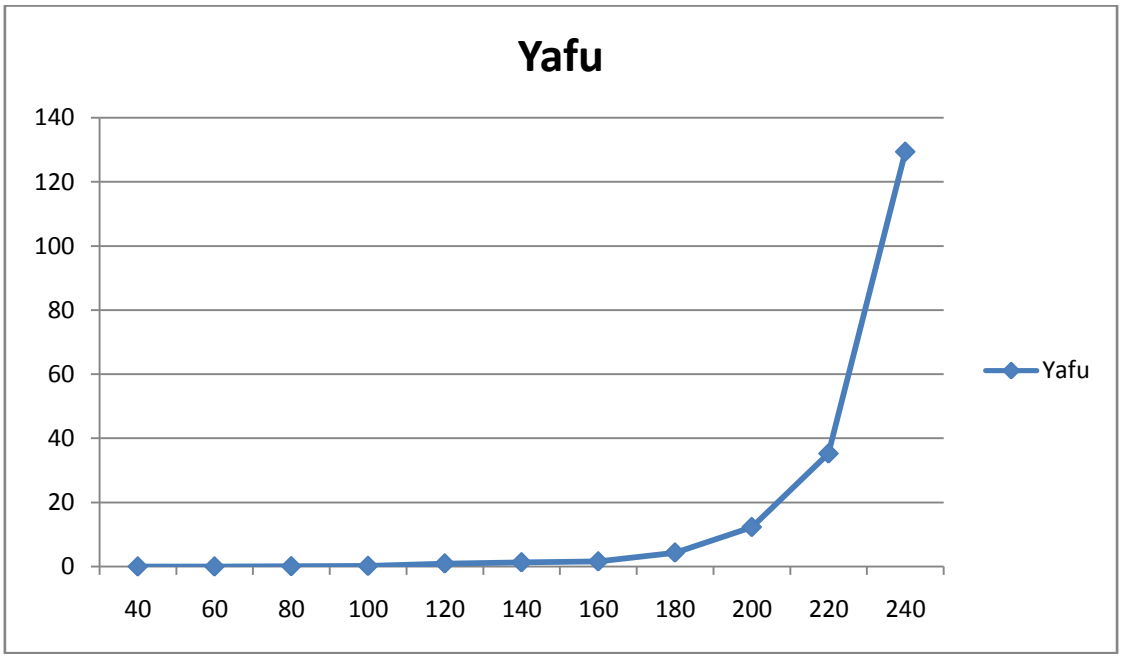
Msieve

Tamaño de la clave(bits)	Tiempo(segundos)
40	0
60	0
80	0
100	1
120	1
140	1
160	1
180	4
200	12
220	43
240	179



Yafu:

Tamaño de la clave(bits)	Tiempo(segundos)
40	0
60	0
80	0,09
100	0,18
120	0,94
140	1,3
160	1,59
180	4,34
200	12,3
220	35,23
240	129,32

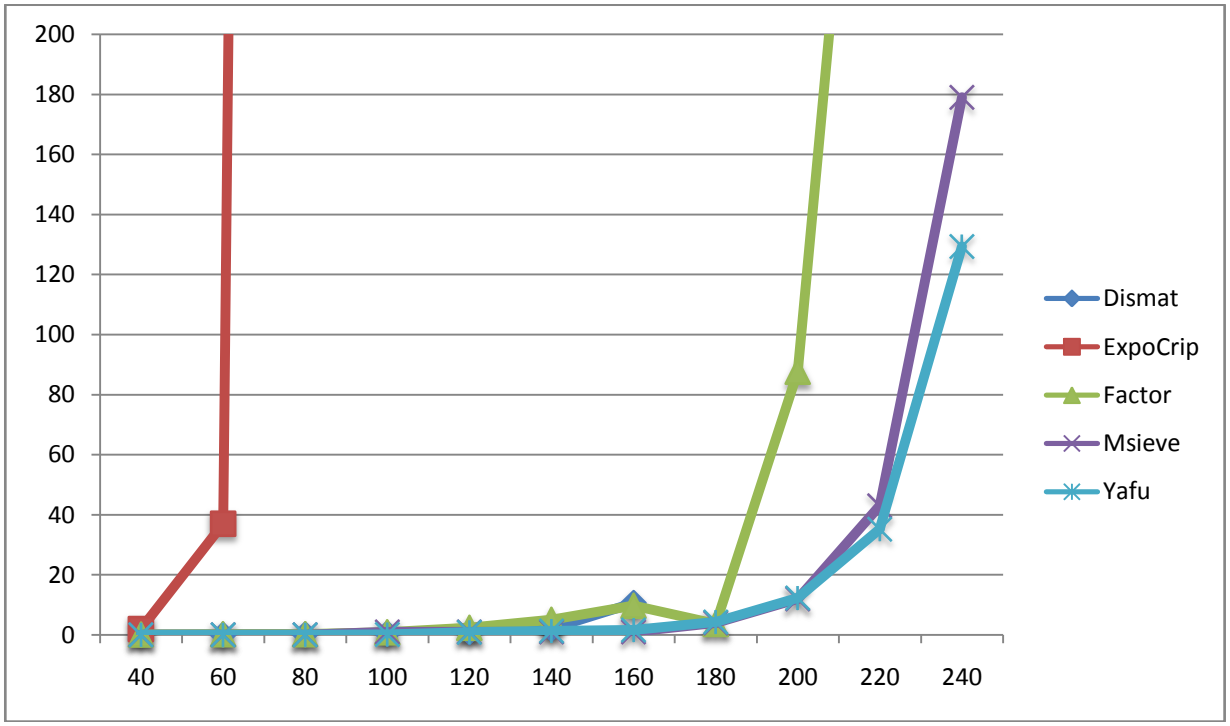


Analizando los resultados, vemos que hay grandes disparidades entre los algoritmos. Por ejemplo, ExpoCrip se dispara factorizando 80 bits a 2589 segundos, mientras que msieve o Yafu logran factorizar dicha cantidad de bits en menos de un segundo.

También nos podemos percatar de que puede haber cosas “ilógicas” como en el caso de Factor, que en factorizar 160 bits tarda 9.7 segundos, y para 180 tarda 3.7 segundos. La

explicación de esto recae en dos puntos clave: uno es la forma de tomar las medidas de tiempo, que se basa en el tiempo transcurrido. Se ha intentado minimizar la carga de trabajo en el sistema, para que haya menos procesos y de esta forma, los tiempos no se vean afectados por que el procesador tenga que atender otras tareas. Sin embargo, este factor debe tenerse en cuenta, ya que por mucho que se intente minimizar, siempre quedarán tareas a realizar (incluso del propio sistema operativo). El otro punto, es que quizás para los números utilizados en la factorización, se han incluido algunas técnicas más avanzadas para ayudar en la factorización, de forma que para el número de 180 bits le ha sido más favorable esa "ayuda" que para el de 160. No podemos hablar con toda seguridad por qué desconocemos el código fuente escrito para ExpoCrip, aunque se puede intuir este hecho.

La siguiente gráfica muestra todos los programas utilizados, comparándolos, para obtener una mejor perspectiva de cuáles son los mejores programas, de cara a realizar el segundo apartado de este trabajo:



De todos los programas puestos a prueba vemos claramente que Msieve y Yafu son muy superiores a los demás, tanto en tiempos de ejecución como en tamaños de clave utilizables, Factor también es capaz de factorizar claves algo más grandes que los demás pero sus tiempos de factorización son claramente peores que Msieve y Yafu por lo que utilizaremos éstos dos últimos para la siguiente fase.

2º parte: Comparativa de los mejores algoritmos entre diferentes máquinas

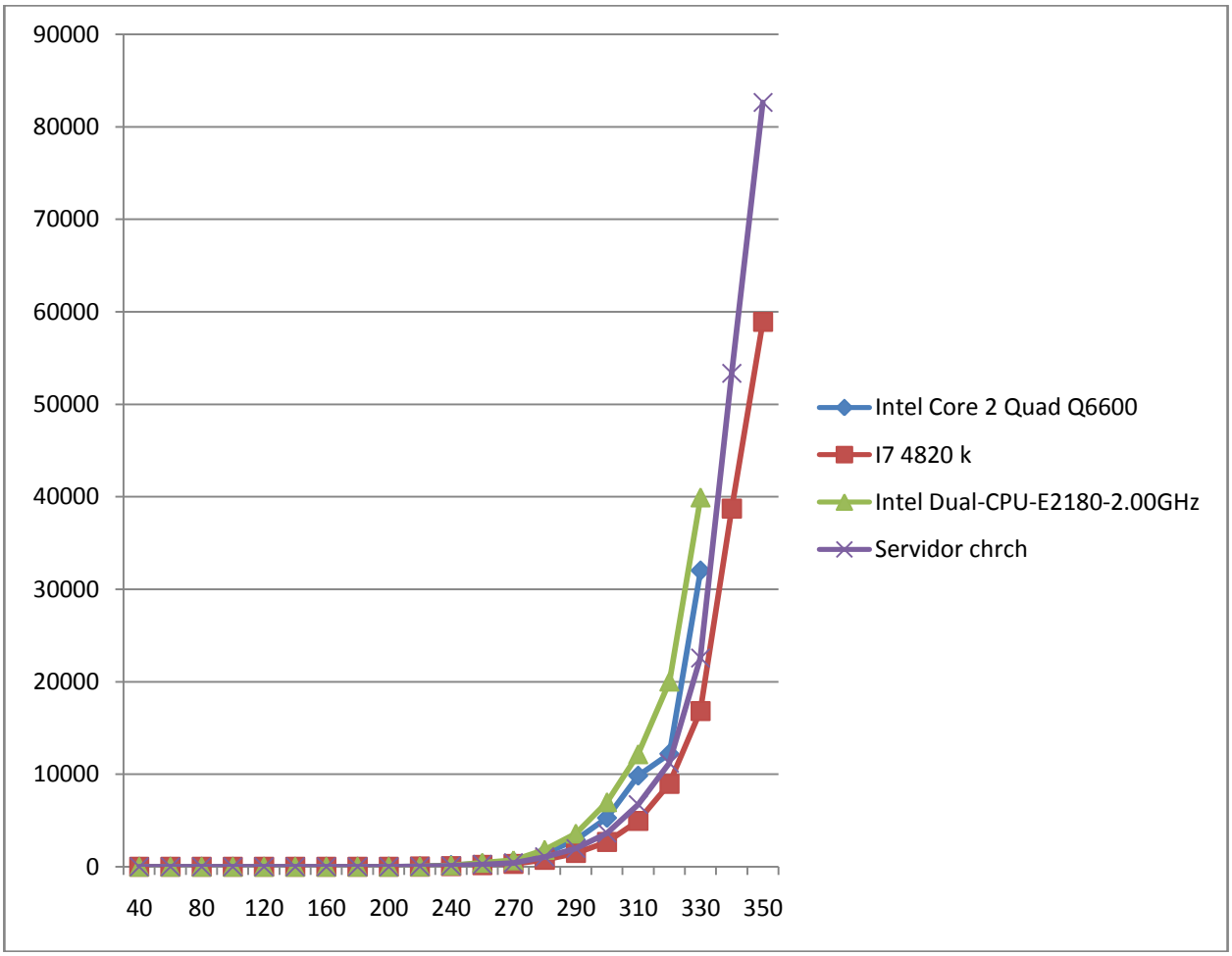
En esta parte, se han cogido los dos mejores algoritmos de factorizado de la parte anterior (que han sido msieve y Yafu) y se comprobarán las diferencias hardware entre procesadores que pertenecen a distintas etapas de la evolución de las prestaciones de un equipo.

Aclaración: Los resultados mostrados se corresponden a la ejecución de los programas con un solo hilo. Se pretende de esta forma comparar las velocidades y las mejoras en las arquitecturas generales bajo un único núcleo de procesamiento. Además, las máquinas no tienen un mismo número de núcleos, por lo que la forma más justa de comparar las prestaciones, es con un solo núcleo.

Primero, analicemos los resultados obtenidos para las pruebas usando el programa msieve:

Bits factorizados	Tiempos (en segundos)			
	Intel Core 2 Quad Q6600	I7 4820k	Intel Dual-CPU-E2180-2.00GHz	Servidor church
40	0	0	0	0
60	0	0	0	0
80	0	0	0	0
100	1	0	1	0
120	1	1	1	0
140	1	1	1	0
160	1	1	1	0
180	4	1	3	1
200	12	4	2	4
220	43	16	35	17
240	179	64	148	85
260	342	181	425	251
270	564	300	700	420
280	1437	756	1826	1066
290	2907	1476	3587	1963
300	5280	2667	6983	3619
310	9840	4952	12168	6705
320	12219	8946	20025	11192
330	32031	16830	39941	22556
340		38699		53329
350		58882		82611
360				151034

Ahora, mostremos los resultados en una gráfica, para poder apreciar de una mejor forma la diferencia. Nótese que hay algunas factorizaciones que no han sido realizadas (las de 340,350 y 360 bits). Esto es así porque la capacidad de cómputo de los procesadores más lentos harían que la prueba durase demasiado, y de todas formas, la diferencia ya se puede apreciar a los niveles que se han alcanzado en las pruebas de este documento.



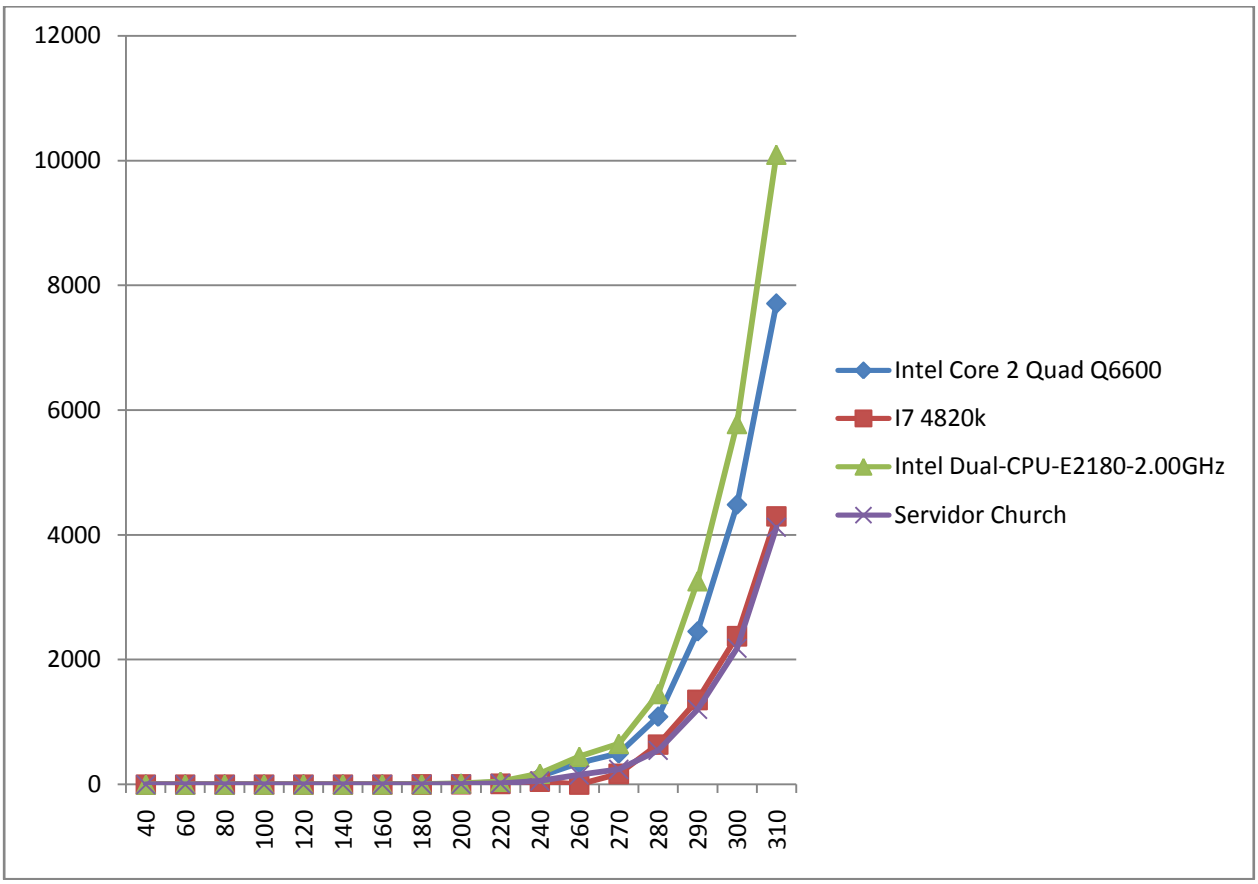
Aclaración: Vemos que los procesadores más rápidos son los del servidor Church y el I7 4280k, siendo este último más rápido. Vemos también como prácticamente el procesador Intel Dual-CPU E2180 tarda incluso algo más de tiempo en factorizar 330 bits que lo que tarda el i7 en factorizar 340.

Pasamos a ver las medidas de tiempo tomadas para la factorización con el programa Yafu:

Este programa (también ejecutado con un solo thread) tiene unas mejoras significativas respecto a msieve, de hecho, el algoritmo que utiliza de factorización parece mejor. La contra es que el algoritmo puede llegar a factorizar tan solo hasta los 310 bits.

Bits factorizados	Tiempos (en segundos)			
	Intel Core 2 Quad Q6600	I7 4820k	Intel Dual-CPU-E2180-2.00GHz	Servidor church
40	0	0	0	0
60	0	0	0,1719	0
80	0,09	0,03	0,1719	0,0645
100	0,18	0,06	0,3906	0,1576
120	0,94	0,31	0,9844	0,4079
140	1,3	0,44	0,5938	0,4986
160	1,59	0,53	1,5938	0,6470
180	4,34	1,61	5,5938	2,2089
200	12,3	4,1	15,0781	6,2112
220	35,23	11,95	45,8125	16,6109
240	129,32	43,76	174,3281	58,4427
260	343,3299	139,5678	445,2969	152,9995
270	499,3059	170,34	652,6875	242,7461
280	1087,2038	640,56	1451,7344	545,8356
290	2452,6341	1355,6	3258,375	1203,5022
300	4483,7401	2375,08	5778,0156	2182,0063
310	7705,3019	4295,42	10092,4844	4118,4772

Al igual que antes, reflejamos los resultados en una gráfica para hacernos una mejor idea.



Aclaración: Los tiempos del servidor Church y del procesador i7 son prácticamente idénticos, ya que la gráfica están casi superpuestas. Aunque en los dos peores procesadores, los resultados siguen siendo coherentes con los de Msieve, ya que vuelve a ser el Dual-CPU E2180 el más lento (y con mucha diferencia respecto a los más rápidos).

Aclaración: En la presentación expuesta en clase la factorización con yafu de la clave de 260 bits tuvo una salida en 0,2 segundos lo cual se pudo producir por el fichero de logs de yafu que almacena información de factorizaciones anteriores para agilizar las búsquedas de los primos p y q, repetimos la prueba limpiando los datos del fichero log y el tiempo que mostró fue de 139,5678 segundos lo que ya si se ajusta a la evolución razonable de los tiempos con respecto a los bits a factorizar.

La necesidad del multi-thread

Los tiempos expuestos anteriormente hacían referencia a ejecuciones mononúcleo. Pero uno de los avances que nos ha dejado la tecnología en su avance, ha sido la posibilidad de permitir más de un hilo de ejecución (thread) en una misma máquina. Por tanto, en este apartado vamos a comparar la ganancia en el tiempo que se obtiene al utilizar arquitecturas y programas que permitan el multi-thread.

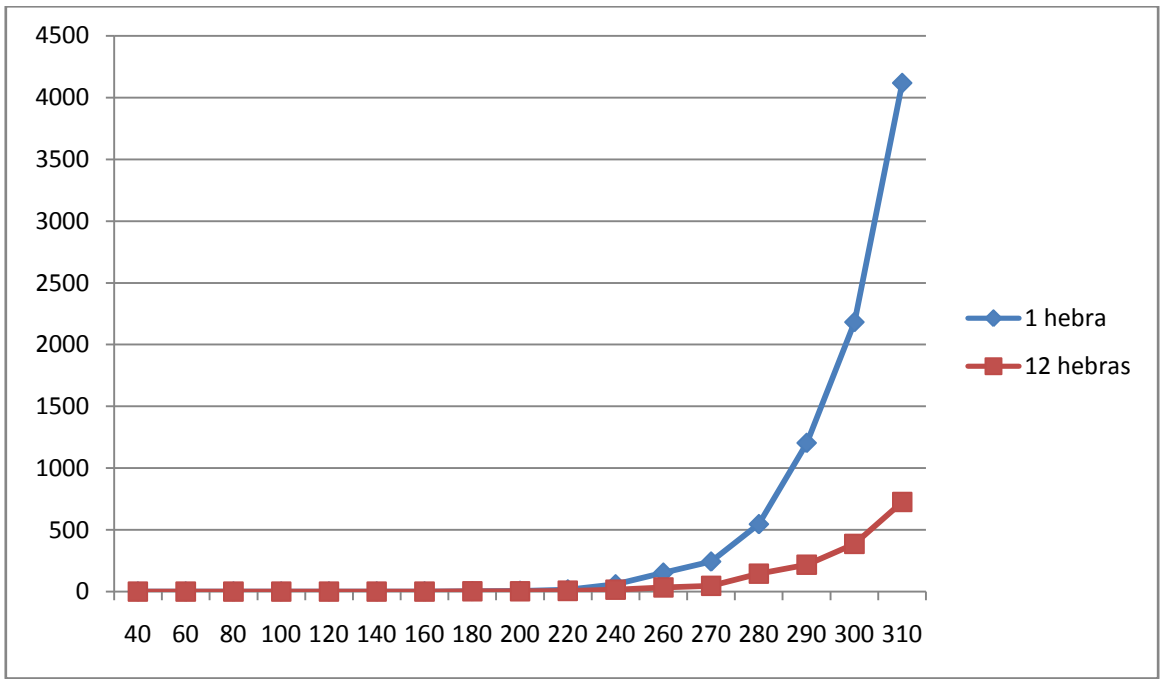
Para el experimento se ha usado el servidor Church. La siguiente tabla recoge los datos

de ejecución del programa Yafu para una ejecución de un thread y para otra con 12 threads:

Bits factorizados	Tiempos (en segundos)	
	1 hebra	12 hebras
40	0	0,0008
60	0	0,0546
80	0,0645	0,0464
100	0,1576	0,0762
120	0,4079	0,3484
140	0,4986	0,4278
160	0,6470	0,5064
180	2,2089	1,3813
200	6,2112	3,2340
220	16,6109	6,0857
240	58,4427	14,8600
260	152,9995	32,5691
270	242,7461	47,3071
280	545,8356	144,3071
290	1203,5022	218,2691
300	2182,0063	387,0829
310	4118,4772	725,8558

Analizando los resultados, nos percatamos de dos cosas. La primera, es que para factorizar números más grandes, la ejecución de 12 hilos tarda un tiempo notablemente mayor, haciendo que sea del orden de algo más de 5 veces más rápida la factorización. La otra cosa a tener en cuenta, es que para números pequeños, la ejecución no mejora demasiado, llegando a incluso en los números más bajos a ser más lenta. Esto es debido a que se necesita inicializar las 12 hebras, asignarles un trabajo, y reunir los resultados de cada hebra al final de su ejecución. Estas acciones hacen que el programa se tome más tiempo para “prepararse” antes de empezar a hacer cálculos, o para presentarlos al usuario.

La siguiente gráfica compara los tiempos de ambas ejecuciones, para ver de una forma más clara (si cabe) la diferencia entre la ejecución con y sin multi-thread:



Estimaciones de tiempos por ajuste de mínimos cuadrados

Vistos los resultados anteriores, ya nos hemos hecho una idea acerca de cómo ha evolucionado el hardware (incluso bajo un comportamiento mononúcleo) a lo largo del tiempo. Obviamente, el número a factorizar si se pretende atacar un sistema basado en RSA cada vez será de más dígitos conforme pasa el tiempo, provocado por este avance de la tecnología. Sin embargo, podríamos hacernos una idea de lo que tardaría cualquiera de estas máquinas en lograr sacar los primos necesarios para poder descifrar una comunicación capturada (por ejemplo, los datos de una cuenta bancaria).

Esta estimación la haremos mediante un ajuste de mínimos cuadrados, con un ajuste exponencial, ya que viendo los tiempos de las máquinas en las gráficas anteriores, nos percatamos que el tiempo necesario para factorizar un número crece de forma exponencial.

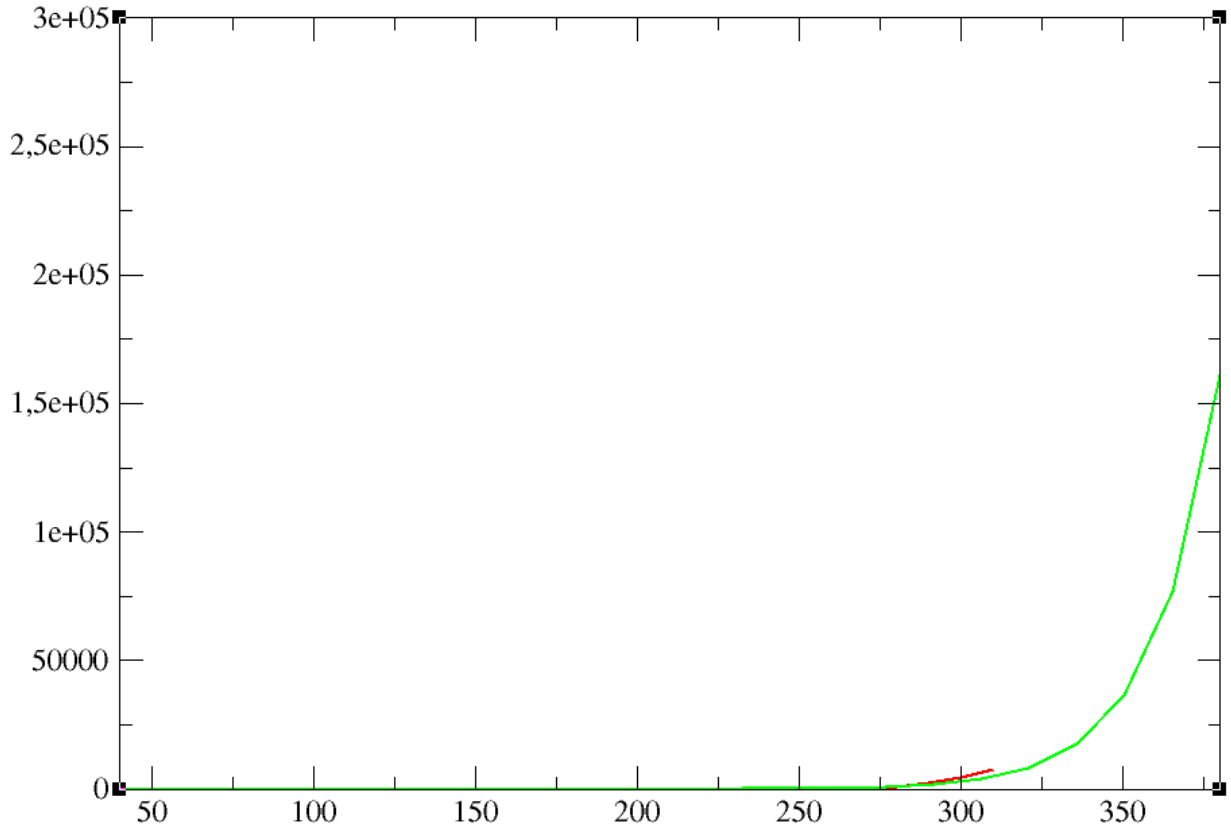
El tiempo estimado será:

$$Y = a*e^{b*x}$$

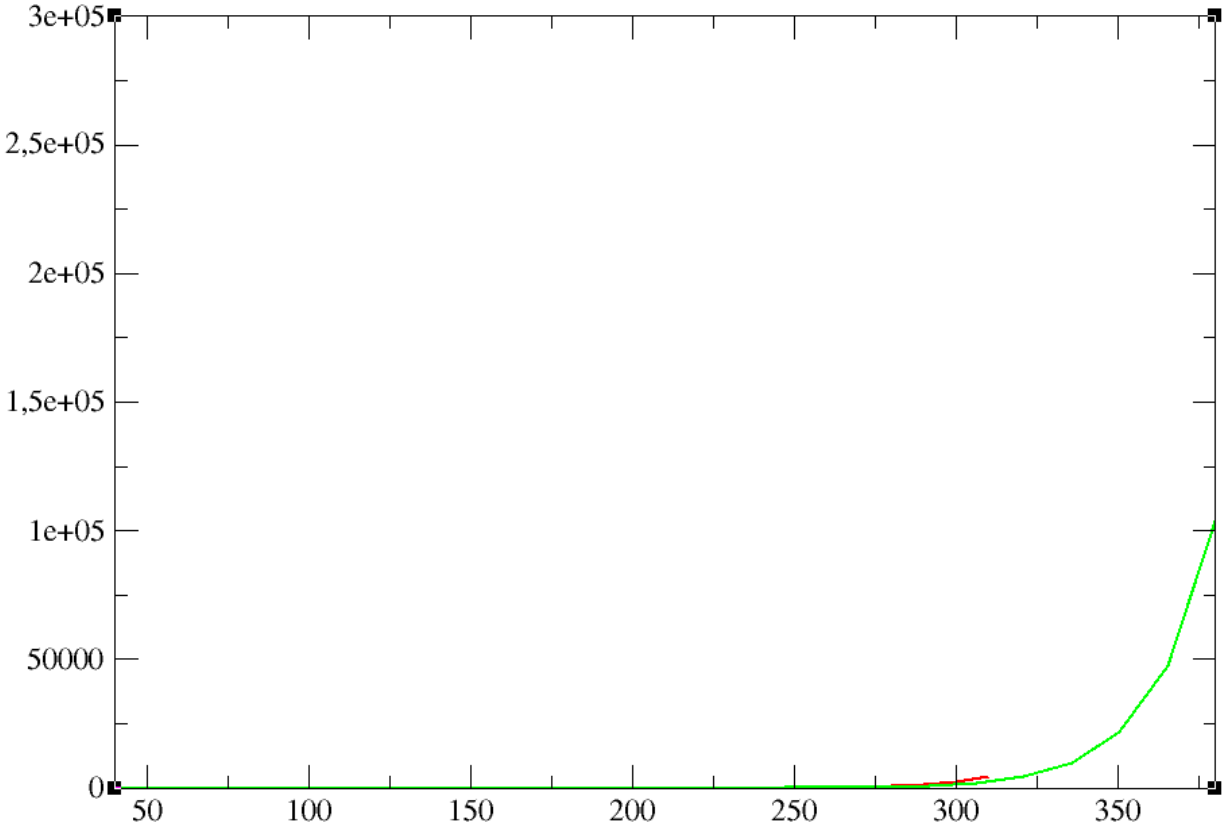
Donde Y es el tiempo en segundos necesario, X los bits a factorizar, a y b coeficientes.

Para el cálculo de estos coeficientes y de las aproximaciones de los tiempos en general, hemos usado el programa *Xgrace*, y los tiempos dados por el algoritmo *Yafu*. Las siguientes gráficas muestran la aproximación hasta 380 bits de cada una de las máquinas. La gráfica roja representa el tiempo real, y la verde la aproximación. El eje X (abajo) son los bits factorizados, y el eje Y (izquierda) el tiempo necesario en segundos.

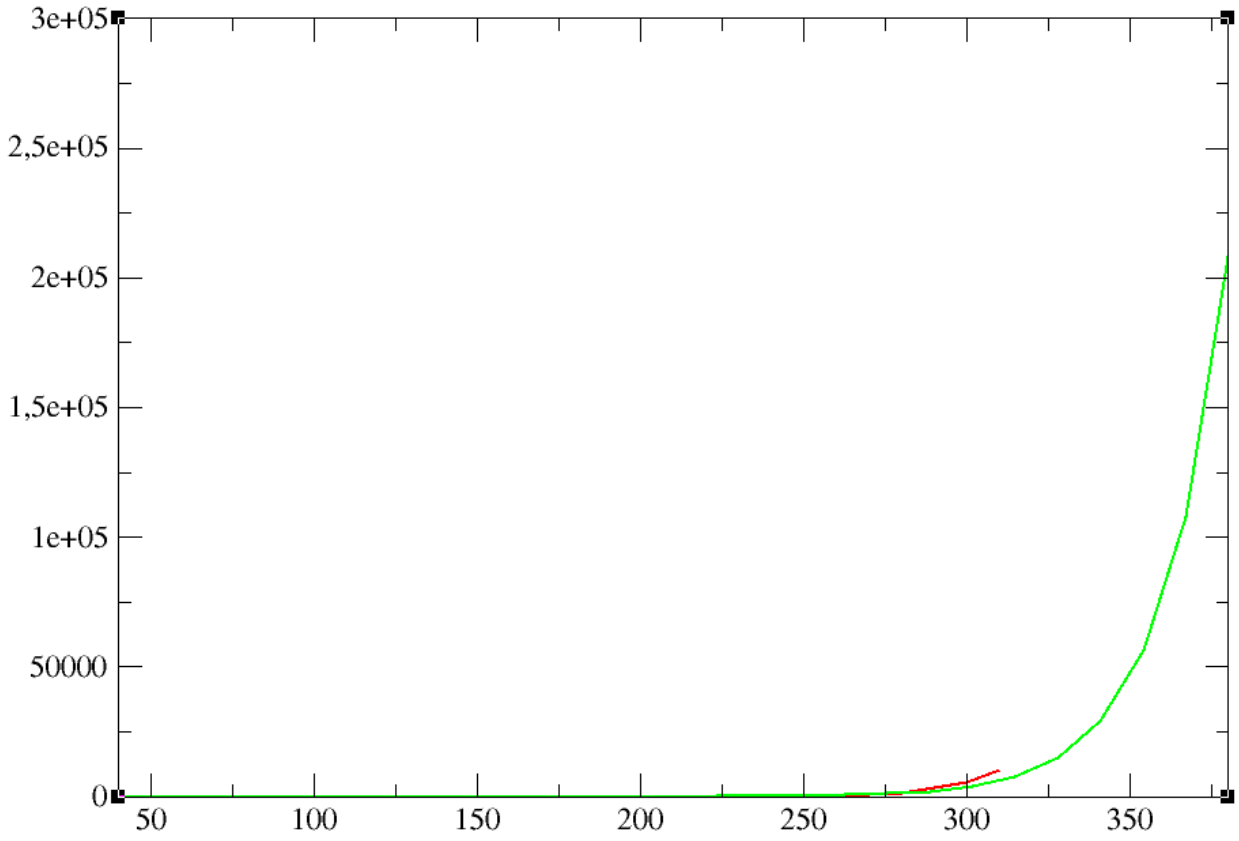
Intel Core 2 Quad Q6600



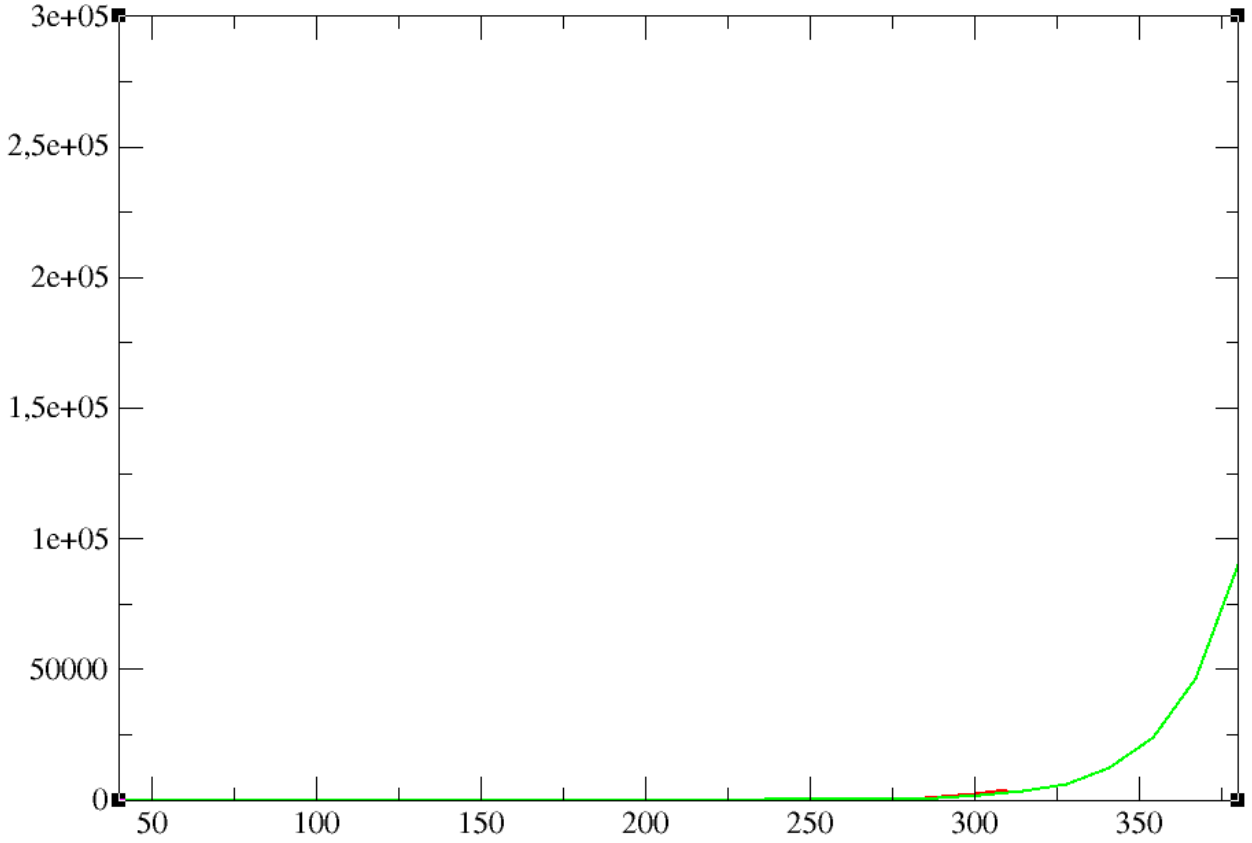
I7 4820k



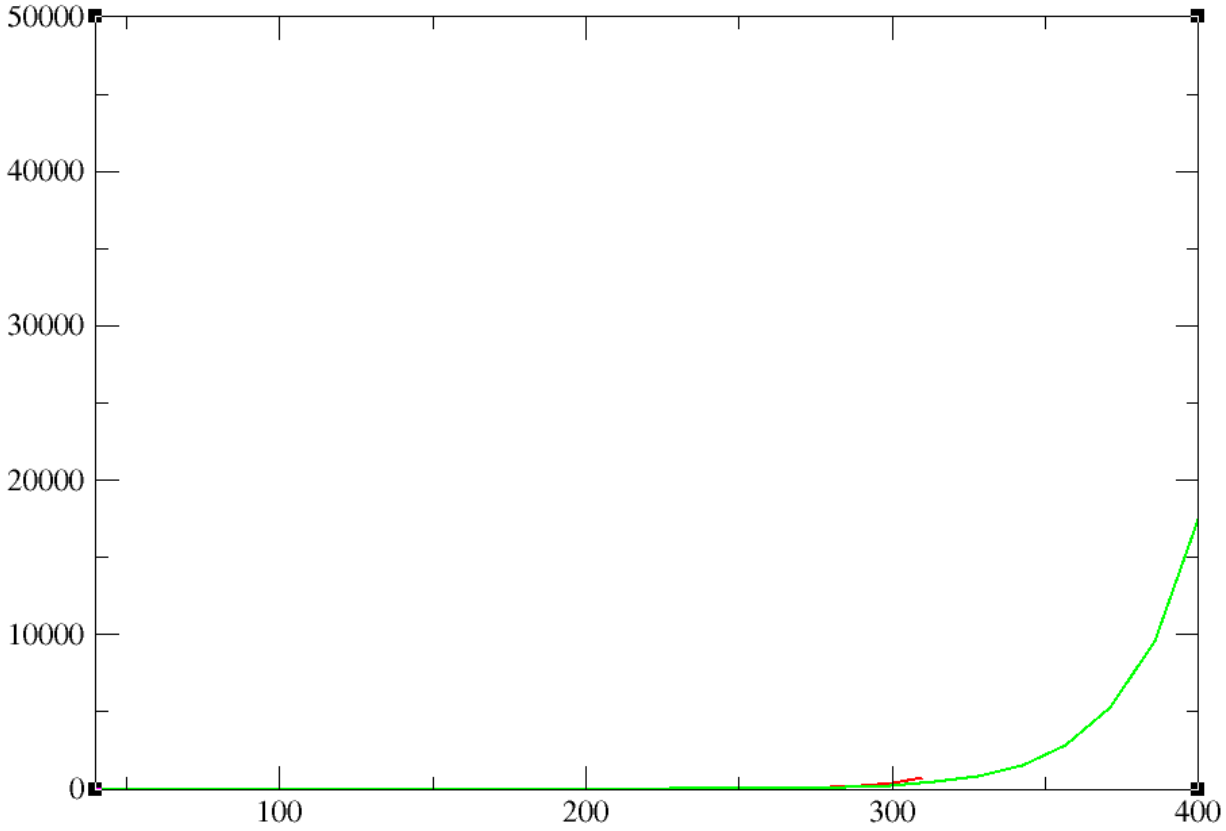
Intel Dual-CPU-E2180-2.00GHz



Servidor church (1 hebra)



Servidor church (12 hebras)



En esta última gráfica (la del servidor church con 12 hebras) se ha reducido la escala de tiempo de 300.000 segundos a 50.000, para poder ver mejor el ajuste, y se ha llegado hasta los 400 bits factorizados, ya que la aproximación permite generar una gráfica apreciable hasta los 400 bits (a diferencia de los anteriores, que dispararía el tiempo).

Para la obtención de las gráficas, se han calculado los coeficientes a y b descritos anteriormente, dejando las funciones de aproximación de la siguiente forma:

Intel Core 2 Quad Q6600

$$Y = 0.00028553 * e^{0.054759\ X}$$

I7 4820k

$$Y = 0.00018719 * e^{0.05298\ X}$$

Intel Dual-CPU-E2180-2.00GHz

$$Y = 0.0011699 * e^{0.049991\ X}$$

Servidor church (1 hebra)

$$Y = 0.00036413 * e^{0.050857\ X}$$

Servidor church (12 hebras)

$$Y = 0.0009888 * e^{0.041723\ X}$$

Llegados a este punto, y con todos estos datos, sería interesante calcular el tiempo que cualquiera de éstas máquinas tardaría en factorizar un número usado en una comunicación de hoy día. Se recomienda usar como mínimo una cifra de 2048 bits para cifrar la comunicación, aunque hoy día se sigue usando 1024 bits. Algunas corporaciones tienen un cifrado mas fuerte con 4096 bits. Por lo que utilizando las ecuaciones de aproximación calculadas (donde la X es la longitud en bits), podemos estimar el tiempo necesario por cada máquina en lograr romper una clave usada hoy día.

Bits factorizados	Tiempos (en años)			
	Intel Core 2 Quad Q6600	I7 4820k	Servidor church (1 hebra)	Servidor church (12hebras)
1024	$2.93*10^{13}$	$2.16*10^{12}$	$4.78*10^{11}$	112.525.815
2048	$4.58*10^{37}$	$7.86*10^{35}$	$1.97*10^{34}$	$4.03*10^{26}$
4096	$2.32*10^{86}$	$1.04*10^{83}$	$3.39*10^{79}$	$5.20*10^{63}$

Nota: No se ha incluido en la comparación los años necesarios del procesador Dual-CPU E2180 por que no ha sido posible calcular un ajuste correcto, ya que se porta bien en la aproximación de los valores más inmediatos (400,500 bits…), pero en los valores calculados, no sale una cifra razonable (llega a tardar menos que el procesador I7 4820k). De hecho intuimos un mal ajuste al ver el exponente al que se eleva el número e, que es el segundo más bajo de todos (cuando debería ser el más alto).

Viendo los resultados obtenidos, deducimos que actualmente es imposible intentar descifrar una comunicación cifrada basada en RSA con las máquinas actuales, ya que la máquina que menos tarda es el servidor church (y no es precisamente una máquina de escritorio que esté al alcance económico de cualquiera) y tarda 112.525.815 años en descifrar 1024 bits. Y hay que tener en cuenta que 1024 bits se siguen usando hoy día, aunque este cifrado está siendo sustituido por el de 2048 bits. Partiendo de que se estima que la edad del universo es de varios miles de millones de años, cualquiera de las máquinas analizadas, si se hubiera puesto a hacer cálculos en el mismo instante en el que empezó a originarse el universo, a día de hoy aún no habría acabado de hacer cálculos, para un cifrado de 2048 bits.

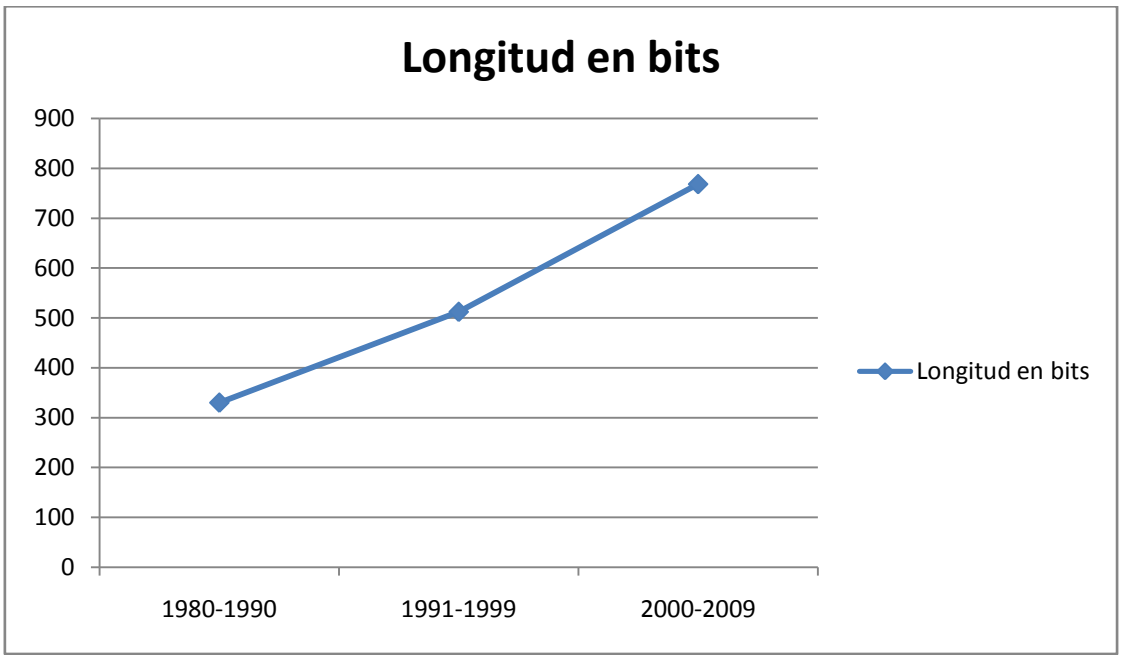
3ª parte: Evolución histórica. Computación Distribuida vs Supercomputación. Nuevos ataques.

Evolución histórica[7]

La factorización de números ha tenido una evolución a lo largo de los últimos años. En la siguiente tabla se muestra el avance de la factorización, incluyendo la longitud del número factorizado (tanto en bits como en dígitos) y el año en el que se logró factorizarlo:

Año	Longitud en bits	Dígitos
2009	768	232
2005	663	200
2003	530	160
1999	512	155
1996	430	130
1994	426	129
1993	397	120
1992	364	110
1991	330	100

En la siguiente gráfica se muestra por décadas los datos de la tabla anterior. Puede parecer que la gráfica tiene una tendencia linear, pero no es así, ya que si nos fijamos, de 1990 al 2000 se han factorizado poco menos de 200 bits de diferencia, mientras que de la década del 2000 al 2009 se han factorizado casi 300 bits de diferencia, por lo que tiene una tendencia exponencial.



El número de dígitos que tiene un número de X bits se puede calcular de la siguiente forma:

$$\log_{10} Y = X * \log_{10} 2$$

Donde X son el número de bits que forman el número, e Y es el número de dígitos totales que forman el número.

Estas fechas se corresponden a las anunciadas por el RSA Challenge, una iniciativa de la empresa RSA, en la que se premia a aquellas personas que sean capaces de factorizar un número de determinada longitud.

Dicha iniciativa tiene su razón de ser en las mejoras tecnológicas que trae consigo el avance del tiempo. Conforme pasa el tiempo, los equipos alcanzan mejores prestaciones, y también va bajando su precio de compra. Equipos que en su día solamente estaban al alcance de grandes organizaciones y /o empresas, hoy día se pueden conseguir por unos pocos cientos de euros. Esto fuerza a tener que hacer claves de longitud mayores, para que la seguridad que ofrece el algoritmo RSA siga siendo fuerte.

El efecto secundario de manejar cada vez claves mas largas, es que el tiempo necesario para poder cifrar y descifrar la clave (legítimamente) es cada vez mayor. Sin embargo, este tiempo crece de forma lineal, mientras que el tiempo necesario para poder factorizar (es decir, el tiempo que necesita el atacante) crece de forma exponencial. Aunque es injusto comparar estos tiempos, ya que un atacante, y un emisor legítimo, se enfrentan a trabajos distintos. De todas formas, desde el punto de vista legítimo, sale a cuenta que se produzca este fenómeno.

La siguiente tabla muestra, en datos, lo que se habla en el párrafo anterior. Son los números que fueron factorizados (entre otros muchos) a lo largo de la historia, así como el tiempo (medido en horas CPU) que necesitaron estos "atacantes" para poder factorizarlos. Estos atacantes no iban con malas intenciones, sino que lo hicieron por propósitos de investigación, así como para poder cobrar la recompensa del RSA Challenge:

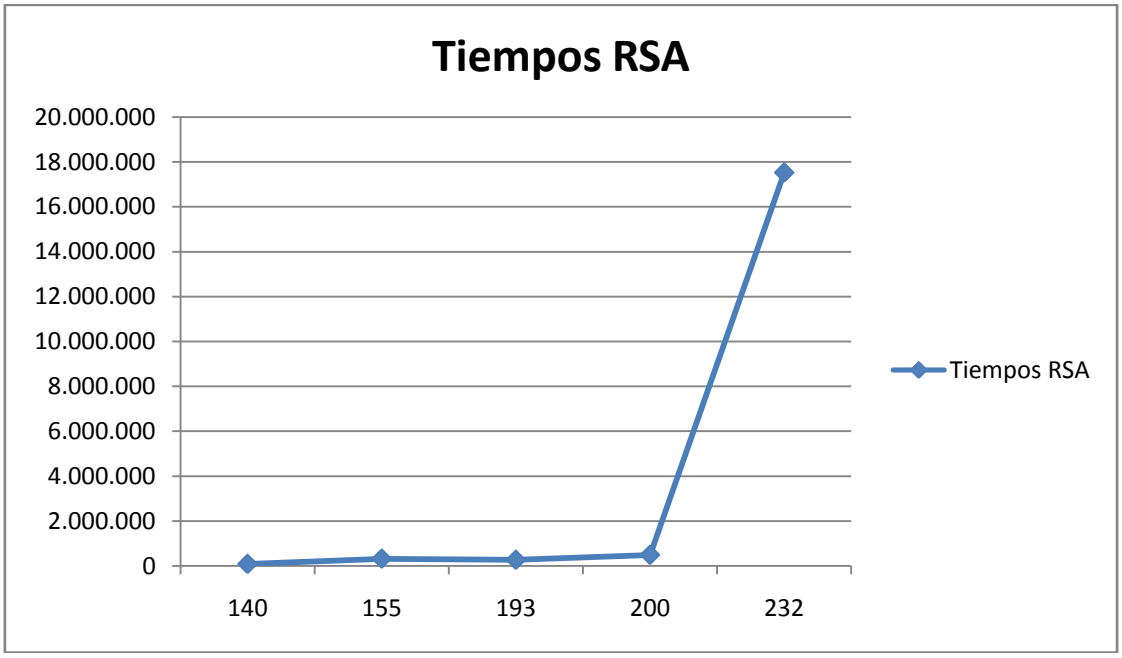
Año	Dígitos del número factorizado	Horas CPU necesitadas
2-2-1999	140	77.964
22-8-1999	155	312.732
2-11-2005	193	262.800
X-5-2005	200	481.800
12-12-2009	232	17.520.000

El concepto de "hora CPU" simboliza el tiempo real que el programa está en el procesador, descartando tiempos necesarios para entrada/salida, tiempos de captación de datos de memoria, etc. En pocas palabras: el tiempo que la CPU está realizando operaciones del programa, descartando cualquier otro factor que impida la ejecución del programa.

Cada CPU usada en la factorización RSA fue distinta en cada prueba. De hecho, se usaba mas de un equipo en cada prueba, para poder agilizar el tiempo necesario (tiempo real) para presentar los resultados. Y es que como es lógico, una hora de CPU en un Pentium IV a 2.4 GHz no produce la misma capacidad de computo que un i7 a 3.2GHz.

A continuación, se detallan los equipos usados en cada reto superado de factorización:

- **140 dígitos** ➔ 60 equipos con un procesador de 300 MHz de media
- **155 dígitos** ➔ 160 Sunworkstations de 175 a 400 MHz cada una, 8 SGI Origin de 250MHz, 120 equipos con Pentium II de 300 a 450 MHz, 4 equipos Compaq de 500 MHz
- **193 dígitos** ➔ Varios equipos con procesadores Opteron de 2.2GHz (no se especifica número concreto)
- 200 dígitos ➔ 80 equipos con procesadores Opteron de 2.2 GHz, donados por la compañía Fujitsu
- **232 dígitos** ➔ Fue una cooperación entre varias organizaciones y cooperadores (como Scott Contini, Paul Leyland, EPFL CWI, etc.). No se especifica el número de equipos ni su velocidad media, aunque en el comunicado de los investigadores aseguran que se utilizó hardware de doble núcleo.


Tiempos RSA
20.000.000
18.000.000
16.000.000
14.000.000
12.000.000
10.000.000
8.000.000
6.000.000
4.000.000
2.000.000
0
140
155
193
200
232
Tiempos RSA

Supercomputación VS Computación Distribuida[8][9][10]

Un supercomputador nace de una idea en la que se tiene una gran carencia computacional, y un elevado techo económico, para poder montar una infraestructura a la altura de estas necesidades. Un supercomputador no está al alcance de cualquiera: solamente se lo pueden permitir grandes organizaciones, como multinacionales importantes y órganos de gobierno.

La necesidad de un supercomputador es principalmente, poder llevar a cabo cálculos de computación que un ordenador (o varios de estos) de la gama del mercado "de a pie" tardaría muchos años en terminar. Los propósitos de estos cálculos son muy diversos: pueden ser con fines comerciales (cómo por ejemplo, una empresa farmacéutica intenta predecir la evolución del ARN de diversos virus), con fines de espionaje (como muchas teorías de la conspiración defienden, o algunos hechos más palpables, como es el caso Snowden, el cuál afirmaba que el gobierno de los EEUU podría ver información sensible de los ciudadanos, información que en teoría estaba cifrada y solo una gran capacidad de computo puede descifrar), y otros tantos fines.

La lista *top500* es una famosa página que trata del ámbito de los supercomputadores, haciendo una lista (cada mes de junio y noviembre) en los que ordena los computadores más rápidos de los que se tiene constancia hasta ahora. Tener un supercomputador introducido en esta lista es misión de algunas organizaciones y empresas, que lo toman como "buena imagen" y como un elogio.

España tiene actualmente 8 supercomputadores, que han llegado muy altos en la lista *top50,* como por ejemplo *Magerit* en la universidad Politécnica de Madrid, considerado el supercomputador de propósito general y sin uso de aceleración gráfica más potente de España, alcanzando el puesto 136 en Junio de 2011. Otro ejemplo, esta vez quedándonos más cerca de casa, es el supercomputador de la UGR, llamado *Ugrgrid*, que también consiguió colarse en la lista *top500* en Junio de 2007.

Una de las ventajas de poseer un supercomputador, es que la empresa u organización es la única consciente de los cálculos que se están llevando a cabo en él, y salvo por temas de filtraciones, se podría guardar algún proyecto de supercomputación que contenga información sensible, bien sea por que generaría un escándalo social (como el caso Snowden), o bien por que sean los resultados de algún trabajo de investigación, aplicado luego a fines comerciales, los cuales si caen en las manos de la competencia podría ser nefasto.

Sea cuál sea el propósito, la necesidad está ahí, y como ocurre normalmente, satisfacer una gran necesidad acaba derivando en un desembolso económico importante. Y este es el gran punto en contra de la supercomputación. No solamente el coste de la adquisición y la instalación del supercomputador (que puede rondar los millones de euros), sino el mantenimiento de éste, sobre todo en el apartado de refrigeración. Se necesita un sistema de refrigeración avanzado, que esté funcionando todo el tiempo que el supercomputador está encendido (que normalmente suele ser siempre). Hay que tener en cuenta que algunos procesadores de estos sistemas pueden irradiar hasta diez veces

más de calor que una estufa convencional. Ante este problema, que obliga a pagar facturas eléctricas astronómicas, IBM ha desarrollado una tecnología innovadora de refrigeración por agua, inspirándose en la circulación de la sangre en las personas: llevar el agua hasta el "corazón" de la máquina. Estamos hablando de introducir canalizaciones de agua que llegan dentro del procesador, no solamente acoplándose arriba, como se hacía antaño. Esto obviamente obliga a construir los procesadores con una arquitectura muy distinta, aunque IBM ha demostrado que puede ahorrarse hasta un 40% en la factura eléctrica del sistema de refrigeración.

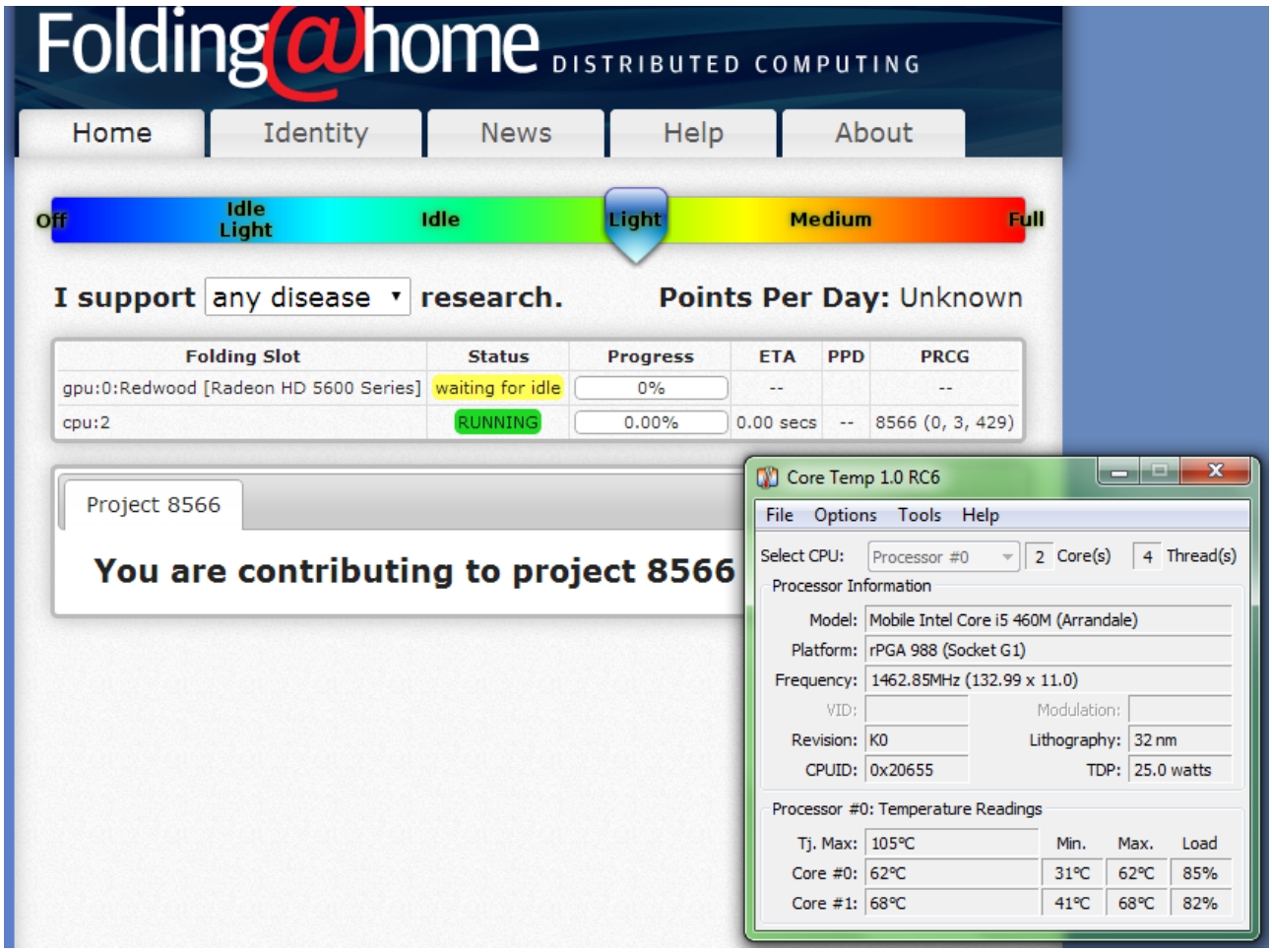
La otra gran desventaja, es tener que instalar todo el sistema en un único sitio, por lo que hay que disponer de un espacio dedicado a tal efecto, que debe ser albergado en un lugar de bastantes metros cuadrados.

Ante todos estos problemas, surge una nueva idea, inducida por el avance tecnológico en la sociedad, en la que desde hace tiempo, tener un ordenador en casa está al alcance de la mano de muchas familias. Estos equipos son equipos de escritorio, pensados para llevar a cabo unas tareas de un usuario "de a pie", y para nada están diseñados para poder satisfacer las necesidades computacionales a las que se enfrentan los supercomputadores.

Sin embargo, hay que tener dos puntos principales en cuenta; uno es que el tiempo que el procesador de un equipo de escritorio está ocioso es bastante elevado, en torno al 90%, causado por las esperas de entrada/salida. Y el otro punto importante, es que hay muchísimos equipos conectados a Internet en todo el mundo. Si juntamos estos dos factores, en los que se aprovechan los "tiempos muertos" de equipos con prestaciones pésimas (en comparación con los supercomputadores), y que hay cientos de millones a lo largo de todo el mundo, nos encontramos con la idea principal de la computación distribuida.

En la computación distribuida, se divide el problema que tiene un supercomputador en muchos subproblemas, de tal forma que cada uno de éstos se le encarga a un equipo de escritorio. De esta forma, se consigue que todos trabajen en paralelo, y que aunque tengan por separado muy malas prestaciones en comparación con un supercomputador, al ser tantísimos equipos, reduce drásticamente el tiempo necesario en acabar la tarea.

Hay dos formas de participar en un cálculo de computación distribuida: voluntaria e involuntaria.
En la forma voluntaria, se participa descargando un programa, que se encargará de recibir un trabajo que le mandará el servidor que esté al cargo del proyecto (o bien de forma offline y compartiendo los resultados manualmente después), y comienza la ejecución del problema asignado. Un ejemplo es el de la Universidad de Standford, en la que descargando un programa, se puede participar en distintos proyectos, que aseguran ser proyectos de investigación en campos de salud.



En la imagen se muestra una captura de pantalla de la contribución del proyecto 8566, en la que se observa cómo el procesador incrementa de forma considerable la carga que tiene (apartado Load). Hay que decir que el rendimiento general del sistema no se ve afectado de forma general teniendo esta aplicación ejecutándose, debido a que en éste momento se están llevando a cabo tareas más ligeras computacionalmente hablando. Este tipo de programas es incompatibles si el usuario desearía ejecutar algún juego (ya que sí vería afectado el rendimiento de éste), renderización gráfica, o cualquier otra aplicación en la que el usuario necesite una mayor computación..

La otra forma de participar en un problema de computación distribuida, es de forma involuntaria, en la que nuestro equipo forma parte de una BotNet, tras haberse producido una infección por parte de un virus informático. En este momento, cuando la persona que controla la BotNet decida mandar a nuestro equipo una tarea, éste la ejecutará y mandará los resultados. Por desgracia, normalmente los fines de estas personas no suelen ser de etnia científica, sino que está relacionado con algún ciberdelito. Por ejemplo, se ha podido capturar los paquetes de datos de una transferencia bancaria, cifrados con el algoritmo RSA. Es posible utilizar una BotNet si es lo suficientemente grande, potente, y el cifrado es relativamente débil.

Las ventajas de la computación distribuida, es su bajo coste, en el sentido de que “ya está comprada y mantenida” (porque habrá equipos conectados a Internet en todo momento), y que no se necesita un sitio amplio en el que instalar los equipos, ya que éstos están

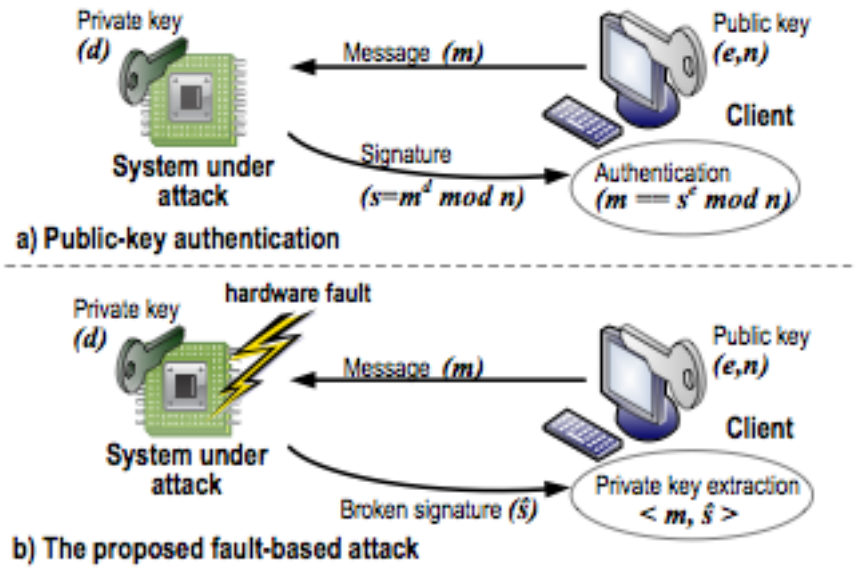
esparcidos por muchos hogares del mundo. Por el contrario, no se puede mantener una discreción acerca del proyecto, ya que mucha gente tiene parte de éste en su propia máquina. Incluso bajo una forma de computación distribuida involuntaria, un usuario avanzado que haya contraído la infección, podría mediante sus conocimientos, llegar al fondo del asunto y saber qué está haciendo exactamente su equipo. Además hay que orientar los esfuerzos a hacer publicidad del proyecto (si lo hacemos de forma voluntaria), para que un amplio abanico de personas quieran participar en él.

De todas formas, ninguna de las dos opciones (ni la computación distribuida ni la supercomputación) es mejor que la otra, todo depende del caso. Si queremos llevar a cabo un proyecto sin disponer de capital, que no es ningún secreto y que puede ser conocido por la sociedad, y que trata acerca de un tema socialmente aceptado, nuestra mejor opción es la computación distribuida. Por contra, si queremos simular como se portaría un prototipo de avión militar bajo muchas condiciones meteorológicas y de altura, y cuyo diseño no puede ser de conocimiento público, la opción a tomar será la de utilizar un supercomputador.

Nuevos ataques contra RSA[11][12]

Recientemente los investigadores Andrea Pellegrini, Valeria Bertacco y Todd Austin de la universidad de Michigan han llevado a cabo con éxito un experimento de ataque a RSA-1024, este ataque no se basa en la factorización, ni en ningún tipo de cómputo matemático si no en la generación de fallos hardware para producir volcados de información no autorizada, el trabajo se ha llamado **"FaultBasedAttack of RSA Authentication"** o "ataque basado en fallos para autenticación RSA". Este fallo sólo se ha probado en dispositivos como reproductores multimedia, smartphones, y algunos ordenadores pero no se cree posible hacia grandes infraestructuras como servidores de empresas etc.

El ataque se produce provocando ligeras variaciones del voltaje suministradas al equipo que posee la llave privada poniéndolo bajo "presión" y provocando que cometa pequeños errores en sus comunicaciones, estos fallos muestran pequeños fragmentos de la clave que tras la recopilación de muchos puede ser reconstruida completamente. En este caso de estudio se consiguió sacar la clave de 1024 bits en alrededor de 100 horas, lo cual mediante un ataque de factorización como los que hemos realizado en este trabajo habría tardado billones de años.



Cómo hemos dicho sólo se puede realizar contra dispositivos que no tengan protección hacia esas variaciones de tensión y los grandes servidores y supercomputadores tienen medidas protectoras ante ellos.

La ocurrencia de fallos hardware: La tecnología actual basada en chips de silicio a escalas tan reducidas hace que la ocurrencia de fallos hardware transitorios sea un fenómeno natural, no tiene porqué ser inducido, la pueden causar partículas alfa ambientales o neutrones que conmutan el estado de algunos transistores. Al igual que se producen de esta forma natural los errores pueden ser inducidos por un atacante. Una vulnerabilidad sistemática a estos ataques también puede ser introducida en el proceso de fabricación por lo que algunos componentes en el sistema pueden ser

massusceptiblesa estos errores que otros.

Varios de los productos electrónicos de consumo, como los ordenadores “ultra-portátiles”, teléfonos móviles y dispositivos multimedia son particularmente susceptibles a los ataques basados en fallos, es fácil para un atacante obtener acceso físico a este tipo de sistemas. Además, incluso un usuario legítimo de un dispositivo podría perpetrar un ataque basado en fallos en él para extraer información confidencial que un fabricante del sistema quiere mantener segura, como en el caso de los reproductores multimedia.

En el experimento realizado por estos investigadores se pusieron a prueba 10000 firmas calculadas utilizando RSA-1024. Una vez recogidas utilizando el sistema basado en fallos se comprobaron utilizando las claves públicas de las 10000 firmas para ver si eran correctas, 8800 fueron correctas y solo un 12% había incurrido en un error de un solo bits en el resultado. Por lo que el sistema funciona con un margen de error del 12% lo cual es bastante poco teniendo en cuenta que para recolectar la firma entera hay que extraer cientos de fragmentos de la misma para poder reconstruirla a posteriori, y en la recolección de todos estos fragmentos se pueden introducir bit erróneos como es el caso, o perder algún bits de la clave que sea correcto.

Como conclusión de este nuevo ataque se puede decir que ya no es necesario disponer de grandes sistemas de cómputo que realicen la factorización de las claves, ni sistemas distribuidos que se repartan el trabajo, simplemente con un ordenador personal podemos “escuchar” el flujo de datos que generan los fallos hardware y aprovecharlos para conseguir las claves de los dispositivos, por lo que la complejidad computacional que en un ataque de factorización es “pseudo exponencial” pasa a ser lineal en este tipo de ataques puesto que el tiempo de “rotura” de la clave depende del tamaño de esta de una forma lineal, si estos investigadores han conseguido sacar las claves de 1024 bits en alrededor de 100 horas se podría suponer que para una clave de 2048 bits se tardaría el doble si el atacante es el que genera los errores en el hardware puesto que sólo se necesita recolectar la información del flujo de datos y no es necesario ningún cómputo para esta recolección, el único cómputo necesario es a la hora de unir la clave pero sigue sin ser de orden exponencial.

Bibliografía:

[1] http://es.wikipedia.org/wiki/RSA

[2] http://www.criptored.upm.es/software/sw_m021_02.htm

[3] http://www.criptored.upm.es/software/sw_m001l.htm

[4] http://www.compapp.dcu.ie/~mike/shamus.html

[5] http://sourceforge.net/projects/msieve/

[6] http://sourceforge.net/projects/yafu/

[7] http://www.emc.com/emc-plus/rsa-labs/historical/rsa-155-factored.htm

[8] http://www.top500.org/list/2013/11/?page=1

[9] http://folding.stanford.edu/

[10] http://www.agenciasinc.es/Reportajes/IBM-se-inspira-en-la-circulacion-humana-para-refrigerar-superordenadores-con-agua

[11] http://www.solociencia.com/informatica/10043012.htm

[12] http://web.eecs.umich.edu/~valeria/research/publications/DATE10RSA.pdf