Implementación GnuPG con Curvas Elípticas

Sergi Blanch i Torné[1] y Ramiro Moreno Chiral[2]

[1] Escola Politècnica Superior, Universitat de Lleida. Spain.
d4372211@alumnes.eps.udl.es
[2] Departament de Matemàtica. Universitat de Lleida. Spain.
ramiro@eps.udl.es

Resumen Usando normas y estándares ampliamente criptoanalizados, se ha colaborado con el Open Source realizando un módulo que añade los algoritmos ECElGamal y ECDSA a la aplicación criptográfica GnuPG. Presentamos una descripción de tal módulo y de cómo se ha insertado en el GnuPG.

1. Introducción

Aunque para las implementaciones *software* las curvas elípticas no parecen muy necesarias, para sistemas con capacidad limitada, los criptosistemas con curvas elípticas resultan ideales. Para equiparar la seguridad de una clave RSA de 1024 bits, sólo necesitaremos claves de 192 bits con curvas elípticas definidas sobre un cuerpo finito. Pero hay más: para igualar la seguridad de una clave RSA de 7680 bits, muy lenta en los cálculos, y que podríamos considerar como "de paranoia militar", sólo necesitaremos 384 bits con las curvas elípticas y un tiempo de cómputo parecido al RSA de 2048 bits.

Sin embargo, con la intención de colaborar con la comunidad criptográfica mediante un pequeño grano de arena, hemos realizado una de esas implementaciones *poco necesarias*. Ya que, aunque se sepa que un criptosistema es muy robusto no nos sirve de mucho si no tenemos acceso práctico a él. Quiénes conocen las curvas elípticas saben que son robustas, ágiles y resistentes a ataques. Por ello nos ha parecido necesario añadir los criptosistemas basados en curvas elípticas a alguna aplicación de propósito general que nos permita a los usuarios usarlos y, hacerlo en convivencia con los criptosistemas que utilizamos ahora.

Dos ideas han orientado nuestro trabajo. Por un lado, buscar una serie de algoritmos basados en estándares ampliamente criptoanalizados. La comunidad ha trabajado largamente en la norma *P1363* del IEEE y en el estándar *FIPS PUB 186-2* del NIST. Nunca podemos estar seguros de que estos estándares no tengan vulnerabilidades, pero es más fácil confiar en ellos que en algo completamente nuevo.

Por otro, usar *software* libre. La balanza de nuestra decisión se inclinó (como no podía ser menos) por el programa llamado *GnuPG (Gnu Privacy Guard)*. Basado en *OpenPGP* y hecho con rigor, se puede considerar competente con el mítico y conocido *Pretty Good Privacy (PGP)*, del que proviene. Actualmente este *software* mantiene tres ramas de trabajo: *stable* (1.2.x), *developers* (1.3.x), y

experimental (1.9.x). Para introducir los criptosistemas basados en curvas elípticas nos pareció idónea la versión para desarrolladores, la 1.3.5.

2. Estándares y Algoritmos

Planteamos en esta sección una breve descripción de los estándares en los que se ha basado la implementación realizada, así como algunas razones del porque de la opción elegida.

2.1. Norma P1363 del IEEE

En la búsqueda de un estándar que se pueda considerar suficientemente criptoanalizado, nos detuvimos a estudiar la norma P1363, publicada por el IEEE en 1999. El proyecto para llegar hasta la norma final empezó en 1994, pero no se incluyó el soporte para curvas elípticas hasta dos años después, en 1996. Inicialmente estandarizaba la algorítmica para los criptosistemas basados en los problemas clásicos del *IFP, Integer Factorization Problem* y del *DLP, Discrete Logarithm Problem*, hasta añadir los criptosistemas basados en el *ECDLP, Elliptic Curve Discrete Logarithm Problem*.

El hecho de extraer nuestra algorítmica de una norma como ésta, no sólo nos da seguridad, ya que lleva largo tiempo en estudio, sino que, a la vez, tenemos una buena descripción de las operaciones matemáticas en los distintos niveles que podemos requerir: aritmética de multiprecisión entera y la operatividad en el grupo de puntos de una curva elíptica, $E(\mathbb{F})$.

En nuestra implementación hemos usado principalmente la sección 7 y la A.9 de la P1363. En la primera se describen las primitivas necesarias para tratar el problema del logaritmo discreto sobre curvas elípticas: primero los parámetros del dominio de la curva, que definen el *setup* del criptosistema: permiten inicializarlo; luego los pares de claves, pública y privada, del usuario, para acabar con la resolución de problemas basados en el protocolo de intercambio de claves Diffie-Hellman.

En la sección 9 del Anexo A, se nos da una visión sobre los algoritmos que podemos utilizar para realizar los operaciones con los puntos de la curva elíptica. Y otras muchas más posibilidades de las que realmente necesitaremos. Podremos elegir sobre qué cuerpo finito definir el grupo de puntos de la curva: si sobre $\mathbb{F}_{2^m}$ o sobre $\mathbb{F}_p$. En esta implementación se eligió usar el segundo tipo, trabajando así en el grupo de puntos $E(\mathbb{F}_p)$. Y la otra elección que había que hacer, era si usar coordenadas afines o proyectivas para la representación de los puntos de las curvas elípticas. Por una parte tenemos el plano afín, $\mathbb{A}_2(\mathbb{F}_p) = \mathbb{F}_p \times \mathbb{F}_p$, en el que no existe el punto del infinito, $\mathcal{O}$, y por otro lado tenemos el plano proyectivo $\mathbb{P}_2(\mathbb{F}_p)$ compuesto por ternas de elementos de $\mathbb{F}_p$ (dos dimensiones con tres coordenadas) en el que sí está definido el punto del infinito de la curva, $\mathcal{O}_E$, y que permite sumar y doblar puntos sin tener que realizar cocientes en el cuerpo base. Estas dos características del plano proyectivo lo hacen especialmente atractivo para usar en una implementación que pretende ser ágil. Optamos, pues, por la representación proyectiva para los puntos de $E(\mathbb{F}_p)$.

2.2. FIPS PUB 186-2 del NIST

El *National Institute for Standars and Technology* (en adelante NIST), dependiente del departamento de Comercio de EEUU, publicó al año siguiente de la aparición de la norma P1363 del IEEE, un estándar sobre el que debían basarse los sistemas de firma digital del gobierno de ese país.

En el Apéndice 6 de este documento, aparece detallada una recomendación de *setup*, en la que se enumeran los parámetros de una lista de curvas elípticas y los órdenes de los cuerpos base sobre las que hay que definirlas (esos datos están también recogidos en el reciente libro [HMV04, p. 262]). Con la misma confianza que la que tuvieron en su día las *S-boxes* del criptosistema simétrico DES, se puede seguir la recomendación de esta norma. Nuestra opción fué seleccionar las curvas recomendadas en el Apéndice de la FIPS PUB 186-2 para prefijarlas como *setup* de la implementación realizada.

Esta decisión conlleva distintas ventajas a la vez que inconvenientes. Empezando por las ventajas, hay que decir que disponer de los parámetros de una curva elíptica, sobre todo de su cardinal, es mucho mejor que calcularlos en tiempo de ejecución: haría falta un cómputo intensivo. Digamos una desventaja: al prefijar la curva elíptica del criptosistema se pierde aleatoriedad en el *setup*. Mejorar las desventajas, sin perder ninguna de las posibles ventajas, es lo que se tratará de hacer en un futuro, y será comentado con más detalle al final de este documento.

3. Algunos detalles de la implementación

Esta sección está dedicada a expones con brevedad algunos aspectos básicos del cómo de la implememtación: la definición del *setup* y los algoritmos de cifrado/descifrado y firma/verificación que se han usado.

3.1. Setup del criptosistema

Hemos hablado de los parámetros de una curva elíptica, el conjunto de datos que constituye el *setup* del criptosistema. Matemáticamente podemos describirlo como la sextupla

$$\mathcal{S} = \{p, a, b, G, n, h\}$$

El primer elemento de la tupla, p, define el módulo del cuerpo base sobre el que se realizarán las operaciones aritméticas básicas. El segundo y tercer elemento, a y b, son los parámetros de una curva elíptica en la *forma reducida de Weierstraß*,

$$y^2 = x^3 + ax + b,$$

definida sobre un cuerpo finito primo $\mathbb{F}_p$, con $p > 3$. Como cuarto elemento de la tupla, tenemos el punto generador, G, del subgrupo cíclico de puntos sobre el que se plantea el ECDLP, es decir, $\langle G \rangle \subseteq E(\mathbb{F}_p)$. La relación entre el cardinal del grupo $\langle G \rangle$ y el número de puntos de la curva elíptica, es decir el orden del

grupo total $E(\mathbb{F}_p)$, está definida por los dos últimos elementos de la tupla $\mathcal{S}$ del *setup*, siendo n el orden del generador, i.e., $|\langle G\rangle| = n$, donde, siguiendo la P1363 del IEEE, n ha pasado un test de primalidad de Rabin con 50 bases, i.e., la probabilidad de que no sea primo es de $1/2^{100}$. Y su relación con el cardinal de la curva está dada por el *cofactor* h,

$$|E(\mathbb{F}_p)| = hn.$$

En lo que sigue vamos a usar la nomenclatura de [HMV04]: $a \in_R [1, m]$, $a, m \in \mathbb{Z}_{>0}$, con la que indicaremos que a es un valor aleatorio entre 1 y m del mismo tamaño en bits que m. Así, la clave secreta d de los criptosistemas basados en el ECDLP será un valor $d \in_R [1, n-1]$, tal que al sumar el generador consigo mismo d veces obtengamos otro punto Q que no sea el punto en el infinito $\mathcal{O}_E$,

$$Q = d \cdot G \neq \mathcal{O}_E.$$

ALGORITMO 1 (GENKEYPAIR)

INPUT: Curva Elíptica($E(\mathbb{F}_p)$), y punto generador G.
OUTPUT: Clave secreta d, clave pública Q.

Generar la clave secreta $d \in_R [1, n-1]$;
Calcular $Q = d\dot{G}$;
si $Q = \mathcal{O}_E$ **entonces**
  Ir al principio del algoritmo
fin si
Return(d,Q);

En cuanto a las estructuras de datos básicas implementadas, la clave pública contendrá tanto el *setup* como el punto Q, y la clave secreta contendrá a la clave pública junto con el valor d,

$$\begin{aligned} \texttt{ECC\_setup} &= \mathcal{S} = \{p, a, b, G, n, 1\} \\ \texttt{ECC\_public\_key} &= \{\mathcal{S}, Q\} \\ \texttt{ECC\_secret\_key} &= \{\mathcal{S}, Q, d\} \end{aligned}$$

Nótese que se ha tomado el cofactor $h = 1$, para así magnificar el tamaño del subgrupo cíclico $\langle G\rangle$: todas las curvas del NIST tienen cofactores "bajos", concretamente son $h \leq 4$. Obviamente, para el cofactor elegido, es $\langle G\rangle = E(\mathbb{F}_p)$, es decir, usamos curvas elípticas con grupo de puntos cíclico. Está elección hace al grupo $\langle G\rangle$, en el que en realidad se plantea el ECDLP, perfectamente resistente al ataque Pohlig–Hellman, ya que su orden es primo. Las curvas elegidas son también resistentes a los ataques por *isomorfismos*, que se basan en la construcción de algún isomorfismo entre el grupo de puntos $\langle G\rangle$ y el grupo $\mathbb{F}_p^+$ (para el caso en que $\mid E(\mathbb{F}_p) \mid = p$), o el $\mathbb{F}_{p^k}^*$, en alguna extensión de índice $1 < k \leq 20$ del cuerpo $\mathbb{F}_p$ (cuando $\text{mcd}(n, p) = 1$), pero en las curvas del NIST no son viables ninguno de estos ataques.

3.2. Algoritmos de cifrado/descifrado ECElGamal

Los algoritmos descritos en estos dos apartados se han sacado del norma P1363 del IEEE, como ya se ha dicho, respetando simbología y nomenclatura. Añadimos un apartado dedicado a los ataques mediante *lattices* de los sistemas tipo ElGamal[1] (ver [Ngu04], entre otros).

ALGORITMO 2 (ENCRYPT ($A \to B$))

INPUT: Clave pública $pkey_B$ y texto plano numérico z.
OUTPUT: Punto resultante R, cifra c.

Generar una clave de sesión $k \in_R [1, (pkey_B.n) - 1]$;
$P = k \cdot pkey_B.Q$ /*$Q_B = d_B \cdot G_B$; $P = k \cdot d_B \cdot G_B$*/
$R = k \cdot pkey_B.G$
$c = z \cdot P_x$
si (size(c) < size($pkey_B.n$) **entonces**
  ir al principio del algoritmo;
fin si
Return(R,c);

ALGORITMO 3 (DECRYPT)

INPUT: Punto resultante R, cifra c, y clave privada $skey_B$.
OUTPUT: Texto plano numérico z.

Recibido (R,c)
$P = skey_B.d \cdot R$ /*$P = d_B \cdot k \cdot G_B$*/
$z = c \cdot P_x^{-1} \pmod{p}$.
Return(z);

3.3. Algoritmos de firma/verificación ECDSA

ALGORITMO 4 (SIGN ($A \to B$))

INPUT: Clave secreta $skey_A$ y resumen del mensaje #hash.
OUTPUT: Par de números (r, s) /*Se cumple $0 < r, s < skey_A.n$*/.

Generar una clave se sesión $k \in_R [1, (skey_A.n) - 1]$;
$I \leftarrow k \cdot (skey_A.G)$;
$i \leftarrow I_x$;
$r \leftarrow i \pmod{skey_A.n}$;
si $r = 0$ **entonces**
  ir al principio del algoritmo;
fin si
$s \leftarrow k^{-1} \cdot (\#hash + (skey_A.d) \cdot r) \pmod{skey_A.n}$;
si $s = 0$ **entonces**
  ir al principio del algoritmo;

[1] Agradecmos al *referee* su indicación al respecto.

fin si
Return (r, s).

ALGORITMO 5 (SIGN VERIFICATION)

INPUT: Calve pública ($pkey_A$), resumen del mensaje (#hash), y par de números (r, s).
OUTPUT: Valor booleano de aceptación o repudio.

Verificar $(r, s) \in [1, (pkey_A.n) - 1]$;
$h \leftarrow s^{-1} \pmod{pkey_A.n}$;
$h_1 \leftarrow (\#hash) \cdot h \pmod{pkey_A.n}$;
$h_2 \leftarrow r \cdot h \pmod{pkey_A.n}$;
$Q \leftarrow h_1 \cdot (pkey_A.G) + h_2 \cdot (pkey_A.P)$;
si $Q = \mathcal{O}_E$ **entonces**
  repudiar
si no
  $i = Q_x \pmod{pkey_A.n}$;
  si $i = r$ **entonces**
    aceptar
  si no
    repudiar
  fin si
fin si

3.4. Ataques mediante *lattices* a los criptosistemas tipo ElGamal

Durante bastante tiempo GPG permitía el uso de firma o de cifrado+firma con claves tipo ElGamal sobre grupos $\mathbb{F}_p^*$. Hasta que, en un anuncio en Internet el 27 de noviembre de 2003, el equipo de desarrollo del GPG hizo público que tales claves eran débiles y quedaban descartadas del sistema. La razón de tal debilidad estriba en que la clave secreta d y la clave de sesión o *nonce* k se elgían, por razones de eficiencia, de tamaños mucho menores que p: para cada tamaño posible de p se tenía un "umbral" para ambos valores, denotado como q_{bit} y tabulado en la llamada *tabla de Wiener*, de magnitudes siempre menores que $1/4$ del tamaño de p y al crecer p mucho menores de $1/4$: por ejemplo, para p's de 3840 bits es $q_{bit} = 296$. Y por eso eran susceptibles de un ataque por *lattices* o redes sobre $\mathbb{Z}^n$. Concretamente, una búsqueda del *vector más próximo* en la red es eficiente en esos casos, en los que $k = 3/2q_{bit}$, y, mucho más, si también la clave secreta $d \ll p$: de ser así una firma basta para encontrar d en un PC en menos de un segundo (cf. [Ngu04]). Pero todo ello no es aplicable a la implementación que hemos realizado en el módulo ECC sobre el GnuPC. Veamos algunas razones.

1. Nuestra implementación sólo usa el algoritmo ECC-ElGamal para cifrado: la firma se realiza siempre con ECC-DSA, que es un algoritmo resistente al ataque que nos ocupa, según se dice en [Ngu04].

2. Hemos evitado expícitamente el paso por los umbrales q_{bit} de la tabla de Wiener: tanto las claves de sesión, k, como las claves secretas, d, son del tamaño de p. Eso se ha querido indicar más arriba con el símbolo $\in_R$.
3. Todas las inversiones de los algoritmos utilizados se realizan $\mod p$ o $\mod n$, así que no hay necesidad de imponer a las claves de sesión ninguna restricción adicicional, por lo que no se merma su aleatoriedad.

4. Integración en el GnuPG del módulo ECC implementado

Como guía para todos los desarrolladores de *software* criptográfico, y basándose en el primero de este tipo de propósito general llamado *PGP*, se diseño el *OpenPGP* que se está convirtiendo en un estándar . El *PGP* o *Pretty Good Privacy*, escrito por Phil Zimmermann fijó unas convenciones y unas estructuras de datos, desarrolladas posteriormente en un *RFC* o *Request For Comments* que, con número 2440 y bajo el título *OpenPGP Message Format*, hizo público todo lo necesario para realizar una implementación compatible. Quedaba un vacío en la implementación de *software* criptográfico bajo Open Source, que llenó el proyecto iniciado por Werner Koch al que llamó *Gnu Privacy Guard*, (*GnuPG*).

Una vez decidido y visto el programa sobre el que realizaremos el módulo de curva elíptica, aún nos queda por decidir el lenguage con que programarlo. Pudiendo utilizar un lenguage orientado a objetos como `C++`, con el cual podríamos obtener un código bastante natural, pulido y próximo al utilizado en la algorítmica de las normas. Sin embargo, los actuales módulos del GnuPG estan realizados en `C`. Para tomar una decisión sobre el lenguaje resultó muy útil consultarlo a todos los desarrolladores del software. La respuesta fue muy clara: la decidión pasa por valorar hasta que público se desea llegar. Si se quiere realizar un módulo para uso privado, usar objetos facilita la programación, pero, si se desea que el módulo pueda llegar a trascender y ser criptoanalizado por la comunidad el lenguaje debe ser `C`. Esto, junto con la integración en el entorno GnuPG, nos da todos los motivos para escoger este último lenguaje.

El módulo programado, ha seguido un esquema de programación descendente que nos permite modular a la vez que facilita el posterior trabajo de integración en el programa GnuPG. Existe un primer nivel de programación que realiza la función de interficie con el resto del programa y que utiliza las estructuras de datos propias del programa general. En un segundo nivel de descenso se traducen esas estructuras y se sirven hacia el primer nivel o hacia este segundo, en el que ya aparecen ya las estructuras de datos propias del módulo implementado. Podríamos considerar las funciones en estos dos niveles como repetidas, y su principal diferencia estriba en las estructuras que usa.

Ya en los dos niveles inferiores, tercero y cuarto, se mezclan funciones criptográficas con otras específicamente matemáticas que podrían haberse separado e insertado en la librería matemática del programa. Debido a que este módulo se desarrolló en un contexto académico, y sin renunciar a mejoras futuras,

optamos por dispersar al mínimo las funciones programadas. El tercer nivel descendente describe las funciones que realizan operaciones con los puntos de la curva, mientras que en el cuarto están las que son usadas con las coordenadas de dichos puntos, es decir, la aritmética sobre el cuerpo base $\mathbb{F}_p$. Quedan algunos fragmentos de código que no caben en el esquema descrito, ya que tienen un uso especialmente auxiliar. En este grupo se engloban desde funciones parciales hasta otras similares a los constructores y destructores de la programación orientada a objetos, en su versión en lenguaje `C`.

Una vez programado el módulo ECC e integrado mediante las *Autotools*[2] en el GnuPG, hubo que modificar algunas partes del software para que éste lo use e interaccione con él. La primera modificación que realizamos fué integrar en la generación de claves del GnuPG las nuevas claves ECC. Por otro lado, al realizar las otras operaciones (cifrado-descifrado, firma-verificación) el programa realiza comprobaciones, entre ellas si el algoritmo indicado es válido para realizar la operación. Finalmente, el GnuPG usa un identificador que aparece al listar las claves: una letra descriptiva del algoritmo, justo después de la medida de dicha clave. Para el módulo ECC elegimos la descriptiva E.

5. Tests de pruebas

Con el módulo ECC operativo, valoramos su comportamiento comparando sus tiempos de ejecución del cifrado y firma con curvas elípticas con los correspondientes estándares del GnuPG: ElGamal y la firma DSA, sobre el grupo multiplicativo $\mathbb{F}_p^*$. También con los algorimos RSA de cifrado y firma. Las pruebas se hicieron sobre las distintas operaciones del módulo, a saber, generación de claves, cifrado y descifrado, firma y verificación y listado de claves, usando un total de 26 claves distintas. Para aquellas que requieren una interacción, se utilizaron los ficheros ejemplo que se proporcionan con el GnuPG para el chequeo de su funcionamiento: cuatro ficheros binarios y tres de texto de variadas longitudes. Las condiciones iniciales son iguales para todos, salvo el algoritmo público escogido y el nivel de entropía que pueda tener en ese momento el ordenador.

El computador utilizado para realizar y estudiar dichos chequeos es un Intel Xeon dual a $3,06$GHz (bus frontal de 533MHz) con 8GB de RAM, con el Sistema Operativo Red Hat Linux Advanced Server 3, utilizando el compilador *gcc 2.96*.

La tabla nos muestra los tiempos obtenidos según las longitudes de las claves. Los cinco niveles elegidos, por orden creciente, son los correspondientes a los niveles de seguridad para los siguientes sistemas de cifrado simétrico: SKIPJACK (80 bits), 3DES (112), AES-small (128), AES-medium (192) y AES-large (256), según tabla[3] en [HMV04, p. 19]. Se detalla el tiempo consumido en las distintas operaciones, distinguiendo entre tiempo de ejecución en *modo usuario* o en *modo kernel*. El promedio del total es la medida que nos permite comparar el

[2] Herramienta de *software* libre que facilita enormemente la creación de grandes proyectos

[3] También se encuentran datos similares en el Anexo D de la P1363 y en el FIPS PUB 186-2.

Procesos		ECC					ELG/DSA (RSA)							
		192	224	256	384	521	1024		2048		4096		7680	8192
`GenKey`	user	1,22	1,44	1,78	4,36	10,62	48,48	(3,35)	701,00	(47,92)	7739,19	(543,88)	60438,40	78134,38
	kernel	0,19	0,23	0,24	0,34	0,33	0,16	(0,11)	0,22	(0,12)	0,35	(0,12)	1,12	0,33
	total	1,41	1,67	2,02	4,70	10,95	48,64	(3,46)	701,22	(48,04)	7739,54	(544,00)	60439,52	78134,71
	media	0,05	0,06	0,08	0,18	0,42	1,87	(0,13)	26,97	(1,85)	297,68	(20,92)	2324,60	3005,18
`Encrypt`	user	3,88	4,29	5,06	11,74	25,80	6,10	(0,99)	20,29	(0,96)	76,48	(1,46)	516,34	595,85
	kernel	0,63	0,91	0,97	0,97	1,17	0,61	(0,46)	0,60	(0,61)	0,50	(0,52)	0,61	0,57
	total	4,51	5,20	6,03	12,71	26,97	6,71	(1,45)	20,89	(1,57)	76,98	(1,98)	516,95	596,42
	media	0,03	0,03	0,03	0,07	0,15	0,04	(0,01)	0,12	(0,01)	0,43	(0,01)	2,84	3,28
`Decrypt`	user	3,61	4,57	5,51	15,08	35,31	5,59	(1,78)	17,69	(7,71)	109,60	(48,36)	372,03	424,71
	kernel	0,65	0,63	0,87	1,27	1,31	0,35	(0,44)	0,51	(0,40)	0,42	(0,37)	0,62	0,50
	total	4,26	5,20	6,38	16,35	36,62	5,94	(2,22)	18,20	(8,11)	110,02	(48,73)	372,65	425,21
	media	0,02	0,03	0,04	0,09	0,20	0,03	(0,01)	0,10	(0,04)	0,61	(0,27)	2,05	2,34
`Sign`	user	3,34	4,21	5,03	13,24	30,01	4,17	(1,52)	12,92	(7,32)	43,81	(47,87)	140,89	153,50
	kernel	0,59	0,53	0,66	0,84	1,46	0,39	(0,28)	0,29	(0,34)	0,41	(0,29)	0,41	0,34
	total	3,93	4,74	5,69	14,08	31,47	4,56	(1,80)	13,21	(7,66)	44,22	(48,16)	141,30	153,84
	media	0,02	0,03	0,03	0,08	0,17	0,03	(0,01)	0,07	(0,04)	0,24	(7,14)	0,78	0,85
`Verify`	user	2,38	3,26	4,15	11,36	26,30	3,55	(0,46)	11,07	(0,50)	37,59	(0,91)	120,02	130,91
	kernel	0,93	0,81	0,86	1,09	1,44	0,69	(0,64)	0,68	(0,64)	0,69	(0,71)	0,62	0,59
	total	3,31	4,07	5,01	12,45	27,74	4,24	(1,10)	11,75	(1,14)	38,28	(1,62)	120,64	131,50
	media	0,02	0,02	0,03	0,07	0,15	0,02	(0,01)	0,07	(0,01)	0,21	(0,01)	0,66	0,72
`ListKey`	user	0,13	0,16	0,19	0,53	1,29	0,18	(0,03)	0,66	(0,03)	2,13	(0,08)	6,93	7,51
	kernel	0,06	0,06	0,10	0,11	0,09	0,09	(0,05)	0,05	(0,07)	0,09	(0,03)	0,06	0,07
	total	0,19	0,22	0,29	0,64	1,38	0,27	(0,08)	0,71	(0,10)	2,22	(0,11)	6,99	7,58
	media	0,01	0,01	0,01	0,03	0,05	0,01	(0,00)	0,03	(0,00)	0,09	(0,00)	0,27	0,29
Total Procesos		17,61	21,10	25,42	60,93	135,13	70,36	(10,11)	765,98	(66,62)	8011,26	(644,60)	61598,05	79449,26
Media Procesos		0,03	0,03	0,04	0,085	0,19	0,33	(0,03)	4,56	(0,33)	49,87	(4,73)	388,53	502,11

Cuadro 1. Tiempos de ejecución en segundos

speedup de los diferentes criptosistemas y tamaños de clave. Destacamos algunas comparaciones:

- Los tiempos promedios para la generación de claves son mucho mejores en el módulo ECC que en cualquier otro criptosistema de los tabulados.
- Cualquier operación en el módulo ECC es más rápida que en ElGamal/DSA sobre $\mathbb{F}_p^*$.
- En general, las operaciones tipo RSA son más rápidas que las del módulo ECC, sin embargo, al aumentar el tamaño de las claves, la ventaja se decanta por el ECC en descifrado y firma.

6. Desarrollo futuro

Como casi siempre ocurre en los desarrollos de *software*, nuestro código no es perfecto: contiene puntos conocidos que podemos considerar *bugs*, sin olvidar que además pueden existir otros que no conozcamos. El *bug* que contiene nuestra implementación se centra en un error producido al intentar acceder a la clave secreta en el caso que ésta haya sido protegida por una *frase de paso*. Las claves funcionan correctamente en caso de no protegerlas con esa frase. Pero, muy probablemente, durante la creación de la clave (y de su cifrado simétrico con la frase de paso) el proceso no se realiza correctamente. Estamos estudiando la

posibilidad de que el fallo esté en la recuperación de la clave antes de ser usada para el descifrado o la firma digital.

A pesar de parecer una limitación, el hecho de haber fijado los parámetros del *setup* del criptosistema obedece al criterio de facilitar posibles ampliaciones por el que hemos optado en esta implementación. O bien fijamos esos parámetros, o bien los generamos cuando el usuario desee crearse un par de claves. Esta última opción, genial desde el punto de vista criptoanalítico, tiene un coste muy elevado que se cargaría sobre el usuario final. Hay un término medio entre ambas soluciones: mantener una tabla de *setups candidatos* precalculados por parte de los desarrolladores. Si este número de setups precalculados es lo suficientemente grande, se acercará mucho a generarlo en línea sin sufrir la penalización en tiempos de cómputo.

La actual estructura del módulo, que incluye todas las funciones, facilita su lectura y la visión del flujo de instrucciones. En la dirección hacia una versión definitivamente integrada en el *software*, y que siguiese la estructura del resto de módulos, que sólo contienen las funciones criptográficas, sería muy ventajoso extraer las funciones matemáticas e incluirlas directamente en la librería de números grandes que utiliza dicho programa. Con este paso, se ampliarían las funcionalidades directamente sobre la librería *gcrypt* a la vez que las funcionalidades del GnuPG.

Cabe resaltar la existencia de distintas bases sobre las que montar un criptosistema de curva elíptica: en la actual implementación se ha usado cuerpos finitos modulo un primo suficientemente grande, $\mathbb{F}_p$, pero también hay otros cuerpos que podríamos haber utilizado. Principalmente destacan los cuerpos finitos de característica 2 con un grado de extensión m suficientemente grande, $\mathbb{F}_{2^m}$. Ampliar el módulo a los cuerpos de característica 2 es uno de lo objetivos a alcanzar en un futuro por su rapidez operacional.

Referencias

[1991] PGP Message Exchange Formats. 1996 August.

[2440] OpenPGP Message Format. 1998 November

[3156] MIME Security with OpenPGP. 2001 August.

[3278] Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS). 2002 April.

[NIST] FIPS PUB 186-2, Digital Signature Standard (DSS), U.S.Department of Commerce/National Institute of Standards and Technology. 2000 January 27.

[P1363] IEEE P1363/D13 (Draft Version 13) Standard Specifications for Public key Cryptography. 1999 November 12.

[PKCS#1] PKCS#1 v2.1: RSA Cryptography Standard. RSA Laboratories. 2002 June 14.

[HMV04] D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[ECDSA] D. Johnson, A. Menezes, S. Vanstone, *The elliptic Curve Digital Signature Algorithm (ECDSA)*. Dept. of Combinatorics & Optimization, University of Waterloo, Certicom, Canada.

[Men93] Alfred Menezes, *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993

[MOV97] Alfred J, Menezes, Paul C.van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, first edition, 1997. http://cacr.math.uwaterloo.ca/hac/

[Mill86] V. Miller, *Uses of elliptic curves in cryptography* Advances in Cryptology - Crypto '85, Lecture Notes in Computer Science, n218 (1986), Springer-Verlag, pp. 417–426.

[Ngu04] P. Q. Nguyen, *Can we trust cryptographic software? Cryptographic flaws in GNU Privacy Guard v1.2.3* Proceedings of Eurocrypt'04, Lecture Notes in Computer Science, 3027 (2004), Springer, pp. 555–570.