La Columna de Matemática Computacional

Sección a cargo de

Tomás Recio

El objetivo de esta columna es presentar de manera sucinta, en cada uno de los números de La Gaceta*, alguna cuestión matemática en la que los cálculos, en un sentido muy amplio, tengan un papel destacado. Para cumplir este objetivo el editor de la columna (sin otros méritos que su interés y sin otros recursos que su mejor voluntad) quisiera contar con la colaboración de los lectores, a los que anima a remitirle (a la dirección que se indica al pie de página*[1]*) los trabajos y sugerencias que consideren oportunos.*

En este número . . .

. . . el profesor de la Universidad de Santiago, José Luis Gómez Pardo, contribuye con un extenso artículo, en el que describe de manera particularmente amena las aplicaciones de la teoría computacional de números y de las curvas elípticas a la criptografía.

Como en los dos artículos del mismo autor, recientemente aparecidos en La Gaceta, también en esta ocasión Gómez Pardo nos deleita con un estilo que engancha fácilmente al lector por su capacidad para conjugar

- el desarrollo histórico y la mayor actualidad (con múltiples referencias a documentos accesibles sólo a través de internet),
- la descripción minuciosa de los fundamentos matemáticos y de su desarrollo hasta la fecha: por ejemplo, se ha introducido una cita en relación con el anuncio de un resultado (con fecha 6 de agosto de 2002) en el que se presenta un algoritmo determinista de tiempo polinómico para determinar la primalidad de un entero
- las aplicaciones a problemas tecnológicos de amplio interés (véase la referencia a Windows Media Player, al final del artículo, por ejemplo)
- ofreciendo al lector la posibilidad de interactuar con el contenido del texto, mediante los programas en Mathematica que el autor nos presenta.

Por todo ello deseamos agradecer sinceramente, desde estas páginas, al prof. Gómez Pardo por su extraordinaria colaboración.

[1]Tomás Recio. Departamento de Matemáticas. Facultad de Ciencias. Universidad de Cantabria. 39071 Santander. recio@matesco.unican.es

Criptografía y curvas elípticas

por

José Luis Gómez Pardo

Introducción

Es bien sabido que, contrariamente a lo que G.H. Hardy pensaba cuando escribió, en 1940, su libro autobiográfico *Apología de un matemático* [17], la teoría de números tiene muchas aplicaciones, aunque justo es reconocer que la mayor parte de ellas eran difíciles de imaginar en aquella época. Aunque no es la única, una de las razones más importantes de este profundo cambio de perspectiva fue el descubrimiento, hecho por Diffie y Hellman en 1976, de la *criptografía de clave pública* [13]. En estas notas trataré de dar una introducción sencilla a algunas de las aplicaciones de la teoría de números computacional a la criptografía y, en particular, a aquéllas que hacen uso de las curvas elípticas.

Para comprender el alcance de este cambio revolucionario en la criptografía y su impacto en la teoría de números es conveniente, en primer lugar, pasar revista a los objetivos de la criptografía moderna, tal como se usa, por ejemplo, en las transacciones comerciales o bancarias a través de redes informáticas. Los más importantes son:

- **Confidencialidad.** A envía a B un mensaje que no puede ser leido por nadie más.
- **Autenticidad.** Cuando B recibe un mensaje de A, sabe que es realmente A quien lo envía.
- **Integridad.** B puede detectar si el mensaje que le ha enviado A ha sido alterado por un tercero.
- **No repudio.** Después de haber enviado un mensaje a B, A no puede afirmar que el mensaje no es suyo.

La importancia de todos estos objetivos es fácil de comprender si se piensa, por ejemplo, en el intercambio de mensajes originado porque A compre un artículo a B a través de Internet realizando el pago con una tarjeta de crédito.

Los criptosistemas clásicos, en uso hasta finales de los años 70, se basaban en el uso de una clave secreta que A y B compartían y que era la que se usaba tanto para cifrar como para descifrar (de ahí el nombre de *criptosistemas simétricos*, que también reciben). Estos criptosistemas estaban diseñados pensando en el primero de los objetivos mencionados, es decir, en mantener la información secreta, pero carecían de mecanismos para alcanzar los restantes. Por ejemplo, para que B pueda convencer a una tercera persona de que un

cierto mensaje que ha recibido procede de A, se necesita una *firma digital*, que es el análogo electrónico de una firma escrita. Dado que en un criptosistema de clave secreta A y B tienen la misma capacidad para cifrar y descifrar, B podría haber falsificado el mensaje y atribuírselo a A. Aunque los criptosistemas simétricos se adaptan bien a los fines para los que habían sido originalmente concebidos, que eran fundamentalmente militares o diplomáticos, resulta evidente que no son adecuados, por sí solos, para una situación como la antes descrita, es decir, para proteger las comunicaciones entre múltiples personas que probablemente no se conocen y nunca antes han estado en contacto. El uso de una clave secreta requiere que ésta haya sido compartida a través de un *canal seguro*, algo que no está fácilmente disponible en la situación indicada. A todo ello hay que añadir una dificultad adicional derivada del uso masivo que se hace hoy en día de la criptografía. En una red con n usuarios, un criptosistema clásico requiere que cada par de usuarios compartan una clave secreta, lo que implica un total de $n(n-1)/2$ claves y hace que, para n grande, su gestión se haga inmanejable.

Otro de los aspectos insatisfactorios de los criptosistemas tradicionales era la inexistencia de criterios rigurosos para evaluar su seguridad, es decir, su resistencia al *criptoanálisis*. De la misma forma que la criptografía es la ciencia de diseñar sistemas para proteger la información y las comunicaciones (criptosistemas), existe la disciplina opuesta que trata de encontrar métodos para romper –o, en términos más técnicos, criptoanalizar– dichos criptosistemas y recibe el nombre de criptoanálisis. La unión de ambas constituye la *criptología* y la historia de ésta se puede ver como una lucha entre criptógrafos y criptoanalistas en la que, contrariamente a lo que mucha gente piensa, han sido habitualmente los segundos los que han terminado venciendo. Este hecho ya va implícito en el título de la monumental historia de la criptología anterior a la segunda guerra mundial escrita por David Kahn [21] que apunta claramente a quienes fueron los verdaderos héroes de esta historia: "The Codebreakers", es decir, "los rompedores de códigos" o, en terminología más moderna, los criptoanalistas. La historia de la criptología está cuajada de nombres de grandes criptoanalistas: Charles Babbage (profesor de matemáticas en Cambridge, famoso por haber diseñado máquinas precursoras de los ordenadores actuales y que fue el primero en criptoanalizar cifras de sustitución polialfabética a mediados del siglo XIX), Friedrich Kasiski (oficial prusiano autor del "test de Kasiski", mediante el cual criptoanalizó la cifra de Vigenère), Georges Painvin (criptoanalista francés que rompió las cifras alemanas al final de la primera guerra mundial), William Friedman (criptoanalista norteamericano que fue probablemente el primero en desarrollar el criptoanálisis como una disciplina científica y en los años anteriores a la segunda guerra mundial criptoanalizó la máquina japonesa "Purple"), Marian Rejewski (matemático polaco que fue el primero en criptoanalizar la máquina alemana *Enigma* en los años precedentes a la Segunda Guerra Mundial), Alan M. Turing (más conocido por ser el padre de la "teoría de la computación" y que fue miembro destacado del llamado "grupo de Bletchey Park" que, tomando como punto de partida el trabajo de

Rejewski, criptoanalizó las nuevas versiones de la máquina Enigma durante la segunda guerra mundial). A pesar de esto, ha existido siempre una creencia popular en la invulnerabilidad de los criptosistemas, que se pone de manifiesto en una curiosa anécdota, extraída de las memorias de Casanova, en las que éste narra como utilizó el criptoanálisis (el análisis de frecuencias sobre un criptosistema de sustitución) para descifrar un manuscrito que supuestamente contenía la fórmula para transmutar metales en oro. Este manuscrito había sido cifrado por una incauta señora, llamada madame d'Urfé, quien se lo prestó a Casanova en la convicción de que era indescifrable pues sólo ella conocía la clave (que, por cierto, era la palabra "Nabucodonosor"). Cuando Casanova le demostró que había descifrado el manuscrito y, más aun, que conocía la palabra clave, madame d'Urfé se quedó tan asombrada que él aprovechó la ocasión para decirle que un genio se la había revelado, lo que produjo en ella una impresión tremenda. No consta que Casanova haya aprovechado este episodio para fabricar oro, pero sí que lo aprovechó, como era de esperar, para seducir a la señora en cuestión, como se deduce de las palabras con las que termina su relato: *"A partir de ese momento me hice el dueño de su alma y abusé de mi poder. Cada vez que pienso en ello, me apeno y me avergüenzo y, como penitencia, me impongo la obligación de decir la verdad al escribir mis memorias".*

La razón por la cual los criptoanalistas siempre terminaban triunfando ha sido bien expresada por Edgar Alan Poe en su relato "The Gold Bug" ("El escarabajo de oro"[37]) cuando pone en boca de su personaje Will Legrand la siguiente afirmación: *"es dudoso que el ingenio humano pueda crear un enigma de ese género* –se refiere a un criptograma– *que el mismo ingenio humano no resuelva con una aplicación adecuada"*. Pero, eventualmente, los criptógrafos llegarían a pensar: ¿y si no basáramos los criptosistemas o, al menos, no solamente, en el ingenio? ¿y si los basáramos en problemas matemáticos difíciles?

La idea de basar los criptosistemas en problemas matemáticos difíciles iba a ser clave para el desarrollo del nuevo tipo de criptografía (aunque también puede ser aplicada al diseño de criptosistemas simétricos) pues iba a permitir, entre otras cosas, evaluar con mucha mayor precisión la seguridad de los criptosistemas. Históricamente, la confianza depositada en un criptosistema ha ido creciendo a medida que pasaba el tiempo sin que el sistema hubiese sido roto. Esto se pone de manifiesto claramente en la evolución del *criptosistema de Vigenère* [21] que, después de haber sido introducido en el siglo XVI y usado ampliamente durante varios siglos, no fue criptoanalizado hasta el año 1863 por Kasiski, justo cuando su fama era mayor y muchos lo consideraban inexpugnable. El primero en considerar el problema de la seguridad de los criptosistemas desde un punto de vista matemático fue Claude Shannon, el fundador de la "Teoría de la comunicación", quien consideró un modelo basado en el concepto de *seguridad incondicional*, que era la que tendría un criptosistema que fuese resistente al criptoanálisis incluso ante un criptoanalista dotado de recursos computacionales ilimitados. Un tal criptosistema es algo así como *"el Santo Grial de la criptografía"* (en palabras de S. Singh [46])

pero, por sorprendente que parezca, dicho sistema existe. En efecto, Shannon demostró que la llamada *cifra de Vernam* o *cuaderno de uso único* cumple este requisito y es *incondicionalmente segura*. La cifra de Vernam consiste, esencialmente, en utilizar una clave, elegida aleatoriamente, de la misma longitud que el mensaje. Por ejemplo, si el mensaje es una sucesión de bits, se genera aleatoriamente otra sucesión de la misma longitud y se combinan ambas mediante la operación XOR –suma de bits módulo 2– para producir el *criptotexto* (el mensaje cifrado; el mensaje sin cifrar recibe el nombre de *texto claro*). Obsérvese que para cada nuevo mensaje hay que utilizar una nueva clave, de ahí el nombre de cuaderno de uso único. Aparentemente, esto proporciona un criptosistema perfecto pero, en realidad, dista mucho de ser así. Por una parte existe el problema de como generar la clave aleatoriamente. Aunque no es, ni mucho menos, trivial, no me detendré en él; contentémonos con pensar que la clave se puede generar *seudoaleatoriamente*, es decir, de forma determinista pero de modo que pase una serie de tests estadísticos en relación con los cuales se comporta como si hubiese sido generada aleatoriamente. Pero hay otro problema aun mayor que hace que, en la práctica, la cifra de Vernam no sea muy útil. Es el hecho de que las dos partes que usan el sistema han de compartir, mediante un canal seguro, una clave de la misma longitud que el mensaje, lo cual presenta, en la mayoría de los casos, dificultades insalvables y, desde luego, hace que este sistema no sirva para los intercambios comerciales (aunque sí se rumorea que fue usado por espías soviéticos y de la CIA durante la guerra fría y que también es usado en el "teléfono rojo" entre Washington y Moscú).

Una de las ideas clave que subyacen en la propuesta de Diffie y Hellman es la de construir criptosistemas cuyo criptoanálisis sea, en la medida de lo posible, equivalente a la resolución de un problema matemático difícil. Aunque existen muchos problemas que no sabemos resolver, no son adecuados para este propósito y la propuesta fue utilizar problemas computacionales difíciles, en el sentido de que, aun conociendo algoritmos para resolverlos, no podemos hacerlo por no ser factible ejecutarlos en tiempo razonable. Esto conlleva cambiar el modelo de seguridad incondicional utilizado por Shannon, por otro basado en la *seguridad computacional*: el concepto de no factible manejado por Shannon, que significaba *matemáticamente imposible, con independencia de los medios disponibles* se debe sustituir por el de *computacionalmente no factible* cuyo significado es totalmente distinto. En palabras del criptográfo Gilles Brassard, el usuario de un criptosistema no debe esperar ya que el criptoanalista no tenga *información* suficiente para romperlo, sino que no tenga *tiempo*. El uso del modelo de seguridad computacional también permite un análisis más riguroso de la seguridad de un criptosistema en términos de la complejidad de los algoritmos conocidos para criptoanalizarlo, aunque este análisis no proporciona al criptógrafo tranquilidad absoluta, pues siempre queda abierta la posibilidad de que se puedan descubrir nuevos algoritmos más eficientes.

La propuesta concreta de Diffie y Hellman fue basar los criptosistemas en el concepto de *función de dirección única*, una función $f : X \to Y$ tal que $f(x)$

es (computacionalmente) "fácil" de calcular para cada $x \in X$ pero, para la mayoría de los $y \in Y$ es "difícil" de calcular $f^{-1}(y)$. La idea es que "difícil", en este contexto, significa "computacionalmente no factible", es decir, no factible usando los mejores algoritmos conocidos y el mejor hardware. Por el momento no se ha demostrado la existencia de funciones de dirección única, pero existen varias candidatas que se usan satisfactoriamente en la práctica y parecen comportarse como tales, tal como veremos en los apartados siguientes. El uso de funciones de dirección única como funciones de encriptación no es suficiente para poder construir criptosistemas que cumplan el requisito de confidencialidad, pues si se usa una función de dirección única para cifrar, entonces ni siquiera el destinatario legítimo del mensaje sería capaz de descifrarlo. Por eso es interesante el concepto de *función de dirección única con trampa*, que es una función de dirección única para la cual existe una información adicional que permite invertirla de modo eficiente. Basándose en estos conceptos, Diffie y Hellman introdujeron los llamados *criptosistemas de clave pública* o *criptosistemas asimétricos*, que consisten en una familia $\{f_k : \mathcal{M} \to \mathcal{C} | k \in \mathcal{K}\}$ de funciones de dirección única con trampa. Para cada $k \in \mathcal{K}$ debe de ser posible describir un algoritmo para calcular f_k tal que no sea factible obtener a partir de él un algoritmo para invertir f_k, a menos que se conozca la trampa correspondiente a k. Para usar el criptosistema, cada usuario A elige un $a \in \mathcal{K}$ aleatorio y publica en un directorio el "algoritmo de encriptación" E_A que calcula f_a, el cual constituye su *clave pública*. A partir de a también obtiene la trampa que permite invertir f_a mediante el "algoritmo de desencriptación" D_A, pero no la hace pública puesto que esto es lo que constituye su *clave privada*. Si otro usuario B quiere enviar a A un mensaje $m \in \mathcal{M}$, B busca en el directorio público la clave pública E_A de A y calcula $c = f_a(m) \in \mathcal{C}$, que es el criptotexto que envía a A. Dado que A es la única persona que conoce la trampa que permite invertir f_a, solamente A puede recuperar el mensaje m, mediante el algoritmo D_A. Es importante observar que los usuarios ya no necesitan intercambiar claves antes de comunicarse, con lo que los problemas antes mencionados en relación con la distribución y el manejo de las claves quedan automáticamente resueltos. Pero las ventajas de un criptosistema de clave pública van más allá, pues ahora sí son posibles las firmas digitales. Para ello suponemos que $\mathcal{M} = \mathcal{C}$ y si A quiere enviar a B el mensaje firmado m, lo que hace es enviarle, junto con m, la "firma" $s = f_a^{-1}(m)$. Entonces B usa la clave pública de A para calcular $f_a(s) = f_a(f_a^{-1}(m)) = m$. Es claro que solamente A puede haber calculado s a partir de m, pues sólo A tiene capacidad para invertir f_a y así B se convence de que el mensaje procede efectivamente de A. En este esquema no hay secreto pues A ha enviado a B el mensaje m sin cifrarlo. Si se quiere obtener simultáneamente confidencialidad y firma digital, lo que hace A es enviar a B $f_b(m)$ y $f_b(s)$.

RSA

El primer criptosistema de clave pública, y quizá el que más éxito ha tenido hasta la fecha, fue propuesto en 1978, dos años después de la publicación del artículo de Diffie y Hellman, por Rivest, Shamir y Adleman [38], razón por la cual es conocido con el nombre de RSA, que son las iniciales de sus inventores. Se puede mencionar aquí que tanto la criptografía de clave pública como RSA ya fueron descubiertos dos o tres años antes en Inglaterra, por científicos que trabajaban para los servicios secretos británicos, quienes mantuvieron estos hallazgos en secreto hasta hace pocos años [46]. Por esta razón, este hecho no pasa de ser una mera anécdota, sin influencia alguna en la historia de la criptografía.

El criptosistema RSA se basa en la hipótesis, no demostrada pero altamente plausible, de que, para n y e enteros positivos dados, donde n es, usualmente, el producto de dos primos grandes, la función $m \mapsto m^e \pmod{n}$ es de dirección única con trampa. La trampa que permite invertir fácilmente la función es, precisamente, el conocimiento de los factores primos de n, por lo que se puede decir que RSA está basado en la dificultad del problema de la factorización de números enteros. Cada usuario U de RSA construye su clave (o su software lo hace por él) de la manera siguiente. En primer lugar U construye un entero $n_U = p_U q_U$ multiplicando dos primos grandes p_U, q_U de, aproximadamente, el mismo tamaño pero que no estén demasiado próximos (para dificultar al máximo la factorización de n_U). A continuación, calcula $\phi(n_U) = (p_U - 1)(q_U - 1)$ y elige un entero e_U tal que $1 < e_U < \phi(n_U)$ y $\mathrm{mcd}(e_U, \phi(n_U)) = 1$. Finalmente, U calcula el inverso multiplicativo de e_U módulo $\phi(n_U)$, es decir, el único entero d_U tal que $1 < d_U < \phi(n_U)$ y $e_U d_U \equiv 1 \pmod{\phi(n_U)}$. Todos estos cálculos se pueden hacer con facilidad utilizando algoritmos conocidos y, en particular, el cálculo de d_U se puede realizar mediante el algoritmo de Euclides. Una vez completado este proceso, U hace público el par (n_U, e_U), que constituye su *clave pública*, pero se guarda para sí d_U, que es su *clave privada*. n_U recibe el nombre de *módulo*, e_U es el *exponente de encriptación* y d_U el *exponente de desencriptación*.

Para enviar un mensaje a U sólo hay que conocer su clave pública, si el texto claro es m (donde m es un entero menor que n_U), éste se encripta calculando:

$$c = m^{e_U} \pmod{n_U}$$

U, por su parte, descifra el mensaje haciendo uso de su clave privada y calculando:

$$c^{d_U} \pmod{n_U} = m$$

Como ya he indicado, se cree que la función de encriptación que proporciona el criptotexto c a partir de m es una función de dirección única con trampa. Si se conoce la factorización de n_U en producto de primos, es fácil invertir esta función y desencriptar, pues entonces se pueden obtener $\phi(n_U)$ y luego d_U de la misma forma que lo ha hecho U. Aunque no está demostrado, se

cree que criptoanalizar RSA tiene esencialmente el mismo grado de dificultad que factorizar el módulo. Por tanto, es muy importante tener una idea de la dificultad del problema de la factorización para poder saber hasta que punto RSA es un criptosistema seguro. Para ello hay que analizar la complejidad de los algoritmos de factorización conocidos.

La mayoría de los métodos de factorización modernos tienen como punto de partida –como tantas otras cuestiones aritméticas– una observación de Fermat: si conseguimos escribir el número a factorizar n como una diferencia de los cuadrados de dos enteros no consecutivos, tenemos automáticamente una escisión (expresión como producto de dos factores no triviales) de n. En tal caso se tiene que $n = a^2 - b^2 = (a + b)(a - b)$ y, de hecho, si n es impar toda escisión puede ser obtenida de esta forma, pues si $n = ab$, entonces $n = (\frac{1}{2}(a + b))^2 - (\frac{1}{2}(a - b))^2$. El problema de factorizar un entero impar se reduce pues a escribirlo como una diferencia de cuadrados. Pero esto puede ser muy difícil y el enfoque ingenuo consistente en buscar esta expresión por tanteo sólo funciona cuando $n = ab$ con a y b muy próximos a $\sqrt{n}$.

La idea de Fermat fue llevada un paso más adelante, en los años 20 del siglo pasado, por el matemático francés de origen ruso Maurice Kraitchik, quien se dio cuenta de que, en lugar de encontrar enteros a y b tales que $n = a^2 - b^2$, podría bastar encontrar una diferencia de cuadrados que fuese múltiplo de n, es decir, una congruencia de la forma $a^2 \equiv b^2 \pmod{n}$. Esto no garantiza obtener una escisión de n pues lo que significa es que n divide al producto $(a + b)(a - b)$, de modo que todo factor primo p de n divide a este producto y, por tanto, a $a + b$ o $a - b$. Si todos los divisores primos de n dividen a uno de estos factores no podemos obtener una escisión de n pero si los divisores primos de n están distribuidos entre los divisores de *ambos* factores, entonces n no divide a ninguno de los factores $a-b, a+b$ y así basta calcular el máximo común divisor de n y $a - b$ (usando el algoritmo de Euclides) para hallar un factor no trivial de n.

Si n es impar y divisible por al menos dos primos distintos, al menos la mitad de las posibles soluciones de la congruencia $x^2 \equiv y^2 \pmod{n}$ proporcionan una escisión de n. Por tanto, si se generan 10 congruencias de este tipo la probabilidad de que alguna de ellas produzca una escisión de n es superior al $99,9\%$. El problema de la factorización se reduce entonces a encontrar un método eficiente para generar congruencias de esta forma.

Se comienza eligiendo una *base de factores* $B = \{p_1, \ldots, p_k\}$, formada por primos pequeños, que pueden ser los primeros k primos para un valor conveniente de k. A continuación se genera un gran número de *residuos cuadráticos* $a_i^2 \pmod{n}$, los cuales nos proporcionan congruencias de la forma $a_i^2 \equiv c_i \pmod{n}$, donde c_i es el menor residuo no negativo de a_i^2 módulo n. La idea es conseguir que un gran número de estos residuos se pueda factorizar en producto de primos pertenecientes a B (se dice entonces que son B-*lisos*) pues entonces tomando un producto conveniente de dichos residuos podremos conseguir un número que es producto de potencias *pares* de primos de B y, por lo tanto, una congruencia de la forma buscada. Más concretamente, a cada residuo

B-liso $c_i = \prod_{j=1}^{k} p_j^{e_{ij}}$ se le asocia el vector $e_i = (e_{i1}, \ldots, e_{ik})$ cuyas componentes son las multiplicidades primas de c_i reducidas módulo 2, de modo que cada e_i es un vector del $\mathbb{Z}_2$-espacio vectorial $\mathbb{Z}_2^k$, donde $\mathbb{Z}_2 = \{0, 1\}$ denota el "cuerpo de dos elementos". En cuanto el número de residuos cuadráticos B-lisos exceda la dimensión k de este espacio, se tendrá que los vectores correspondientes serán linealmente dependientes y se puede hallar una relación de dependencia lineal entre ellos aplicando "eliminación gaussiana" (u otro método similar) a la matriz formada por dichos vectores. Si, para los vectores $e_1, \ldots, e_m$ correspondientes a los residuos $c_1, \ldots, c_m$ se ha obtenido la relación de dependencia lineal $\sum_{i=1}^{m} \alpha_i \cdot e_{ij} \equiv 0 \pmod 2$, con $\alpha_i \in 0, 1$, esto significa que $c = \prod_{i=1}^{m} c_i^{\alpha_i} = \prod_{j=1}^{k} p_j^{\sum_{i=1}^{m} \alpha_i \cdot e_{ij}}$ donde el exponente de cada primo p_j es par. Por tanto $c = b^2$ es un cuadrado y tomando $a = \prod_{i=1}^{m} a_i^{\alpha_i}$, esto nos proporciona la congruencia buscada $a^2 \equiv b^2 \pmod n$.

El problema es, entonces, generar los c_i de manera que estos residuos sean pequeños para que así aumente la probabilidad de que sean B-lisos y, además, hacer que el proceso de factorizar los c_i en producto de primos de B (en los casos en que sea posible) sea eficiente. Los distintos modos de hacerlo son, en algunos casos, muy sofisticados y dan nombre a distintos métodos de factorización modernos. El más potente de todos ellos recibe el nombre de *criba del cuerpo de números* ("Number Field Sieve", NFS) cuya complejidad se ha estimado heurísticamente (con una ligera simplificación que no explicaré) como:

$$O(e^{(64/9)^{1/3}(\log n)^{1/3}(\log\log n)^{2/3}})$$

Denotemos $L_n(\gamma; c) = O(e^{c(\log n)^{\gamma}(\log\log n)^{1-\gamma}})$ [24], donde c es una constante; un algoritmo con este tiempo de ejecución se dice también que tiene complejidad $L(\gamma)$. Los algoritmos de tiempo polinómico son los de complejidad $L(0)$ y los de tiempo exponencial los de $L(1)$. Los algoritmos de tiempo $L(\gamma)$ para algún $0 < \gamma < 1$ son intermedios entre ambos y se dicen de *tiempo subexponencial*. Por tanto, NFS es un algoritmo subexponencial de complejidad $L(1/3)$, mientras que otros algoritmos subexponenciales como la criba cuadrática tienen complejidad $L(1/2)$. De la fórmula anterior se deduce, tomando como referencia que la factorización de un módulo de RSA de 512 bits requirió, en 1999, 5,2 meses de trabajo a varios cientos de ordenadores, que factorizar un módulo de RSA de 1024 bits con los mismos recursos requiriría más de 3 millones de años [16]. A continuación mostraré un ejemplo de como funciona RSA con un módulo de este tamaño, usando *Mathematica*. En primer lugar, voy a generar una clave de RSA con un módulo de 1024 bits. Para simplificar los cálculos no utilizaré cribas ni división a la hora de buscar los primos (que es sólo cuestión de segundos) y basaré esta búsqueda en la función `PrimeQ` de *Mathematica*, la cual proporciona *primos probables*. Es *extremadamente improbable* que alguno de ellos no sea primo y, si uno quiere asegurarse más, puede utilizar después algún test de primalidad determinista (cf. [16] y el recién aparecido preprint [1] para una discusión de estas cuestiones). Al

generar la clave nos aseguramos de que $p-1$ y $q-1$ tienen un factor primo grande (para dificultar la factorización de n, aunque esta precaución es, muy probablemente, innecesaria [39]) y también de que p y q no estén demasiado próximos para que el módulo no se pueda factorizar por el método de Fermat. Los primos p y q se pueden descartar una vez generada la clave.

```
PrimoAleatorioFuerte[a_Integer, b_Integer] := Module[{primo = False, p},
     While[primo == False, p = Random[Integer, {a, b}];
       primo = PrimeQ[p] &&
PrimeQ[Last[FactorInteger[p - 1, FactorComplete -> False]][[1]]]];
     Return[p]]/; b-a >= 2^254

ClaveRSA[bits_Integer] :=
  Module[{a = 2^(Floor[bits/2] - 1), b = 2^(Floor[bits/2]) - 1, dif = 0,
        u, v, p, q, n, e, d, mcd = 0},
    p = PrimoAleatorioFuerte[a, b];
    u = Ceiling[2^(bits - 1)/p]; v = Floor[(2^bits - 1)/p];
    While[dif < 2^((bits - 8)/2),
            q = PrimoAleatorioFuerte[u, v]; dif = Abs[p - q]];
    n = p q;
    While[mcd != 1, e = Random[Integer, {3, n-2}];
     mcd = GCD[e, (p - 1)(q - 1)]]; d = PowerMod[e, -1, (p - 1) (q - 1)];
         Return[{n, e, d, p, q}]]/; bits >= 512
```

Ahora calculo una clave de 1024 bits y sólo muestro la clave pública:

```
{n,e,d,p,q} = ClaveRSA[1024];
{n,e}

{126383789532827951424138341857602546918322625029482706586039748152555267346246\
451913324577386722838172081565427114038996566558669329607413176228546493744196\
431346091042036133572117399732498069101315463456571831719216184246845971668647\
304683830741849459905336492688304399503896175531853969297611772814520047054 3,
370173291642522686212714866934420018303666532586070748175314322185830251118895\
372471218239077524723043053946768561179687106848657197028272929914913432097441\
725042498615527292841791186035928139719565885528212290858250061688489543529771\
281540941838273477725551150490832131781038312253309985467684512122153089 49}
```

Para poder cifrar mensajes, que ordinariamente estarán escritos en los caracteres alfanuméricos correspondientes a algún lenguaje natural, es necesario codificarlos –de forma transparente, sin propósitos criptográficos– en forma numérica (nótese que la palabra *codificar* rara vez se usa en criptografía, aunque es frecuente en las descripciones informales de ésta. La palabra *código* se suele reservar para los códigos correctores de errores o para los códigos usados para conseguir mayor eficiencia en el almacenamiento y la transmisión de la información. Los códigos, por tanto, tienen unos objetivos completamente distintos de los de los criptosistemas). Hay que tener

en cuenta que la función de encriptación actúa sobre enteros menores que el módulo n, por lo que si el mensaje no se puede codificar como un entero de este tamaño, será necesario dividirlo en bloques de tamaño conveniente. Yo voy a usar, para codificar el mensaje en forma numérica, el código ISO Latin-1, un código de 8 bits, extensión del código ASCII, que asigna valores numéricos comprendidos entre 0 y 255 a los caracteres usados por los lenguajes europeos, incluyendo signos de puntuación y caracteres con tilde. En *Mathematica*, las funciones `ToCharacterCode[]` y `FromCharacterCode[]` permiten asignar a cada carácter su código numérico y recíprocamente. Por ejemplo `ToCharacterCode["niño"] = {110, 105, 241, 111}` nos da los códigos de los cuatro caracteres que forman esta palabra (obsérvese que el código de la "ñ", 241, está fuera del rango del código ASCII, que sólo usa los códigos del 0 al 127). Recíprocamente, se tiene que `FromCharacterCode[{110, 105, 241, 111}] = niño`. Si ahora queremos codificar, no caracteres aislados sino un texto, lo más sencillo será asignar a cada caracter su código y considerar éste como un dígito en base 256, de modo que el texto estará representado numéricamente por el número que define, en base 256, la sucesión de dígitos correspondiente. Dado que, como ya he indicado, debemos usar números menores que n para codificar los mensajes y, teniendo en cuenta que, en nuestro caso, n tiene 1024 bits y se utilizan 8 bits para codificar cada caracter, se puede concluir que los bloques en que se debe dividir el mensaje han de tener, a lo sumo, 127 caracteres cada uno.

La codificación de un texto como un entero se hará entonces de la manera siguiente:

```
TextoAEntero[mensaje_String] := (ToCharacterCode[mensaje]).
   (256^Range[StringLength[mensaje] - 1, 0, -1])
```

y el paso de entero a texto será:

```
EnteroATexto[k_Integer] := FromCharacterCode[IntegerDigits[k, 256]]
```

Finalmente, la función de encriptación será:

```
RSAEnc[mensaje_String, e_Integer, n_Integer] :=
   PowerMod[TextoAEntero[mensaje], e, n]
```

y la que desencripta el criptotexto `c`:

```
RSADesenc[c_Integer, d_Integer, n_Integer] :=
   EnteroATexto[PowerMod[c, d, n]]
```

Usando la clave pública antes indicada y estas funciones he encriptado un mensaje `m` de no más de 127 caracteres, que puede tener algún interés para los aficionados a la gastronomía y cuyo criptotexto doy a continuación como un pequeño desafío al lector (aplicando el *principio de Kerckhoffs* [46, 48], lo único que oculto es la clave privada). He calculado:

```
c = RSAEnc[m, e, n]
```

```
912284155687731591682227631558634788613028608372689316390415433440018743554899\
774987982297813207929943833267479725884217551959153173391207727909380248402637\
840939359913487197605541300203049020632197180262484292645820438435844065622908\
212613996355573864772207460964952998335571420950036697002223768901494334 48
```

El desafío es, naturalmente, obtener `m` a partir del criptotexto `c` dado. Usando la función `RSADesenc` para este valor de `c` el resultado se obtiene en un segundo pero, claro está, para poder hacerlo hay que conocer `d` ...

El problema del logaritmo discreto

La primera forma concreta de criptografía de clave pública apareció ya en el artículo original de Diffie y Hellman, aunque no se trata de un criptosistema para la confidencialidad sino solamente de un protocolo para intercambiar claves secretas (para ser usadas en un criptosistema simétrico) que recibe el nombre de *Intercambio de claves de Diffie-Hellman*. Para ello partieron de la idea de que, dado un primo p y un entero g convenientemente elegidos, la función $x \mapsto g^x \pmod p$, donde x es un entero positivo, es de dirección única. Para precisar un poco más, recordemos que si G es un grupo y $g \in G$, el *orden* de g es el menor entero positivo n tal que $g^n = 1$ (donde 1 denota el elemento neutro, o identidad, de G; si no existe un tal n, se dice que el orden de g es infinito). En lo sucesivo supondremos que g tiene orden finito n, en cuyo caso $< g >= \{1, g, g^2, \ldots, g^{n-1}\}$ es un grupo de n elementos (se dice entonces que $< g >$ es el subgrupo cíclico de G generado por g; también se dice que $< g >$ tiene orden n pues se llama orden de un grupo finito a su número de elementos). Se puede identificar $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ con el grupo aditivo de las clases residuales $\pmod n$ y la aplicación $\mathbb{Z}_n \to < g >$ dada por $x \mapsto g^x$ es una biyección que satisface $g^{x+y} = g^x g^y$ o, en otras palabras, un isomorfismo entre el grupo aditivo $\mathbb{Z}_n$ y el grupo multiplicativo $< g >$. El isomorfismo inverso se suele denotar por $\log_g$ y si $h \in < g >$, $\log_g h$ recibe el nombre de *logaritmo discreto* (o índice) de h en la base g; obviamente se tiene que $\log_g(h_1 h_2) = \log_g h_1 + \log_g h_2$ para $h_1, h_2 \in < g >$. El *problema del logaritmo discreto* (PLD) en un grupo G es el problema de, dados $g, h \in G$, hallar $\log_g h$ si existe (es decir, si $h \in < g >$) y lo que Diffie y Hellman pensaron es que el cálculo del logaritmo discreto en el grupo multiplicativo $\mathbb{Z}_p^*$ de las clases residuales no nulas módulo un primo, no es factible si p está bien elegido. Por el contrario, el cálculo de g^x, dados $g \in G$ y x es muy fácil empleando el algoritmo conocido como exponenciación modular o exponenciación binaria (cf. [9, 23, 48]) y suponiendo que, como es bastante habitual, existe un algoritmo eficiente para efectuar la multiplicación en G algo que, desde luego, ocurre en caso de ser $G = \mathbb{Z}_p^*$. En otras palabras, la función $x \mapsto g^x$ es, en este caso, de dirección única. Obsérvese que, aunque G puede ser un grupo cualquiera, el PLD respecto de una base $g \in G$, de orden finito, tiene lugar en el subgrupo cíclico finito $< g >$ de G. Si G es finito, el orden n de g divide al orden de G por el *teorema de Lagrange* [11] y así, si G tiene orden primo, se deduce que G

es cíclico y cualquier elemento $g \neq 1$ de G es un generador y, por tanto, todo elemento de G tiene un logaritmo discreto en la base g.

El intercambio de Diffie-Hellman, en su versión más general, se puede describir entonces en la forma siguiente. A y B quieren compartir de forma segura una clave secreta, usando para ello canales de comunicación abiertos, accesibles a terceras personas. Para ello acuerdan (públicamente) usar un grupo finito G y un elemento $g \in G$. A continuación, A elige aleatoriamente un entero positivo k_A, tal que $1 < k_A < n$, donde n es el orden de g, calcula $g^{k_A} \in G$ y envía g^{k_A} a B. Por su parte, B realiza el proceso análogo, eligiendo un entero positivo k_B y enviando g^{k_B} a A. Finalmente, A calcula $(g^{k_B})^{k_A}$ y B calcula $(g^{k_A})^{k_B}$. Ambos obtienen así el mismo elemento, $g^{k_A k_B} \in G$ y ésta es la clave que acuerdan compartir. Si un adversario C ha observado el intercambio y quiere también obtener la clave secreta, el problema que tiene que resolver es el siguiente: *Dados* $g, g^{k_A}, g^{k_B} \in G$, *hallar* $g^{k_A k_B}$. Este es el *problema de Diffie-Hellman* y es evidente que si C es capaz de resolver el PLD en G también puede resolver el problema de Diffie-Hellman, pues entonces calcula k_A a partir de g^{k_A}, k_B a partir de g^{k_B} y, finalmente, $g^{k_A k_B}$. El recíproco no está demostrado pero se conjetura que ambos problemas son equivalentes por lo que se puede suponer que el intercambio de Diffie-Hellman será seguro si el PLD en G es intratable. Por esta razón (y también porque, como veremos, existen varios criptosistemas basados en él) trataremos de analizar un poco la dificultad del PLD.

En primer lugar, hay que observar que esta dificultad depende mucho del grupo G y del elemento $g \in G$ elegidos. En general, es necesario elegir un elemento g que tenga un orden muy grande pues, de lo contrario, bastaría calcular todas las potencias $g, g^2, \ldots g^n$ para resolver el PLD viendo cual de los elementos de esta lista es un $h \in < g >$ dado. Pero esto no basta, pues aunque el cálculo de todas estas potencias no sea factible, puede ocurrir que el PLD sea fácilmente resoluble como consecuencia de alguna característica especial del grupo G. Un ejemplo de esta situación se obtiene si tomamos $G = \mathbb{Z}_p$, el grupo aditivo de los enteros módulo p, donde p es un primo (que puede ser muy grande). Supongamos entonces que $g \in \mathbb{Z}_p$ es la base elegida (g puede ser cualquier elemento no nulo de $\mathbb{Z}_p$, pues al ser este un grupo de orden primo, $< g >= \mathbb{Z}_p$ y todo elemento $h \in \mathbb{Z}_p$ tiene un logaritmo discreto para la base g). Supongamos entonces dados $g, h \in \mathbb{Z}_p$. Para resolver el PLD tenemos que hallar $x \in \mathbb{Z}^+$ (unívocamente determinado módulo p) tal que $xg = h$ (obsérvese que la operación del grupo $\mathbb{Z}_p$ es aditiva). Esto puede hacerse de la manera siguiente: dado que g es no nulo, g es un elemento del grupo multiplicativo $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$. Por tanto, g tiene un inverso g^{-1} en $\mathbb{Z}_p^*$. Multiplicando por dicho inverso la ecuación $xg = h$ obtenemos que $x = hg^{-1}$. Dado que el inverso de $g \pmod p$ se puede calcular fácilmente mediante el "algoritmo de Euclides extendido", llegamos a la conclusión de que el PLD sobre $\mathbb{Z}_p$ es fácil y, por tanto, este grupo no puede ser utilizado para el intercambio de Diffie-Hellman. Por esta razón, el grupo que Diffie y Hellman propusieron para el intercambio

de claves no fue éste sino el grupo multiplicativo $\mathbb{Z}_p^*$. Este grupo también es cíclico pero su orden $p-1$ no es primo y, por tanto, no todos sus elementos $\neq 1$ son generadores. Los generadores g de $\mathbb{Z}_p^*$ son precisamente las raíces $(p-1)$-primitivas de la unidad módulo p, pues entonces $p-1$ es el menor entero positivo tal que $g^{p-1} \equiv 1 \pmod{p}$; también reciben el nombre de *elementos primitivos* de $\mathbb{Z}_p$. Para usar el intercambio de Diffie-Hellman hay que comenzar pues eligiendo un primo p y determinando un elemento primitivo de $\mathbb{Z}_p$; ambos serán públicos. Supongamos que tomamos:

```
p =
12349330662216929860432462757695088285483414664964639633770810729298902807288\
60299882148155719978503382321612338632762624681449591823873822797377372634716 7
```

que es un número de 512 bits. Utilizando *Mathematica* vemos que `PrimeQ[p] = True`, de modo que p es (muy probablemente) primo. El próximo paso es determinar un generador de $\mathbb{Z}_p^*$ y esto puede hacerse usando el programa siguiente (que, de hecho, encuentra el menor elemento primitivo; el uso del símbolo de Jacobi puede suprimirse sin problema, su única función es ahorrar un poco de tiempo pues es fácil ver que un elemento primitivo no puede ser un residuo cuadrático módulo p, es decir, la ecuación $g \equiv x^2 \pmod{p}$ no tiene solución en x si g es primitivo y, por tanto, el símbolo de Jacobi de g con respecto a p vale -1 [9]):

```
ElementoPrimitivo[p_?PrimeQ, opts___] :=
   Module[{a, i = 1, j = 1, k = 1,
     listadiv = (p - 1)/(First /@ FactorInteger[p - 1])},
      a = Aleatorio /. {opts} /. Options[ElementoPrimitivo];
While[k == 1, If[a, i = Random[Integer, {2, p - 2}], i++];
  If[JacobiSymbol[i, p] == -1, j = 1; k = 2;
    While[j <= Length[listadiv] && k != 1,
      k = PowerMod[i, listadiv[[j]], p]; j++]]; i] /; p > 2
Options[ElementoPrimitivo] = {Aleatorio -> False};
```

Para nuestro primo p vemos que `ElementoPrimitivo[p] = 5`, lo que nos indica que 5 es el menor elemento primitivo de $\mathbb{Z}_p^*$ y podemos usarlo para el intercambio de Diffie-Hellman. Alternativamente, podemos usar la opción de elegir un elemento primitivo aleatorio:

```
g = ElementoPrimitivo[q, Aleatorio -> True]

31182394246839957790549318834051730490201618062635924220504795597596835095926\
68168678465891933501221237948570719143209787600964663494615096986622291436312
```

Los valores de p y g se hacen públicos y ahora A calcula:

```
kA = Random[Integer, {2, p - 2}]

11045931547918911947131241887168871227668070717022527699452781667111105608326\
57471865931367227027805609122324836267155630005472210580285592936619474693802 3
```

y, B, por su parte:

```
kB = Random[Integer, {2, p - 2}]

619683430581460210933352861429797347026980771790783893720514061670035855432 76\
412649011261405358151849957242007725214060918923040680772673052465831060589 24
```

A continuación, A calcula, y envía a B:

```
cA = PowerMod[g, kA, p]

101367733429321822605369215252695000952055479203564979168127811161238746686 32\
750374363330524417400781301124929658163488655937884256558679412607437094290 533
```

y B, por su parte, calcula y envía a A:

```
cB = PowerMod[g, kB, p]

411548364041080509073102211787841838047911749825748909530223552413328598056 03\
325937756303758096687103478562706057313835825375540818225609736987295710745 47
```

Finalmente, la clave secreta que compartirán A y B es:

```
PowerMod[cB, kA, p] = PowerMod[cA, kB, p] =

103100550224562304776782895002444091087158152264955739572016699478521410687 88\
043407006451368091717046437843807481143290276744426945463896275193415138389 037
```

Si queremos evaluar la dificultad del PLD sobre $\mathbb{Z}_p^*$, hay que tener en cuenta que existen dos grandes clases de algoritmos para resolverlo: los genéricos, que se aplican a cualquier grupo sin explotar propiedades especiales de sus elementos y los de tipo especial, que utilizan propiedades adicionales de la forma en que está representado el grupo. Todos los logaritmos discretos funcionan esencialmente en grupos cíclicos, los cuales, estructuralmente, están determinados por su orden ("dos grupos cíclicos son isomorfos si y sólo si tienen el mismo orden"). La diferencia anterior no reside, por tanto, en la estructura del grupo sino en la forma en que están dados sus elementos. Hemos visto que el PLD en el grupo aditivo de $\mathbb{Z}_p$ es fácil, puesto que el algoritmo de Euclides, lo resuelve en tiempo polinómico. Pero este algoritmo se puede utilizar en este caso porque el grupo tiene una estructura mucho más rica, que es la de cuerpo. Si pasamos al grupo multiplicativo $\mathbb{Z}_p^*$, que es donde se plantea el intercambio de Diffie-Hellman, la situación ya no es, ni mucho menos tan sencilla y no se conoce ningún algoritmo de tiempo polinómico para resolver el PLD. Existe una reducción a través del llamado *algoritmo de Pohlig-Hellman* [4, 24, 29], que permite esencialmente reducir el PLD en un grupo de orden n al PLD en grupos cuyos órdenes son los factores primos de n (la idea es resolver el PLD trabajando módulo cada uno de esos factores primos y luego combinar los resultados obtenidos mediante el *teorema chino de los restos*). Por tanto,

si el orden $p-1$ de $\mathbb{Z}_p^*$ es sólo divisible por primes pequeños, el PLD será fácil en este grupo y, en consecuencia, el intercambio de Diffie-Hellman no será seguro. Así, hay que asegurarse de que el primo elegido p tiene la propiedad de que $p-1$ tiene al menos un factor primo "grande". En el caso del primo del ejemplo anterior vemos que:

```
FactorInteger[p - 1]
```

```
{{2, 1}, {3, 2}, {6860739256787183255795812643163937936379674813869244240983 78\
37384993904484936683326786008651109916854573422907701820145815636088434659656 8\
220765207019287, 1}}
```

de modo que $p-1$ tiene un factor primo grande, cuyo tamaño está próximo al de p, pues tiene 153 dígitos decimales y 508 bits (cosa que no ha ocurrido por casualidad sino por haberlo buscado de esta forma, eligiendo primos aleatoriamente hasta dar con uno adecuado). Para determinar si el tamaño del primo p es adecuado hay que examinar los algoritmos conocidos para resolver el PLD en $\mathbb{Z}_p^*$. Combinados con Pohlig-Hellman, los algoritmos genéricos que se conocen (*paso enano-paso gigante* de Shanks, *método rho* de Pollard y *método de los canguros* –también llamado *método lambda*– de Pollard, cf. [4, 9, 29, 34]) tienen complejidad $O(\sqrt{q}) = O(e^{\frac{1}{2}\log q})$, donde q es el mayor primo que divide al orden de G, lo cual significa que son algoritmos exponenciales. Si para un grupo G sólo existen algoritmos de este tipo, entonces el PLD en dicho grupo es difícil siempre que el orden sea divisible por, al menos, un primo grande (de unos 160 bits sería suficiente por el momento, aunque para tener criptografía muy fuerte sería preciso tomarlo más grande). Pero esto no es suficiente para asegurar la dificultad del PLD sobre $\mathbb{Z}_p^*$, pues para este grupo existen algoritmos especiales llamados *algoritmos de cálculo del índice*, que son más rápidos que los algoritmos exponenciales antes mencionados puesto que son subexponenciales. Estos algoritmos son muy similares a los mejores algoritmos existentes para factorizar enteros, en los cuales se inspiran. Como en el caso de la factorización, se parte de una base de factores $\{p_1, \ldots, p_k\}$, donde $p_1, \ldots, p_k$ son primos pequeños vistos como elementos de $\mathbb{Z}_p^*$. A continuación se calcula $g^t \pmod p$ para varios valores de t y cada uno de los números así obtenidos se intenta escribirlo como un producto de los primos de la base de factores. Cuando esto no es posible se descarta g^t, pero si $g^t \equiv \prod_{j=1}^{j=k} p_j^{a_j} \pmod p$, entonces se tiene la congruencia:

$$t \equiv \sum_{j=1}^{k} a_j \log_g p_j \pmod{p-1}$$

Una vez que se han obtenido suficientes ecuaciones de este tipo, se pueden resolver para obtener los valores de $\log_g p_j$, para $j = 1, \ldots, k$. En la segunda parte del proceso, se calcula el logaritmo discreto $\log_g h$ de la manera siguiente. Se toman enteros aleatorios s hasta que $g^s h$ se pueda escribir como un producto

de elementos de la base de factores: $g^s h \equiv \prod_{j=1}^{k} p_j^{b_j} \pmod{p}$ lo cual, tomando logaritmos, nos da:

$$\log_g h \equiv -s + \sum_{j=1}^{k} b_j \log_g p_j \pmod{p-1}$$

Por supuesto, hay que disponer de un método para generar eficientemente las relaciones anteriores y el más potente de ellos es, de nuevo, la criba del cuerpo de números cuya complejidad es subexponencial y se puede estimar, en este caso, como $O(e^{(64/9)^{1/3}(\log p)^{1/3}(\log\log p)^{2/3}})$. Se estima que resolver el PLD con este método es un poco más difícil que factorizar un número compuesto del mismo tamaño que p. Como ya he indicado, los módulos de RSA más grandes que se han factorizado con NFS tienen 512 bits (y la factorización de uno de 576 bits parece inminente). En el PLD se ha avanzado menos –probablemente, porque se le dedican menos recursos que a la factorización–, aunque si se ha resuelto el PLD sobre $\mathbb{Z}_p^*$ para un primo p de 120 dígitos decimales. También, el PLD por el cual Kevin McCurley ofreció en [28] un premio de 100 dólares, ha sido resuelto. En este caso, el primo p tenía 129 dígitos decimales pero era de una forma especial que permitía usar una versión más rápida de la criba del cuerpo de números que la que se puede aplicar a primos arbitrarios. A la vista de todo esto, es imprescindible tomar p, como mínimo, de 512 bits (y asegurarse de que $p-1$ tiene un factor primo de, al menos, 160 bits). Estas condiciones las cumple el ejemplo anterior, aunque para tener un nivel de seguridad razonable habría que tomar p un poco más grande.

El intercambio de Diffie-Hellman no permite intercambiar información de forma confidencial, sino sólo compartir una clave secreta que se genera durante el proceso. Sin embargo, existe una modificación que permite obtener un auténtico criptosistema de clave pública, que recibe el nombre de *criptosistema de El Gamal* y funciona de la manera siguiente. Se elige un grupo finito G y un elemento $g \in G$ (en la propuesta original de El Gamal G era el grupo multiplicativo $\mathbb{Z}_p^*$ y g un generador de G, aunque más adelante veremos que ahora existe una alternativa muy importante). Se supone, además, que cada texto claro está representado por un elemento de G. Cada usuario A elige como clave privada un entero aleatorio a, tal que $1 < a < p-1$ y calcula y publica $g^a \in G$, que es su clave pública. Ahora, si el usuario A quiere enviar al usuario B, cuya clave pública es $h_B = g^b$ (donde b es la clave privada de B que A, por supuesto, no conoce), el mensaje $m \in G$, hace lo siguiente:

- A genera un entero aleatorio k, tal que $0 < k < p-1$, y calcula g^k y $h_B^k m$.
- A envía a B el criptotexto, formado por el par $(c,d) = (g^k, h_B^k m)$.
- B recupera el texto claro a partir del criptotexto, calculando en primer lugar $(g^k)^b = c^b$ (puede hacerlo puesto que B conoce su

propia clave privada b). Ahora bien, $c^b = (g^k)^b = (g^b)^k = h_B^k$ y así B recupera m calculando $m = d(c^b)^{-1}$.

La idea del criptosistema de El Gamal consiste en que A le envía a B el mensaje m disfrazado mediante la multiplicación con h_B^k. Para que B pueda recuperar el mensaje, A le envía, además, el elemento $c = g^k$, que puede ser usado para quitarle el disfraz al mensaje, pero sólo B puede hacerlo, porque para ello hay que conocer b. Obsérvese que el problema de romper el criptosistema de El Gamal es similar al de romper el intercambio de Diffie-Hellman. Si alguien es capaz de resolver el PLD en G, puede calcular la clave privada b de B a partir de su clave pública h_B y, por tanto, puede recuperar m a partir del criptotexto de la misma manera que B. Tampoco en este caso está demostrado que esta sea la única forma de criptoanalizar El Gamal, pero se conjetura que para obtener g^{kb} a partir de g^k y g^b hay, esencialmente, que resolver el PLD. Una idea similar a la subyacente al criptosistema de El Gamal es la que se usa para obtener el *esquema de firmas digitales de El Gamal*. Una variante de este esquema es el llamado *Algoritmo de firmas digitales* (Digital Signature Algorithm, DSA), que es la base del *Digital Signature Standard* (DSS) que fue adoptado como estándar para firmas digitales en los Estados Unidos. Todos estos protocolos están basados en la dificultad de resolver el PLD, pero no incluyo aquí los detalles por brevedad, cf. [4, 19, 24, 25, 29, 47, 48, 49].

Curvas elípticas

Los criptosistemas que hemos visto, basados en la dificultad del PLD sobre $\mathbb{Z}_p^*$, son perfectamente viables y se usan en la práctica. Sin embargo, la existencia de algoritmos subexponenciales para el PLD en este grupo hace que haya que usar primos p relativamente grandes, lo que hace que los criptosistemas basados en este problema, como el de El Gamal, sean poco eficientes y requieran cantidades considerables de memoria. Por eso se observó que sería interesante poder basar estos criptosistemas en grupos para los que sólo existiesen algoritmos genéricos que, como ya he indicado, son exponenciales y, por tanto, hacen que se requieran claves mucho más pequeñas para un nivel de seguridad dado. De ahí la propuesta realizada, independientemente, en 1985, por Neal Koblitz y Victor Miller: *utilizar criptosistemas basados en el PLD sobre los grupos aditivos de las curvas elípticas* [22, 31]. Al principio, esta idea parecía una curiosidad teórica, que quizá llegaría a ser de utilidad práctica en un futuro lejano. Pero tal como dice Koblitz [24, p. 131]: "como suele ocurrir en criptografía, el futuro lejano llegó rápidamente". Hoy en día los criptosistemas basados en curvas elípticas se usan abundantemente en la industria y el comercio y, lo que es más, existe la impresión de que es probable que su importancia crezca en el futuro. Para describir estos criptosistemas, haré una breve introducción a las curvas elípticas.

Recordemos que la característica de un cuerpo es el menor entero positivo p –si existe– tal que $p1 = 0$. Si no existe un tal p se dice que el cuerpo tiene

característica cero, como es el caso de $\mathbb{Q}, \mathbb{R}, \mathbb{C}$, y si la característica es > 0, entonces es siempre un número primo, por ejemplo, $\mathbb{F}_p = \mathbb{Z}_p$ (con la suma y multiplicación usuales módulo p, donde p es un primo) tiene característica p. Una curva elíptica $E(\mathbb{F})$ sobre un cuerpo $\mathbb{F}$ de característica $\neq 2, 3$, es el conjunto de los puntos $(x, y) \in \mathbb{F}^2$ que satisfacen una ecuación de la forma:

$$E : y^2 = x^3 + ax + b$$

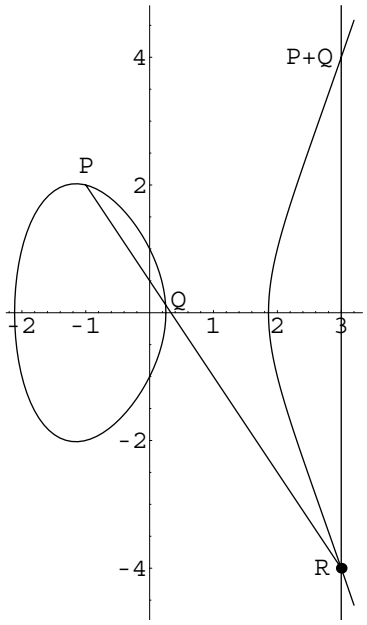
donde $a, b \in \mathbb{F}$ y $4a^3 + 27b^2 \neq 0$, junto con un punto adicional $\mathcal{O}$ llamado *punto del infinito*. $-(4a^3 + 27b^2)$ es el *discriminante* del polinomio cúbico $x^3 + ax + b$ que define la curva y la condición de que no se anule es equivalente a que este polinomio no tenga ceros múltiples o, lo que es lo mismo, a que la curva definida por la ecuación anterior sea *no singular* o *lisa* (no existen puntos de la curva, considerada sobre una clausura algebraica de $\mathbb{F}$, en los que ambas derivadas parciales de la cúbica de su ecuación implícita se anulan simultáneamente). Hay otras definiciones, más generales, del concepto de curva elíptica (cf. [42, 8]), pero ésta será suficiente para nuestros propósitos.

La ecuación $y^2 = x^3 + ax + b$ se llama la *ecuación de Weierstrass* de la curva y las curvas elípticas sobre cuerpos $\mathbb{F}$ de característica 2 o 3 se definen mediante una ecuación de Weierstrass ligeramente más complicada. No describiré este caso pues nuestro interés se centrará en curvas definidas sobre el cuerpo de p elementos, $\mathbb{F}_p$, donde p será un primo grande y, desde luego, $p > 3$.

Lo que hace que las curvas elípticas sean un objeto enormemente interesante en criptografía es que tienen estructura de grupo abeliano (o conmutativo). Esta estructura se puede definir mediante unas fórmulas, que son las que habrá que manejar en los cálculos criptográficos, pero que no permiten atisbar como surgió la idea. Para ver esto es conveniente representar una curva elíptica sobre $\mathbb{R}$; consideremos, por ejemplo, la curva $F(\mathbb{R})$ definida por la ecuación de Weierstrass $F : y^2 = x^3 - 4x + 1$:

```
<<Graphics`ImplicitPlot`

F[x_, y_] := y^2 - x^3 + 4x - 1;
ImplicitPlot[F[x, y] == 0, {x, -2.3, 3.2},
  Epilog -> {Line[{{-1, 2}, {3, -4}}], Line[{{3, -5}, {3, 5}}],
   Text["P", {-1, 2.2}], Text["Q", {0.45, 0.2}],
   Text["R", {2.8, -4}], Text["P+Q", {2.6, 4}],
      AbsolutePointSize[4], Point /@ {{-1, 2}, {1/4, 1/8}, {3, 4}, {3, -4}}}];
```



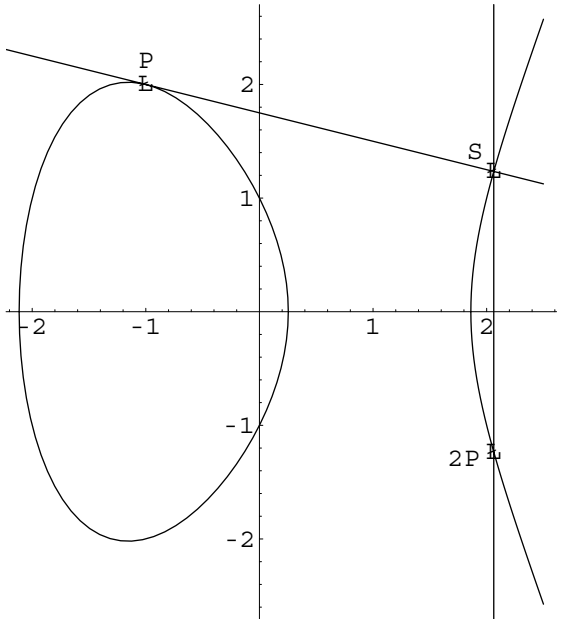
Vemos que una recta como la que une los puntos $P = (-1, 2)$ y $Q = (1/4, 1/8)$, corta a la curva en un tercer punto que, en este caso es, como se puede calcular fácilmente $R = (3, -4)$. Es fácil demostrar que siempre ocurre así: si el cuerpo sobre el que está definida la curva es algebraicamente cerrado (por ejemplo, $\mathbb{C}$), todas las rectas cortarán a la curva en tres puntos; si no lo es –como es el caso de $\mathbb{Q}$ o $\mathbb{R}$– esto ya no se verifica necesariamente, pero sigue siendo cierto que si una recta corta a la curva en dos puntos, entonces la corta en un tercer punto adicional único. Esto es precisamente lo que ocurre en el ejemplo anterior. Si consideramos la curva definida por la ecuación F como una curva elíptica $F(\mathbb{Q})$ sobre $\mathbb{Q}$ ya sabemos que, dado que los puntos P y Q de la curva tienen coordenadas racionales, el punto R también va a tenerlas. Pues bien, esto es todo lo que se utiliza para definir la estructura de grupo en la curva: Si $R = (x, y)$ es el tercer punto de intersección de la recta PQ con la curva, entonces se define la suma $P + Q$ como el punto de coordenadas $P+Q = (x, -y)$, es decir, $P+Q$ es el simétrico de R con respecto al eje de las x (nótese que el grafo de la curva es simétrico con respecto a este eje). Tomemos ahora, sobre la misma curva, la recta tangente en el punto P (obsérvese que, como consecuencia de que la curva es no singular, existe una tangente única bien definida en cada punto). Calculamos la tangente en P:

```
t[x_, y_] := Derivative[1,0][F][-1,2](x+1) + Derivative[0,1][F][-1, 2](y-2);
 Simplify[t[x, y] == 0]
```

```
x + 4 y == 7
```

Ahora representamos la curva y la tangente que acabamos de calcular:

```
ImplicitPlot[F[x,y] == 0, {x,-2.3, 2.5},  Epilog ->
  {Line[{{-2.3,2.325}, {2.5,1.125}}], Line[{{33/16,-4}, {33/16,4}}],
  Text["P", {-1,2.2}],  Text["S", {1.9,1.4}], Text["2P", {1.8,-1.3}],
   {AbsolutePointSize[4], Point /@ {{-1,2}, {33/16,79/64}, {33/16,-79/64}}}}];
```



Observando la figura parece evidente que la tangente en P sólo corta a la curva elíptica en dos puntos: el propio P y $S = (33/16, 79/64)$, en contradicción con lo afirmado antes. Pero las apariencias engañan y los puntos de intersección son en realidad 3, como se puede comprobar usando de nuevo *Mathematica* para calcularlos:

```
Solve[{F[x, y] == 0, t[x, y] == 0}]
```

```
{{y -> 79/64, x -> 33/16}, {y -> 2, x -> -1}, {y -> 2, x -> -1}}
```

Vemos que el punto P aparece dos veces y esto se debe a que -1 es una raíz doble de la ecuación cúbica resultante de eliminar la variable y en el sistema formado por las dos ecuaciones anteriores. Es decir, la propiedad anteriormente enunciada de que si una recta en $\mathbb{F}^2$ corta a la curva en dos puntos, entonces la corta en un tercer punto adicional único, se debe entender en el sentido de *"contando los puntos de intersección con su multiplicidad correspondiente"*. El punto de tangencia tiene, en este caso, un contacto de orden 2 con la curva (la tangente puede interpretarse también como el "límite" de las secantes determinadas por dos puntos que se van aproximando hasta coincidir) y, por eso, de acuerdo con la regla anteriormente indicada para la suma de dos puntos,

tenemos que en este caso $2P = (-1, 2) + (-1, 2) = (33/16, -79/64)$. La regla geométrica usada para definir la suma sirve también si uno de los puntos es el punto del infinito $\mathcal{O}$. El método más apropiado para trabajar con este punto es ver la curva elíptica como una curva en el plano proyectivo sobre $\mathbb{F}$, y trabajar con las coordenadas proyectivas correspondientes. Se puede ver entonces que la tangente en $\mathcal{O}$ es la recta del infinito, que tiene un contacto de orden 3 con la curva en $\mathcal{O}$. Por tanto, el punto del infinito es un punto de inflexión de la curva (el que la tangente tenga un contacto de orden ≥ 3 es la definición algebraica de punto de inflexión; en el caso de las curvas elípticas, que son cúbicas, la multiplicidad de la intersección tiene que ser precisamente 3). Pero si nos limitamos a ver la curva en el plano afín, como hemos hecho hasta ahora, el punto del infinito no se puede visualizar directamente, sino que cabe imaginarlo situado "infinitamente al norte", de modo que todas las rectas verticales cortan a la curva en ese punto. Así, la regla anterior para la suma es consistente con el hecho de que $\mathcal{O}$ va a ser el elemento neutro o identidad del grupo, pues se tiene que $\mathcal{O} + \mathcal{O} = \mathcal{O}$ y, más generalmente, que $P + \mathcal{O} = \mathcal{O} + P = P$. Para ello basta observar que, si $P = (x, y)$, entonces la recta $\mathcal{O}P$ es la recta vertical que pasa por P y el tercer punto de intersección de esta recta con la curva es, en virtud de la simetría de ésta con respecto al eje de las x, el punto de coordenadas $(x, -y)$. Por otra parte, si P y Q tienen la misma coordenada x, entonces $P + Q = \mathcal{O}$, de modo que $Q = -P$, es decir, el opuesto de $P = (x, y)$ es el punto $P = (x, -y)$. Usando la descripción dada de la suma se pueden deducir fácilmente fórmulas analíticas que proporcionan las coordenadas de $P + Q$ en función de las de P y Q [4, 20, 24, 29, 45, 48]. Si la curva $E(\mathbb{F})$ está dada por la ecuación de Weierstrass $E : y^3 = x^2 + ax + b$, estas fórmulas son las siguientes:

Sea $P = (x_1, y_1) \in E(\mathbb{F})$. Entonces $-P = (x_1, -y_1)$ y si $Q = (x_2, y_2) \in E(\mathbb{F})$ con $Q \neq -P$, entonces $P + Q = (x_3, y_3)$, con:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

donde $\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{si } P \neq Q, \\ \frac{3x_1^2 + a}{2y_1} & \text{si } P = Q. \end{cases}$

Aunque las representaciones gráficas anteriores son para curvas elípticas sobre $\mathbb{R}$, la definición dada de la suma y las fórmulas anteriores para la suma de dos puntos, que son puramente algebraicas, son válidas sobre un cuerpo arbitrario $\mathbb{F}$ (de característica $\neq 2, 3$ en nuestro caso). La suma está caracterizada por ser $\mathcal{O}$ el elemento neutro junto con el "principio de la secante-tangente": *la suma de tres puntos de la curva es $\mathcal{O}$ si y sólo si los tres puntos están alineados*. Es evidente que la suma es una operación interna en $E(\mathbb{F})$ que es, además, conmutativa, que $\mathcal{O}$ es un elemento neutro para la suma y que cada $P \in E(\mathbb{F})$ tiene un simétrico, $-P$. La única parte no trivial de la demostración de que $E(\mathbb{F})$ es un grupo abeliano es probar la asociatividad de la suma. Esto

se puede hacer directamente, usando las fórmulas anteriores, aunque es algo extremadamente tedioso. Otras formas de hacerlo son: usando la teoría algebraica de divisores en curvas como en [6, 42] o bien usando una demostración geométrica basada en el teorema de Bezout sobre intersecciones de curvas [20, 45].

Las curvas elípticas sobre $\mathbb{Q}$ ocupan un lugar central en la *geometría algebraica aritmética* y su estudio está conectado a muchos problemas importantes de teoría de números, habiendo jugado un papel fundamental en la demostración del *teorema último de Fermat* (cf. [14]). Aunque las curvas elípticas de interés criptográfico son principalmente las definidas sobre cuerpos finitos, las curvas elípticas sobre $\mathbb{Q}$ también juegan un papel en este aspecto, porque ciertos ataques criptoanalíticos se basan en el "levantamiento" de puntos de curvas elípticas sobre cuerpos finitos a puntos de curvas elípticas sobre $\mathbb{Q}$ [18, 43, 44]. En relación con el grupo de una curva elíptica sobre $\mathbb{Q}$, hay un resultado fundamental que fue demostrado por L. Mordell en 1922: *el grupo abeliano $E(\mathbb{Q})$ es finitamente generado* (más generalmente, por el *"teorema de Mordell-Weil"*, el grupo de una curva elíptica sobre un "cuerpo de números" es finitamente generado y todavía existen versiones más generales de este teorema [8, 42]). Esto significa que existe un número finito de puntos en la curva tal que cualquier otro punto puede ser obtenido a partir de ellos aplicando la regla de la secante-tangente un número finito de veces. De modo más preciso, se tiene que $E(\mathbb{Q}) \cong E_{tors} \times \mathbb{Z}^r$, donde el "grupo de torsión" E_{tors}, formado por los puntos de orden finito (o puntos de torsión) es finito y el número r (número de generadores de la "parte infinita") se llama *rango* de la curva. El cálculo del grupo de torsión de $E(\mathbb{Q})$ es fácil mediante un algoritmo que se deduce del *teorema de Nagell-Lutz* según el cual si $P = (x, y)$ es un punto de torsión distinto de $\mathcal{O}$, entonces las coordenadas x e y son números enteros y, además, si $y \neq 0$, entonces y^2 divide a $4a^3 + 27b^2$ [42, 8]. Además, existe un teorema muy profundo de B. Mazur que muestra que sólo hay 15 grupos (la mayoría de ellos cíclicos) que pueden ser el grupo de torsión de $E(\mathbb{Q})$ (y cada uno de ellos es el grupo de alguna curva $E(\mathbb{Q})$) [42]. Además de los puntos de torsión, puede haber otros puntos con coordenadas enteras pero, por un teorema de Siegel, se sabe que el número de estos puntos es también finito [42]. Una vez que se ha calculado E_{tors}, lo que resta para determinar $E(\mathbb{Q})$ es calcular el rango r. Sin embargo, este cálculo es difícil y, de hecho, no se conoce un algoritmo para llevarlo a cabo, aunque sí hay un "algoritmo conjetural" debido a J. Cremona [8, 10]. Existen numerosos problemas abiertos relacionados con el rango de las curvas elípticas racionales. Por ejemplo, se cree que para cada $r \geq 0$ existe una curva de rango r, pero esto no está probado. En la búsqueda de curvas de rango elevado, el récord, por el momento, está en una curva de rango ≥ 24 encontrada por R. Martin y W. McMillen en 2000 [41]. Relacionada con el rango está también la *conjetura de Birch y Swinnerton-Dyer* que es, sin duda, uno de los problemas abiertos más importantes en aritmética y está incluido entre los 8 *"problemas del milenio"*, por la solución de cada uno de los cuales el Instituto Clay ofrece un premio de un millón de dólares [50].

Una magnífica exposición sobre los problemas abiertos más importantes de la geometría algebraica aritmética se encuentra en las notas de Alice Silverberg [41].

En el ejemplo anteriormente considerado, la curva $F(\mathbb{Q})$, con $F : y^2 = x^3-4x+1$, tiene grupo de torsión trivial (una pequeña paradoja terminológica: el único punto de orden finito de $F(\mathbb{Q})$ es el punto del infinito (!)) y rango 2, siendo $\{(0,1),(-1,2)\}$ un conjunto generador del grupo [10]. Sin embargo, existen curvas elípticas racionales que sólo tienen un número finito de puntos (todos los cuales son, necesariamente, de torsión). Consideremos, por ejemplo, la curva $G(\mathbb{Q})$, con $G : y^2 = x^3 + 1$. Repitiendo el cálculo antes realizado por el método de la secante-tangente vemos que si tomamos $P = (2,3)$ el cual es, obviamente, un punto de $G(\mathbb{Q})$, la tangente en este punto es la recta de ecuación $y = 2x - 1$, la cual corta a la recta en P con multiplicidad 2 y también la corta en $(0,-1)$, de modo que $2P = (0,1)$. Si ahora repetimos el proceso para calcular $4P = 2P + 2P$, vemos que la tangente en $(0,1)$ es la recta $y = 1$ que tiene una intersección de multiplicidad 3 con la curva en el punto $(0,1)$. Esto significa que 2P es un punto de inflexión y también que $2P + 2P = -2P$, es decir, $6P = 3(2P) = \mathcal{O}$ y $4P = (0,-1)$. De esto también se deduce que P es un punto de orden 6 y que la curva contiene un subgrupo de orden 6 generado por P. Como se puede observar fácilmente, los puntos de inflexión son precisamente los puntos de 3-torsión, es decir, aquéllos cuyo orden es un divisor de 3. En este caso, hay exactamente 3 de estos puntos: $\mathcal{O}$, $2P$ y $4P$ (la curva $F(\mathbb{Q})$ antes considerada sólo tiene como punto de inflexión $\mathcal{O}$, aunque la curva real $F(\mathbb{R})$ también tiene 3 puntos de inflexión). También hay un punto de orden 2, $3P = (-1,0)$ (es inmediato que un punto de la curva tiene orden 2 si y sólo si su coordenada y es 0). Además, vemos que $5P = -P = (2,-3)$ y así este subgrupo de orden 6 está formado por $\{\mathcal{O},(2,3),(0,1),(-1,0),(0,-1),(2,-3)\}$. Ya en el siglo XVIII, Euler demostró (usando "descenso infinito", cf. [12, p. 533]) que $G(\mathbb{Q})$ no tiene más puntos que éstos, de modo que el grupo de la curva es cíclico de orden 6 (isomorfo a $\mathbb{Z}_6$) y, por tanto, la curva tiene rango 0.

A partir de ahora voy a considerar curvas elípticas sobre cuerpos finitos y me centraré en el caso de las definidas sobre un cuerpo primo $\mathbb{F}_p$, con $p > 3$ (aunque también se usan en criptografía las curvas sobre los cuerpos finitos de característica 2 y, por otra parte, la teoría general es prácticamente la misma si se consideran curvas sobre un cuerpo finito $\mathbb{F}_q$, donde $q = p^k$, con $p > 3$). Obviamente, el grupo $E(\mathbb{F}_p)$ es siempre finito (puesto que $\mathbb{F}_p^2$ lo es). Más aun, por un teorema de Cassels [9, p. 286], el grupo $E(\mathbb{F}_p)$ es siempre de la forma $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, donde n_1 divide a n_2 y a $p-1$. Para hacernos una idea de cómo es este grupo vamos a empezar por considerar un cuerpo muy pequeño, $\mathbb{F}_{2003}$, y la curva elíptica $H(\mathbb{F}_{2003})$ definida por la ecuación de Weierstrass

$$H : y^2 = x^3 + 3x + 1$$

Para generar los puntos de la curva podemos calcular los valores de $x^3 + 3x + 1 \pmod{2003}$ cuando x recorre $\mathbb{F}_{2003}$ y ver cuales de ellos son residuos

cuadráticos módulo 2003 (es decir, cuadrados en $\mathbb{F}_{2003}$), lo que se puede hacer calculando su símbolo de Legendre (o de Jacobi) [23] que debe ser distinto de -1. Si $x_0^3 + 3x_0 + 1 \pmod{2003}$ es un residuo cuadrático no nulo, tendrá dos raíces cuadradas en $\mathbb{F}_p$, que nos proporcionarán las coordenadas y de los dos puntos de la curva correspondientes a la coordenada x_0 (éstos son los dos puntos distintos de $\mathcal{O}$ en los que la recta $x = x_0$ corta a la curva). Usando *Mathematica* podríamos hacer:

```
<< NumberTheory`NumberTheoryFunctions`

X = Select[Range[2003]-1,JacobiSymbol[#^3+3 #+1,2003] != -1&];
Simetrico[P_List,p_Integer]:= {P[[1]],Mod[-P[[2]],p]};
L1={#,SqrtMod[Mod[#^3+3 #+1,2003],2003]}&/@X; L2=Simetrico[#,2003]&/@L1;
LH = Union[{{Infinity,Infinity}},L1,L2];
```

Ahora `LH` es una lista con todos los puntos de $H(\mathbb{F}_p)$, incluyendo el del infinito que he representado en la forma $\mathcal{O} = \{\infty, \infty\}$ (o también, {Infinity, Infinity} en la terminología de *Mathematica*). El orden de la curva es:

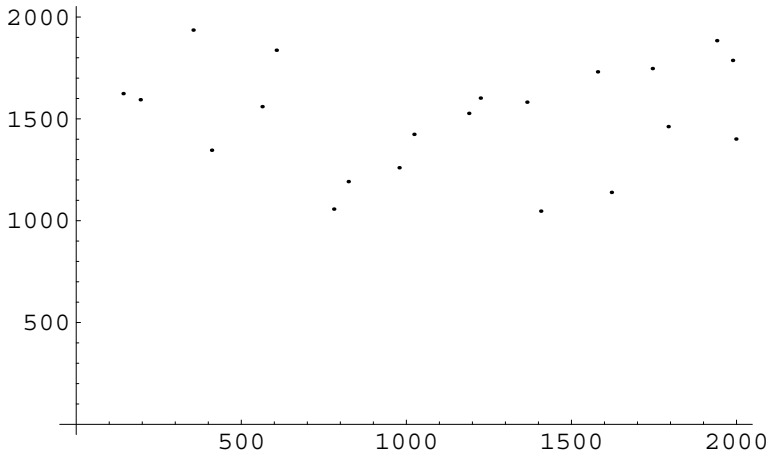
```
Length[LH]

2013
```

Como son demasiados puntos no mostraré la lista explícitamente. Por ejemplo, el primero de ellos es `First[LH] = {0,1}` y el último es `Last[LH] = {∞,∞}`. Lo que sí podemos hacer es representar la curva gráficamente, excluyendo el punto del infinito:

```
ListPlot[Drop[LH, -1], PlotStyle -> PointSize[0.005]];
```

El aspecto visual de la curva no es aquél al que estamos habituados (hemos dado un salto de lo continuo a lo discreto) pero no deja de ser sugestivo, sobre todo si se gira la imagen 90° en el sentido de las agujas del reloj. La gráfica anterior no tiene la misma significación que la de una curva sobre $\mathbb{R}$, por ejemplo, tres puntos de la curva pueden estar en una recta de $\mathbb{F}_{2003}$ sin que estén "alineados visualmente". Sin embargo, la figura conserva la simetría respecto a un eje horizontal (que en este caso no es la recta $y = 0$ debido a que estamos usando el "menor residuo no negativo" y no el "menor residuo absoluto" –cf. [23, p. 133] para la definición– para representar los elementos de $\mathbb{F}_{2003}$). También en este caso, cada recta vertical que corta a la curva en un punto distinto de $\mathcal{O}$ contiene, además de $\mathcal{O}$, dos puntos de la curva que son opuestos en la operación del grupo. Como el orden de la curva es impar, no hay puntos de orden 2.



Dado que $2013 = 3 \cdot 11 \cdot 61$ es libre de cuadrados, el grupo de la curva es cíclico, como se deduce del teorema de Cassels antes mencionado o, más generalmente, del teorema de estructura de los grupos abelianos finitos. Este teorema nos dice que el grupo es un producto de grupos cíclicos cuyos órdenes son potencias de primos, de modo que, en este caso, el grupo de la curva es isomorfo a $\mathbb{Z}_3 \times \mathbb{Z}_{11} \times \mathbb{Z}_{61}$ el cual, a su vez, es isomorfo a $\mathbb{Z}_{2013}$ por el teorema chino de los restos (cf. [11]) y, en consecuencia, cíclico. Para poder operar en dicho grupo se pueden programar en *Mathematica* las fórmulas para la suma de puntos que he dado anteriormente. Comenzaré definiendo la curva elíptica mediante los tres parámetros a, b, p, de su ecuación de Weierstrass:

```
CurvaEliptica[a_Integer, b_Integer, p_?PrimeQ] :=
   If[Mod[4 a^3 + 27 b^2, p] != 0, {a, b, p},
       Print["Los parametros dados no definen una curva eliptica"]] /; p > 3

H = CurvaEliptica[3, 1, 2003];
```

La curva definida por la ecuación H está ahora inicializada en la variable `H` de *Mathematica*. Para saber si un punto está en una curva, podemos usar:

```
PuntoQ[P_List, E_List] :=  SameQ[P, {Infinity, Infinity}] ||
   SameQ[Mod[P[[2]]^2 - P[[1]]^3 - E[[1]] P[[1]] - E[[2]], E[[3]]], 0]
```

Por ejemplo, `PuntoQ[#,H]&/@Take[LH,-4] = {True,True,True,True}`, nos dice que los cuatro últimos puntos de la lista `LH` están efectivamente en la curva. Para sumar puntos y para multiplicar un entero por un punto de la curva, traducimos a *Mathematica* las fórmulas antes dadas para la suma:

```
SumaEliptica[P_List, Q_List, E_List] :=
  Module[{x1 = P[[1]], y1 = P[[2]], x2 = Q[[1]], y2 = Q[[2]], lambda,
  a = E[[1]], b = E[[2]], p = E[[3]], oblicua = False},
```

```
   Which[SameQ[x1, Infinity], Return[Q], SameQ[x2, Infinity], Return[P],
    P == Q,
      If[Mod[y1, p] == 0, Return[{Infinity, Infinity}],
      lambda = Mod[Mod[3 Mod[x1 x1, p] + a, p]*PowerMod[2 y1, -1, p], p];
      oblicua = True],
    True,
      If[Mod[x1, p] == Mod[x2, p], Return[{Infinity, Infinity}],
      lambda = Mod[Mod[y2 - y1, p]*PowerMod[x2 - x1, -1, p], p];
oblicua = True]];
      If[oblicua == True, x3 = Mod[Mod[lambda lambda, p] - x1 - x2, p];
      y3 = Mod[Mod[-y1, p] + Mod[lambda (x1 - x3), p], p]; Return[{x3, y3}]]]

MultEliptica[n_Integer, P_List, E_List] :=
  Module[{bits=IntegerDigits[n, 2], punto=P, suma={Infinity, Infinity}, i},
     i = Length[bits];
       While[i >= 1, If[bits[[i]] == 1, suma = SumaEliptica[suma, punto, E]];
         punto = SumaEliptica[punto, punto, E]; i--];
            Return[suma]] /; n > 1

MultiplicacionEliptica[n_Integer, P_List, E_List] :=
   Which[n == 0, Return[{Infinity, Infinity}],
     n == 1, Return[P],
     n == -1, Return[{P[[1]],Mod[-P[[2]],E[[3]]]}],
     n > 1, Return[MultEliptica[n, P, E]],
     True, Return[MultEliptica[-n, {P[[1]],Mod[-P[[2]],E[[3]]]}, E]]]
```

Sabemos que el grupo de la curva $H(\mathbb{F}_{2003})$ es cíclico y, para poder trabajar en él, es necesario antes de nada determinar un generador. Dado que el grupo tiene orden 2013, el número de generadores es `EulerPhi[2013] = 1200` [11, p. 65]. Por tanto, no será difícil hallar un generador, es decir, un elemento de orden 2013, mediante fuerza bruta, calculando los órdenes de elementos de la curva –que, por el teorema de Lagrange, han de ser divisores del orden n de ésta– hasta encontrar uno adecuado (veremos un método mejor más adelante). Podemos usar la función:

```
OrdenEliptico[P_List, E_List, n_Integer] :=
   Module[{div = Divisors[n], i = 1, Q = P},
        While[Q != {Infinity, Infinity},
         Q = MultiplicacionEliptica[div[[i + 1]], P, E]; i++]; div[[i]]]
```

que, aplicada a los primeros 15 puntos de la lista `LH`, nos da:

```
OrdenEliptico[#, H, 2013] & /@ Take[LH, 15]

{671, 671, 671, 671, 61, 61, 671, 671, 671, 671, 671, 671, 2013, 2013, 671}
```

Esto muestra que el punto que ocupa la posición 13 en la lista es un generador. Este punto es `P = LH[[13]] = {14,28}` y se podría tomar como base para el logaritmo discreto en esta curva: cualquier otro punto tiene un logaritmos discreto en la base `P`. Aunque no aparecen entre los primeros puntos de la

lista sabemos que tienen que existir, por ejemplo, puntos de orden 3 y puntos de orden 11. Los puntos de orden 3 serán: `U = MultiplicacionEliptica[671, P, H] = {1387, 618}` (puesto que $2013/3 = 671$) y su simétrico `V = {1387, 1385}`. Estos son, junto con $\mathcal{O}$, los puntos de inflexión de esta curva. Podemos comprobar de una manera más explícita mediante la regla de la secante-tangente que, por ejemplo, `V`, es efectivamente un punto de inflexión. Para ello hacemos:

```
h[x_, y_] := y^2 - x^3 - 3x - 1;

t[x_, y_] := Derivative[1, 0][h][1387, 1385](x - 1387) +
              Derivative[0, 1][h][1387, 1385](y - 1385);

tmod[x_, y_] := PolynomialMod[t[x, y], 2003];
```

lo cual, dado que `tmod[x,y] = 1041 + 1336 x + 767`, nos dice que la recta tangente a la curva en el punto `V` es la de ecuación $1041 + 1336x + 767y \equiv 0 \pmod{2003}$. Si ahora calculamos la intersección de esta recta con la curva:

```
Solve[{tmod[x, y] == 0, h[x, y] == 0, Modulus == 2003}]

{{Modulus -> 2003, y -> 1385, x -> 1387}, {Modulus -> 2003, y -> 1385,
     x -> 1387}, {Modulus -> 2003, y -> 1385, x -> 1387}}
```

vemos que, como ya sabíamos por la teoría antes expuesta, la intersección tiene multiplicidad 3. Un punto de orden 11 de la curva es, por ejemplo, `MultiplicacionEliptica[183, P, H] = {517, 899}`. Menciono este hecho como curiosidad porque, por el teorema de Mazur antes mencionado, un punto de una curva elíptica sobre $\mathbb{Q}$ no puede tener nunca orden 11.

Si P es un punto de una curva elíptica, su orden es el logaritmo discreto de $\mathcal{O}$ en la base P. Podemos, pues, construir un programa para calcular logaritmos discretos por fuerza bruta, similar al usado para calcular el orden de un punto. Sin embargo, en este caso no se puede aplicar el teorema de Lagrange y hay que ir recorriendo todos los posibles valores hasta encontrar el logaritmo discreto:

```
LogaritmoDiscretoEliptico[Q_List, P_List, E_List] :=
   Module[{R = P, i = 1}, While[R != Q, R = SumaEliptica[P, R, E]; i++]; i]
```

Podemos hacer ahora una comparación experimental del tiempo requerido por la función `MultiplicacionEliptica[#,P,H]&` con el requerido por su inversa `LogaritmoDiscretoEliptico[#,P,H]`, para la curva y el generador anteriores. Calculamos, sucesivamente:

```
MultiplicacionEliptica[#,P,H]&/@ Range[1000,2000,100]//Timing

{0.02 Second,{{698,1805},{1713,423},{1919,44},{337,21},{1222,1439},{122,
       595},{1666,160},{1084,1680},{716,1569},{482,928},{208,1041}}}

LogaritmoDiscretoEliptico[#,P,H]&/@ %[[2]]//Timing

{2.554 Second,{1000,1100,1200,1300,1400,1500,1600,1700,1800,1900,2000}}
```

Vemos que la disparidad entre los tiempos es notable, como consecuencia de que, mientras que para la multiplicación elíptica hemos usado un algoritmo de tiempo polinómico (el análogo aditivo de la exponenciación binaria, en el cual se procede mediante "duplicaciones y sumas" para reducir el número de operaciones a una función polinómica del tamaño del entero por el que multiplicamos), para el logaritmo discreto hemos usado un algoritmo de tiempo exponencial (el más ingenuo de todos, consistente en calcular todos los múltiplos de P por enteros hasta alcanzar el punto cuyo logaritmo estamos buscando). Obsérvese que aquí no es posible "atajar" como se hace cuando lo único que se busca es nP para un valor dado de n. Ahora no sabemos cual puede ser el logaritmo discreto y hay que ir avanzando de 1 en 1, sumando P cada vez, pues cualquier valor intermedio (entre 1 y el orden de la curva) podría ser el logaritmo buscado. A pesar de que cada suma elíptica se ejecuta en tiempo polinómico, se requieren $O(n)$ de estas sumas, siendo n el orden del grupo. Existen algoritmos más eficientes para calcular logaritmos discretos, pero todos los conocidos siguen siendo algoritmos exponenciales.

Si ahora aplicamos los procedimientos anteriores a la curva elíptica sobre $\mathbb{F}_{2003}$ definida por la ecuación de Weierstrass $K : y^2 = x^3 + x + 1$ vemos que el orden de esta curva es 2008 y, al ser par, ahora sí existen puntos de orden 2. La razón de fondo por la que esto sucede es que el polinomio cúbico x^3+x+1 tiene tres ceros en $\mathbb{F}_{2003}$, los cuales van a dar lugar a los tres puntos de orden 2 que tiene la curva. En cambio, el polinomio cúbico $x^3 + 3x + 1$ que definía la curva $H(\mathbb{F}_{2003})$ anterior no tiene ceros en $\mathbb{F}_{2003}$ (es irreducible sobre este cuerpo) y por eso el orden $\#H(\mathbb{F}_{2003})$ era impar. Comprobamos esto con *Mathematica*:

```
Factor[x^3 + 3 x + 1, Modulus->2003]

(1 + 3 x + x^3)

Factor[x^3 + x + 1, Modulus->2003]

(275 + x) (564 + x) (1164 + x)
```

Esto ya nos dice que los puntos de orden 2 de la curva $K(\mathbb{F}_{2003})$ son precisamente los de coordenadas (839,0), (1439,0) y (1728,0). Además, vemos que como hay 3 puntos de orden 2 el grupo no puede ser cíclico. De hecho, es fácil comprobar que el subgrupo de $K(\mathbb{F}_{2003})[4]$ formado por los puntos de 4-torsión (aquellos cuyo orden es divisor de 4) es isomorfo a $\mathbb{Z}_2 \times \mathbb{Z}_4$. Para verlo bastaría comprobar que existe en el grupo un elemento de orden 4 (teniendo, además en cuenta que el orden es $2008 = 2^3 \cdot 251$). Más explícitamente, si se realizan los cálculos similares a los realizados antes con la curva H, para obtener la lista `LK` de los puntos de la curva K, podemos entonces hacer:

```
LK[[Flatten[
     Position[MultiplicacionEliptica[4, #, K] & /@ LK, {Infinity,Infinity}]]]]

{{182, 849}, {182, 1154}, {839, 0}, {1271, 668}, {1271, 1335}, {1439, 0},
{1728, 0}, {Infinity, Infinity}}
```

Esta es la lista de los puntos de $K(\mathbb{F}_{2003})[4]$, cuyos órdenes son:

```
OrdenEliptico[#,K,2008]&/@%

{4,4,2,4,4,2,2,1}
```

En consecuencia, el grupo $K(\mathbb{F}_{2003})$ es isomorfo a $\mathbb{Z}_2 \times \mathbb{Z}_4 \times \mathbb{Z}_{251}$ y a $\mathbb{Z}_2 \times \mathbb{Z}_{1004}$, lo que también se deduce del teorema de Cassels una vez que se sabe que hay tres elementos de orden 2.

Antes de pasar a discutir con más detalle el uso de las curvas elípticas en criptografía, mencionaré que las curvas elípticas sobre cuerpos finitos también tienen aplicación a otras cuestiones relacionadas como son, el reconocimiento de primos (a través del algoritmo ECPP, "Elliptic Curve Primality Proving") y la factorización de enteros (ECM, "Elliptic Curve Method", de H.W. Lenstra). Dos buenas referencias para estos aspectos son [7, 9]. Para curvas elípticas en general, la referencia estándar es el libro de Silverman [42], junto con [45, 8]. El uso criptográfico de las curvas elípticas está analizado con detalle en [29] y [4] y también en textos más generales como [9, 23, 24, 47, 48, 49]. También existen dos excelentes referencias en castellano: [33, 36], aunque la primera está más orientada a la teoría de códigos. Por otra parte, existe una gran cantidad de software adecuado para trabajar con curvas elípticas, desde el de los paquetes Pari-GP [35], Magma [27], MIRACL [32], APECS (excelente paquete para *Maple* de Ian Connell [2]), hasta alguno más especializado, como el programa `mwrank` de J. Cremona para determinar el rango de las curvas elípticas sobre $\mathbb{Q}$ [10].

Criptografía basada en curvas elípticas

El gran interés de las curvas elípticas para la criptografía radica en dos circunstancias básicas. La primera es que las curvas elípticas sobre cuerpos finitos proporcionan una enorme cantidad de grupos, lo cual permite una gran flexibilidad al elegirlos. La segunda, y todavía más importante, es que, por el momento, no se conoce ningún algoritmo subexponencial para resolver el PLD sobre estos grupos. De hecho, Miller argumentó en [31] su conclusión de que *"es extremadamente improbable que un ataque tipo 'cálculo del índice' en curvas elípticas llegue a funcionar alguna vez"*. La razón más importante es que, para curvas elípticas, no existe un buen análogo del concepto de "primo pequeño" para poder formar una "base de factores" (o su análogo aditivo). Una posibilidad sería usar puntos con "coordenadas pequeñas", pero el problema es que no hay suficientes y, además, cuando un punto de una curva elíptica se descompone en suma de otros, estos pueden tener coordenadas mucho más grandes (a diferencia de lo que ocurre cuando factorizamos un entero, en cuyo caso los factores siempre son más pequeños que el número). Otros autores también han apoyado las tesis de Miller y, en particular, Silverman y Suzuki [44] han hecho un análisis muy preciso del problema llegando a las mismas

conclusiones. Por eso, se considera que las versiones de Diffie-Hellman, El Gamal y DSA para curvas elípticas, son seguras con un grupo de orden mucho menor que el que hay que usar cuando se toma $\mathbb{Z}_p^*$. Para poder implementar uno de estos criptosistemas sobre una curva elíptica es necesario, en primer lugar, poder determinar el orden del grupo, que debe ser divisible por un primo lo bastante grande como para que el PLD no se pueda resolver mediante algoritmos genéricos.

Hemos visto en el ejemplo pequeño de la sección precedente que el orden de la curva –que era, en este caso, 2013– estaba bastante próximo al primo p (2003). Hay un argumento heurístico sencillo, pero muy sugestivo, que indica que esto es lo que cabe esperar. En efecto, dada la ecuación de Weierstrass $y^2 = f(x)$ (con $f(x) = x^3+ax+b$), se puede hacer una lista de los puntos de la curva siguiendo el procedimiento que hemos empleado en el ejemplo citado: se calculan los valores de $f(x) \pmod p$ cuando x recorre todos los posibles valores $0, 1, \ldots p-1$ y se determina cuales de ellos son residuos cuadráticos módulo p. Como la mitad de los elementos no nulos de $\mathbb{F}_p$ son residuos cuadráticos, cabe esperar que, aproximadamente, la mitad de los valores no nulos de $f(x)$ sean cuadrados en $\mathbb{F}_p$. Cada uno de ellos tiene dos raíces cuadradas, de modo que la mitad de las veces el valor de x nos proporciona dos puntos de la curva y la otra mitad no nos proporciona ninguno. Así, el número de puntos obtenido de esta forma es, aproximadamente, p, y si se añade el punto del infinito, el número de puntos esperado es $p+1$. Este argumento no es una demostración, pero la estimación que proporciona fue establecida rigurosamente por Hasse en los años 30. El *teorema de Hasse* afirma que $\#E(\mathbb{F}_p) = p + 1 - t$, donde $|t| \leq 2\sqrt{p}$ (este teorema fue generalizado por Weil al demostrar la llamada *Hipótesis de Riemann para curvas sobre cuerpos finitos*), cf. [42]. Para todos estos posibles valores existen curvas sobre $\mathbb{F}_p$ con ese orden y, además, por un resultado de Lenstra, los valores que toma el orden $\#E(\mathbb{F}_p)$ tienen, aproximadamente, distribución uniforme [29]. El parámetro t recibe el nombre de *traza* (o traza de Frobenius) de la curva y ésta se dice *supersingular* cuando $t = 0$, es decir, cuando $\#E(\mathbb{F}_p) = p+1$ (otra paradoja terminológica: ¡las curvas supersingulares son no singulares!). Teniendo en cuenta lo ya dicho anteriormente el *problema del logaritmo discreto elíptico* (PLDE) es el siguiente: Dada una curva elíptica $E(\mathbb{F}_p)$, un punto $P \in E(\mathbb{F}_p)$ de orden n, y un punto $Q \in E(\mathbb{F}_p)$, determinar el entero l, $0 \leq l \leq n-1$, si existe, tal que $Q = lP$.

Para que el PLDE sea difícil es conveniente elegir una curva $E(\mathbb{F}_p)$ cuyo orden $\#E(\mathbb{F}_p)$ tenga un factor primo grande. Veamos, antes de nada, lo imprescindible que es este requisito. Consideremos el primo $p = 56789012345678677$ y la curva elíptica definida sobre $\mathbb{F}_p$ por la ecuación de Weierstrass $J : y^2 = x^3 + 5x + 3$. El orden de la curva se puede calcular fácilmente –como indicaré más adelante– y, en este caso, resulta ser $n = 56789012247238106 = 2 \cdot 11 \cdot 43 \cdot 953 \cdot 1237 \cdot 3881 \cdot 13121$. El primo p no es lo suficientemente grande para que la curva pueda ser usada criptográficamente, puesto que el PLDE sobre J se podría resolver fácilmente –aunque n no fuese producto de primos pequeños– por los métodos basados en la "paradoja del cumpleaños" [4, 9]

como, por ejemplo, el método rho de Pollard, aun siendo éste un algoritmo de tiempo exponencial $O(\sqrt{p})$. Sin embargo, resulta obvio que el PLDE en esta curva sería muy difícil de resolver por fuerza bruta como habíamos hecho en el ejemplo precedente. Lo que voy a mostrar ahora es como se puede usar la reducción de Pohlig-Hellman para después resolver el problema fácilmente mediante fuerza bruta. Comenzamos inicializando la curva y su orden:

```
J = CurvaEliptica[5, 3, 56789012345678677];
n = 56789012247238106;
```

A continuación comprobamos que el orden es, efectivamente, un producto de primos pequeños:

```
FactorInteger[n]
```

```
{{2,1},{11,1},{43,1},{953,1},{1237,1},{3881,1},{13121,1}}
```

Como el orden n es libre de cuadrados, sabemos ya que el grupo de la curva es cíclico. Podemos encontrar fácilmente un generador del mismo mediante el siguiente programa, que es una modificación del antes usado para encontrar un generador del grupo multiplicativo de un cuerpo finito. La entrada del programa es la curva (que debe ser un grupo cíclico) y su orden.

```
GeneradorEliptico[E_List, n_Integer] :=
  Module[{x = 1, y = 1, j = 1, R = {Infinity, Infinity},
      listadiv = n/(First /@ FactorInteger[n])},
    While[R == {Infinity, Infinity}, x = Random[Integer, {1, E[[3]] - 1}];
      If[JacobiSymbol[x^3 + E[[1]] x + E[[2]], E[[3]]] == 1,
        y = SqrtMod[x^3 + E[[1]] x + E[[2]], E[[3]]]; j = 1; R = {1, 1};
        While[j <= Length[listadiv] && R != {Infinity, Infinity},
          R = MultiplicacionEliptica[listadiv[[j]], {x, y}, E]; j++]]]; {x, y}]
```

Apliquemos esto a nuestra curva J de orden n para obtener un generador, que llamamos P:

```
P = GeneradorEliptico[J,n]
```

```
{387745622405146, 21122494081854546}
```

El generador P será la base del logaritmo discreto. Supongamos ahora que tomamos otro punto Q en la curva:

```
Q = {12801635973418499, 38065494702169500};
PuntoQ[Q,J]
```

```
True
```

El problema es buscar el logaritmo discreto de Q en la base P. Para ello, aplicaremos el algoritmo de Pohlig-Hellman y la función anteriormente usada `LogaritmoDiscretoEliptico[]`. En la función siguiente, Q es el punto cuyo logaritmo queremos hallar, P es la base, E es la curva y n el orden de ésta, que suponemos libre de cuadrados; de no ser así el algoritmo también puede ser aplicado pero habría que hacer algunas modificaciones en el programa.

```
PohligHellmanEliptico[Q_List, P_List, E_List, n_Integer] :=
   Module[{f = FactorInteger[n], listadiv = {}, primos = {}},
     listadiv = n/(First /@ f); primos = First /@ f;
     ChineseRemainder[MapThread[LogaritmoDiscretoEliptico[#1, #2, E] &,
         {MultiplicacionEliptica[#, Q, E] & /@ listadiv,
          MultiplicacionEliptica[#, P, E] & /@ listadiv}], primos]]
```

Estamos ya en condiciones de calcular el logaritmo buscado, mediante:

```
PohligHellmanEliptico[Q,P,J,n]
```

```
23456789012345679
```

Este resultado se obtiene en muy pocos segundos porque lo más costoso del proceso es calcular un logaritmo discreto módulo 13121 que se resuelve fácilmente por fuerza bruta. Podemos comprobar que el valor obtenido es, efectivamente, el logaritmo buscado, pues:

```
MultiplicacionEliptica[23456789012345679,P,J]
```

```
{12801635973418499, 38065494702169500}
```

Para comparar, el lector puede estimar el tiempo que requiriría el cálculo de este logaritmo en su máquina usando sólo `LogaritmoDiscretoEliptico[]`, midiendo el tiempo requerido por el cálculo de logaritmos "pequeños" y teniendo en cuenta que el tiempo para calcular un l tal que $lP = R$ por fuerza bruta (dados P y R) es esencialmente proporcional a l (¡no al tamaño de l ni siquiera a $\sqrt{l}$!).

Para elegir una curva para uso criptográfico hay que elegir antes el primo p (es decir, el cuerpo $\mathbb{F}_p$) y, por el teorema de Hasse, el tamaño de p deberá ser aproximadamente el mismo que el buscado para $\#E(\mathbb{F}_p)$; lo más usual es elegir p aleatoriamente. Por lo que acabamos de ver, es deseable que el orden de la curva sea primo o un producto de un primo por un entero pequeño. Si es primo, se estima que bastará que tenga unos 160 bits siempre y cuando sólo se puedan aplicar los algoritmos genéricos, que son exponenciales. Para asegurarse de que esto es así, el orden tiene que cumplir una serie de condiciones adicionales. En primer lugar, hay que evitar las llamadas *curvas anómalas*, que son aquellas para las que $\#E(\mathbb{F}_p) = p$. La razón es que, en este caso, existe un algoritmo de tiempo polinómico debido a Semaev-Smart-Satoh-Araki [25] que computa un isomorfismo entre $E(\mathbb{F}_p)$ y el grupo aditivo de $\mathbb{F}_p$, es decir, $\mathbb{Z}_p$. Esto a su vez proporciona un algoritmo de tiempo polinómico para el PLDE en este caso, pues lo reduce al PLD sobre $\mathbb{Z}_p$. Otro ataque que permite resolver eficientemente el PLDE en algunos casos es el llamado "ataque MOV", debido a Menezes, Okamoto y Vanstone [4, 29, 30]. Se basa en la construcción de una inmersión de $E(\mathbb{F}_p)$ en el grupo multiplicativo $\mathbb{F}^*_{p^k}$ del cuerpo $\mathbb{F}_{p^k}$, para algún entero k, lo cual reduce el PLDE al PLD sobre $\mathbb{F}^*_{p^k}$. Sin embargo, este ataque sólo tiene interés cuando k es pequeño, algo que ocurre para curvas

supersingulares pues entonces se sabe que $k \leq 6$, lo que proporciona un algoritmo subexponencial para el PLDE en este caso. Una curva elíptica generada aleatoriamente es extremadamente improbable que sea supersingular y, por el contrario, es muy probable que para ella $k > \log^2 p$, en cuyo caso sólo se conocen algoritmos exponenciales para el PLDE.

Existen muchos métodos para generar curvas elípticas $E(\mathbb{F}_p)$ para uso criptográfico [25] y uno de los más utilizados es el de elegir sus coeficientes aleatoriamente (en ocasiones este proceso se combina con la elección simultánea de un punto en la curva). A continuación habrá que calcular el orden de la curva para asegurarse de que cumple las condiciones anteriores. Todos los métodos eficientes para calcularlo se basan en un algoritmo de tiempo polinómico descubierto por René Schoof en 1985 [40], cuya idea básica consiste en determinar la traza t de la curva calculando para ello t módulo primos pequeños (convenientemente elegidos teniendo en cuenta el teorema de Hasse, cf. [4] para los detalles) y, finalmente, obteniendo t mediante el teorema chino de los restos (este es el algoritmo que he usado para calcular el orden de la curva del ejemplo precedente; más adelante daré otro ejemplo y una referencia concreta al programa utilizado). El algoritmo de Schoof tiene complejidad $O(\log^8 p)$ –se puede mejorar usando algoritmos asintóticamente rápidos para la multiplicación– y ya es suficiente para calcular $\#E(\mathbb{F}_p)$ cuando p es un primo de, por ejemplo, 160 bits. Para primos apreciablemente más grandes, se puede usar una mejora debida a Elkies y Atkin que da lugar al llamado *algoritmo SEA* (de Schoof-Elkies-Atkin) el cual, bajo ciertas hipótesis razonables, tiene complejidad $O(\log^6 p)$ [4, 10] (aquí, como en gran parte de la discusión anterior, el primo p se puede sustituir por una potencia $q = p^k$, y el cuerpo $\mathbb{F}_p$ por $\mathbb{F}_q$, pero me estoy limitando a considerar curvas sobre $\mathbb{F}_p$, por simplicidad y por su gran interés criptográfico).

Una vez determinado el orden n de la curva elegida, ésta se descarta si n no cumple las condiciones antes mencionadas. Si n no es primo o bien el producto de un primo por un entero pequeño, hay que elegir otra curva y volver a probar, hasta encontrar una con esta propiedad. Esto no es difícil en virtud del resultado de Lenstra según el cual los órdenes de las curvas $E(\mathbb{F}_p)$ están uniformemente distribuidos en el intervalo $[p+1-\sqrt{p}, p+1+\sqrt{p}]$, lo cual nos dice que la probabilidad de que el orden de la curva sea un producto de un primo por un entero pequeño es aproximadamente la misma que la de que un entero aleatorio del mismo tamaño cumpla esa condición. El siguiente paso es comprobar si la curva es anómala o supersingular (lo cual no tiene ninguna dificultad) y, si es así, descartarla (para curvas elegidas al azar, esto es sumamente improbable). Además de todo ésto, se puede tomar una precaución adicional para tener la seguridad de que la curva no será vulnerable al ataque MOV antes mencionado. Teniendo en cuenta que una condición necesaria para la existencia de una inmersión del grupo $E(\mathbb{F}_p)$ en $\mathbb{F}^*_{p^k}$ es que n divida a $p^k - 1$, se puede comprobar que esto no se verifica para valores pequeños de k, para los cuales el PLD en $\mathbb{F}^*_{p^k}$ pudiera ser factible. En la práctica, bastaría comprobar

que n no divide a $p^k - 1$ para $1 \leq k \leq 20$ [25], lo que ya excluye que la curva sea supersingular.

El último paso para la implementación del intercambio de Diffie-Hellman o del criptosistema de El Gamal sobre la curva elegida $E(\mathbb{F}_p)$, con ecuación $y^2 = x^3 + ax + b$, es seleccionar un punto $P \in E(\mathbb{F}_p)$ que sirva de base para el logaritmo discreto y realizar los cálculos adicionales correspondientes, que sólo utilizan la operación de suma de puntos en la curva. La elección de un punto aleatorio en la curva (que sea un generador si el grupo es cíclico o, en todo caso, un punto de orden grande) no ofrece dificultad alguna con un método como el usado en `GeneradorEliptico[]`: se toma al azar un elemento $x_0 \in \mathbb{F}_p$, se calcula $x_0^3 + ax_0 + b$, se determina mediante su símbolo de Legendre si este valor es un residuo cuadrático módulo p y, si es así, se halla una raíz cuadrada del mismo módulo p mediante el algoritmo descrito en [24, p. 47] o [9, Algorithm 2.3.8]. Como la mitad de los elementos de $\mathbb{F}_p^*$ son residuos cuadráticos, esto permite encontrar rápidamente un punto de la curva. Después sólo queda determinar si tiene el orden adecuado y, si no es así, repetir el proceso.

Para ilustrar las ideas anteriores sobre el uso de curvas elípticas en criptografía, voy a mencionar un ejemplo concreto de una curva que ha sido usada por Microsoft en su software, concretamente, en la versión 2 del llamado "Microsoft Digital Rights Management" (MS-DRM), que se aplica a los derechos de reproducción de ficheros de audio en formato .wma (en concreto, a la versión 7 de "Windows Media Audio") y que ha sido descrito por "Beale Screamer" (en adelante, BS), un criptólogo anónimo que, en octubre de 2001 envió al grupo de noticias `sci.crypt` los detalles, junto con una forma de romper el sistema [3]. Esta curva, que denotaré $M(\mathbb{F}_p)$, está definida sobre $\mathbb{F}_p$, donde

```
p = 7859631023794288223766947894468973962074985689 51;
```

y la ecuación M está dada más abajo. Si expresamos p en hexadecimal, como hace BS en sus notas, vemos que:

```
BaseForm[p, 16]

89abcdef012345672718281831415926141424f7
```

lo que quizá muestra la razón por la que fue elegido: si se excluyen los dígitos "4f7" del final, los restantes son: los 16 dígitos del sistema hexadecimal, en orden creciente cíclico empezando por el 8, los 8 primeros dígitos de e, los 8 primeros dígitos de π, y los cinco primeros dígitos de $\sqrt{2}$, lo que proporciona una regla mnemotécnica fácil (y un alto *"nerd appeal"*, en palabras de BS).

En cuanto a la ecuación de la curva, es $M : y^2 = x^3 + ax + b$, donde a y b son, dados también en hexadecimal:

```
a = 37a5abccd277bce87632ff3d4780c009ebe41497;
b = 0dd8dabf725e2f3228e85f1ad78fdedf9328239e;
```

Ahora, inicializamos la curva en *Mathematica*:

```
M = CurvaEliptica[a,b,p];
```

Podemos comprobar si la curva M cumple los requisitos anteriormente expuestos para que el correspondiente PLDE sea "difícil". Para calcular el orden de la curva recurrimos al algoritmo de Schoof. Dado que no está implementado en *Mathematica*, usaremos la versión que viene programada (por M. Scott) en la biblioteca *MIRACL* (Multiprecision Integer and Rational Arithmetic C/C++ Library) de rutinas en C dedicadas a la aritmética de precisión múltiple, que permite trabajar con enteros (y racionales) muy grandes, disponible en [32]. El resultado de ejecutar dicho algoritmo sobre la curva elíptica de Microsoft es el siguiente (el parámetro -h es para indicarle al programa que los datos están en hexadecimal; he suprimido parte de la salida del programa para ahorrar espacio):

```
>Schoof -h 89ABCDEF012345672718281831415926141424F7
37A5ABCCD277BCE87632FF3D4780C009EBE41497
0DD8DABF725E2F3228E85F1AD78FDEDF9328239E

P mod 8 = 7
P is 160 bits long
Counting the number of points (NP) on the curve
y^2= x^3 + 3176890812513255034763174764138276932727469559 27*x +
790528966078787587181205720257185354321006519 34
mod 7859631023794288223766947894468973962074985689 51
11 primes used (plus largest prime powers), largest is 31
NP mod 2 =   1
NP mod 3 =   2
.....................................................
NP mod 29 =  4
NP mod 31 = 23
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................
NP= 7859631023794288223766930248817149576126861574 29
NP is Prime!
```

Vemos que casi la mitad de los dígitos –empezando por el más significativo– de p y del orden n (llamado NP por el programa anterior) coinciden, como estaba predicho por el teorema de Hasse. Además, el orden de la curva es primo, lo cual no es una casualidad sino que, con toda seguridad, fue buscado por Microsoft para hacer el PLDE de la curva lo más difícil posible.

Por otra parte, vemos que la curva $M(\mathbb{F}_p)$ no es anómala, pues $n \neq p$. Además, la curva es resistente al ataque MOV, puesto que:

```
n = 7859631023794288223766930248817149576126861574 29;
MemberQ[PowerMod[p, Range[20], n] -1, 0]

False
```

En consecuencia, $M(\mathbb{F}_p)$ es una curva perfectamente adecuada para la implementación de criptosistemas basados en la dificultad del PLDE. Dado que el orden de la curva es primo, cualquier punto distinto del punto del

infinito es un generador. El utilizado por el software de Microsoft es, según BS, el punto P cuyas coordenadas hexadecimales son las siguientes (con la comprobación de que, efectivamente, es un punto de $M(\mathbb{F}_p)$):

```
P = {16^^8723947fd6a3a1e53510c07dba38daf0109fa120,
      16^^445744911075522d8c3c5856d4ed7acda379936f};
PuntoQ[P,M]

True
```

Veamos ahora como se usa el criptosistema de El Gamal sobre $M(\mathbb{F}_p)$ en el esquema MS-DRM. Ya está fijado el grupo G a usar que será precisamente $M(\mathbb{F}_p)$ y también el punto $P \in M(\mathbb{F}_p)$ que será la base para los logaritmos discretos. Cada usuario B tiene que elegir una clave privada, que es un entero b tal que $1 < b < n-1$, pero en este caso, la clave privada de B la ha generado el software de Microsoft, que la ha ocultado en distintos ficheros del ordenador de B. Supongamos que esta clave privada es, como en el ejemplo dado por BS,

```
b = 16^^757ff01b853496452eea0b0646c3a357a6f33509

670805031139910513517527207693060456300217054473
```

La clave pública de B es entonces el elemento $bP \in M(\mathbb{F}_p)$. La calculamos a continuación:

```
bP = MultiplicacionEliptica[b,P,M]

{1446866629014043092250222559914573373595793129 71,
7834225388893147423200286615155700747880565562 63}
```

Cuando B compra música a través de Internet, su ordenador se pone en contacto con el servidor de licencias del vendedor y le envía su clave pública bP. Entonces, el sistema del vendedor le envía a B un mensaje en la forma que he indicado antes al describir el criptosistema de El Gamal; este mensaje es la licencia que permitirá a B reproducir en su ordenador la música que ha adquirido, pero para que B pueda hacerlo deberá desencriptar el mensaje y recuperar el texto claro correspondiente, que es la clave para poder reproducir el fichero de audio (el cual ha sido encriptado a su vez mediante criptosistemas simétricos). La licencia es un par de puntos de la curva, de la forma: $(kP, Q + k(bP))$, donde Q es el "texto claro" que permitirá reproducir la música. Cuando el ordenador de B recibe esta licencia, usa la clave privada b para calcular $b(kP) = k(bP)$. A continuación, toma el segundo punto de la licencia y le resta este punto, es decir, calcula $Q = Q + k(bP) - b(kP)$, con lo cual B ya puede escuchar la música. Continuando con el ejemplo dado por BS, supongamos que B recibe la licencia formada por los puntos kP y $Q + kbP$, siguientes:

```
kP = {16^^1f78b9f73968f8d12099ae9bbcc25fe73d1b4b54,
      16^^7a3bf9e07fe82b05c2b45bb35f603c417b447f5e};
```

```
QkbP = {16^^18257450abd22b4d1802484230a646c850aad443,
       16^^136a9f66165acb8e500ab0292274deb56ffe34b3};
```

Ambos puntos están en la curva M, pues se tiene que `PuntoQ[kP,M] = PuntoQ[QkbP,M] = True`. A continuación, el software de B multiplica el primero de estos puntos por su clave privada b:

```
bkP = MultiplicacionEliptica[b,kP,M]

{328901393518732637577115650601768681044040715701,
5869478380878159936013505654887888462038879881 62}
```

Ya está completada la primera parte del proceso. Ahora, el software del ordenador de B resta al segundo punto recibido el que acaba de calcular:

```
Q = SumaEliptica[QkbP,Simetrico[bkP,M[[3]]],M]

{14489646124220757767, 669337780373284096274895136618194604469696830074}
```

Finalmente B (o, más bien, su ordenador) ha obtenido las coordenadas del punto Q que le permiten reproducir la música. Los ficheros de audio están encriptados usando criptosistemas simétricos como DES, RC4 y otros, en una forma bastante complicada que no voy a describir y, como indica BS, la coordenada x de Q es la clave de contenido que permite al Windows Media Player desencriptarlos y reproducirlos.

Pero supongamos que B quiere compartir esta música con su amiga C. Para ello le pasa el fichero .wma correspondiente y también el fichero que contiene la licencia $(kP, Q + k(bP))$. Sin embargo, cuando C quiere reproducir la música, no puede hacerlo, porque su software trata de recuperar Q haciendo el cálculo $Q + k(bP) - c(kP)$, donde c es la clave privada de C (claro está que el ordenador de C no conoce la clave privada b de B). Como $kbP \neq kcP$, el punto resultante no es Q y de ahí que C no pueda reproducir la música.

En [3] se muestra la forma de romper el sistema de protección de los ficheros de audio .wma protegidos por el sistema MS-DRM mediante la recuperación de la clave privada del usuario correspondiente al criptosistema de El Gamal elíptico, lo que permite a su vez recuperar, a partir de una licencia válida, la clave de contenido que permitirá reproducir éste en cualquier sistema. Sin embargo, esto no significa, en modo alguno, que el criptosistema basado en curvas elípticas haya sido roto, pues no se conoce la forma de recuperar la clave secreta a partir de la clave pública de B y un usuario C que intercepte la licencia enviada a B no puede usarla para reproducir el contenido. Lo que ocurre en este caso es que la clave privada de B está oculta en su ordenador y BS ha encontrado la forma de que B pueda recuperarla. El problema que tienen los vendedores de contenido es cómo proporcionar a cada usuario U una clave privada que le permita reproducir la música que ha adquirido en su ordenador, sin permitirle a U que pueda transferir esa clave privada a otros ordenadores u otras personas. Es, por decirlo así, un problema

de "criptografía dentro de la criptografía" y la debilidad de la versión 2 del MS-DRM que permitía recuperar la clave privada ya ha sido corregida.

Como conclusión, indicaré que el PLDE es, por lo que sabemos, sustancialmente más difícil que el PLD sobre $\mathbb{Z}_p^*$. En la actualidad se acepta que un nivel de seguridad criptográfica suficiente es el proporcionado por un criptosistema cuya rotura requiera 10^{12} años MIPS, donde 1 año MIPS es la computación que puede realizar una máquina que procese un millón de instrucciones por segundo (1 MIPS), durante un año. Se estima que este nivel de seguridad se alcanza, usando el PLD sobre $\mathbb{Z}_p^*$ como en el DSA (o bien, usando RSA), con una clave (módulo) de 1024 bits. Por el contrario, los criptosistemas basados en el PLDE como El Gamal elíptico (o la versión elíptica del DSA, ECDSA, que es un estándar ANSI desde 1999 [19]), alcanzan este nivel de seguridad –e incluso superior– con módulos de 160 bits (por el momento, el módulo más grande para el que se ha conseguido resolver el PLDE, en el contexto de la Certicom Elliptic Curve Challenge [5], que ofrece premios en metálico, es un módulo de 108 bits [15]) [26, 34]. Esto hace que la criptografía basada en curvas elípticas proporcione un ahorro de ancho de banda e implementaciones más eficientes, y estas son las razones por las que este tipo de criptografía se está usando intensamente en aplicaciones en las que la potencia computacional o el espacio de almacenamiento son limitados, como ocurre, por ejemplo, en las tarjetas inteligentes y en la telefonía móvil.

Bibliografía

[1] M. Agrawal, N. Kayal, N. Saxena, PRIMES is in P, preprint, en `http://www.cse.iitk.ac.in/news/primality.html`.

[2] Apecs, disponible en `ftp://ftp.math.mcgill.ca/pub/Apecs`.

[3] "Beale Screamer", Microsoft's Digital Rights Management Scheme - Technical Details, en: `http://cryptome.org/beale-sci-crypt.htm`.

[4] I. Blake, G. Seroussi, N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press (1999).

[5] Certicom Elliptic Curve Challenge, en: `http://www.certicom.ca/resources/ecc_chall/challenge.html`.

[6] L.S. Charlap, D.P. Robbins, An Elementary Introduction to Elliptic Curves, CRD Expository Report No. 31, Institute for Defense Analysis, Princeton (1988).

[7] H. Cohen, *A Course in Computational Number Theory*, Springer-Verlag (1993).

[8] I. Connell, *Elliptic Curve Handbook*, en `ftp://ftp.math.mcgill.ca/pub/ECH1`.

[9] R. Crandall, C. Pomerance, *Prime numbers, A computational perspective*, Springer-Verlag (2001).

[10] J. Cremona, `http://www.maths.nott.ac.uk/personal/jec/`.

[11] M. Demazure, *Cours d'algèbre*, Cassini (1997).

[12] L.E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea (1952).

[13] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory, 22 (1976), 644-654.

[14] H.M. Edwards, *Fermat's last theorem*, Springer Verlag (2000).

[15] Elliptic Curve Discrete Logarithms Project, en: `http://cristal.inria.fr/~harley/ecdl_top/`.

[16] J.L. Gómez Pardo, Aspectos computacionales de los números primos (II), La Gaceta de la RSME 5, no. 1 (2002), 197-227.

[17] G.H. Hardy, *Apología de un matemático*, Ariel (1981).

[18] M.J. Jacobson, N. Koblitz, J.H. Silverman, A. Stein, E. Teske, Analysis of the xedni calculus attack, Designs, Codes and Cryptography 20 (2000), 41-64.

[19] D. Johnson, A. Menezes, The Elliptic Curve Digital Signature Algorithm (ECDSA), Tech. Report CORR 99-34, Dept. of C&O, University of Waterloo (1999).

[20] M. Joye, Introduction élémentaire à la théorie des courbes elliptiques, Technical Report CG-1995/1, Univ. Catholique de Louvain (1995).

[21] D. Kahn, *The Codebreakers*, 2nd. Ed., Scribner (1996).

[22] N. Koblitz, Elliptic curve cryptosystems, Math. Comp. 48 (1987), 203-209.

[23] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer (1994).

[24] N. Koblitz, *Algebraic Aspects of Cryptography*, Springer (1998).

[25] N. Koblitz, A. Menezes, S. Vanstone, The State of Elliptic Curve Cryptography, Designs, Codes and Cryptography, 19, 173-193, (2000).

[26] A.K. Lenstra, E.R. Verheul, Selecting cryptographic key sizes, J. Cryptology 14 (2001), 255-293.

[27] Magma, `http://magma.maths.usyd.edu.au/magma/`.

[28] K.S. McCurley, The Discrete Logarithm Problem, en Proc. Symp. Appl. Math. (Carl Pomerance, Editor), Amer. Math. Soc. (1990).

[29] A.J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer (1993).

[30] A.J. Menezes, T. Okamoto, S.A. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Trans. Inform. Theory 39 (1993), 1639-1646.

[31] V. Miller, Uses of elliptic curves in cryptography, en Advances in Cryptology-CRYPTO ' 85, Springer Lecture Notes in Computer Science 218 (1986), 417-426.

[32] Miracl, disponible en: `http://indigo.ie/~mscott/`.

[33] C. Munuera, J. Tena, *Codificación de la Información*, Secret. Publicaciones, Univ. Valladolid (1997).

[34] A.M. Odlyzko, Discrete logarithms: the past and the future, Designs, Codes and Cryptography, 19, 129-145 (2000).

[35] Pari-gp, `http://www.parigp-home.de`.

[36] J. Pastor, M.A. Sarasa, *Criptografía digital*, Prensas Univ. Zaragoza (1998).

[37] E.A. Poe, El escarabajo de oro, en Narraciones Completas, Aguilar (1964).

[38] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Comm. of the ACM 21 (1978), 120-126.

[39] R. Rivest, R. Silverman, Are 'Strong' primes needed for RSA?, Cryptology ePrint Archive, Report 2001/007 (2001), en `http://eprint.iacr.org`.

[40] R. Schoof, Elliptic curves over finite fields and the computation of square roots mod p, Math. Comp., 44 (1985), 483-494.

[41] A. Silverberg, Open questions in arithmetic algebraic geometry, IAS/Park City Math. Ser. 9, Amer. Math. Soc. (2001).

[42] J.H. Silverman, *The arithmetic of elliptic curves*, Springer-Verlag (1986).

[43] J.H. Silverman, The xedni calculus and the elliptic curves discrete logarithm problem, Designs, Codes and Cryptography 20 (2000), 5-40.

[44] J.H. Silverman, J. Suzuki, Elliptic curve discrete logarithms and the index calculus, en Advances in cryptology-ASIACRYPT '98, Springer Lecture Notes in Computer Science 1514 (1998), 5-40.

[45] J.H. Silverman, J. Tate, *Rational Points on Elliptic Curves*, Springer (1992).

[46] S. Singh, *The Code Book*, Fourth Estate, London (1999).

[47] D. Stinson, *Cryptography, Theory and Practice*, 2nd. Ed., Chapman & Hall/CRC (2002).

[48] H.C.A. van Tilborg, *Fundamentals of Cryptology*, Kluwer (2000).

[49] W. Trappe, L.C. Washington, *Introduction to Cryptography with Coding Theory*, Prentice Hall (2002).

[50] A. Wiles, The Birch and Swinnerton-Dyer conjecture, en `http://www.claymath.org/prizeproblems/birchsd.pdf`.

José Luis Gómez Pardo
Departamento de Álxebra
Universidade de Santiago
15782 Santiago de Compostela
correo electrónico: `pardo@usc.es`